

Modeling Planning, Control, and Scheduling of Cyber-Physical Systems for Reinforcement Learning

Mirco Maurice Theile

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology
der Technischen Universität München zur Erlangung eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz: Prof. Dr.-Ing. Matthias Althoff

Prüfende der Dissertation:

1. Prof. Dr. Marco Caccamo
2. Prof. Dr. Samarjit Chakraborty
3. Prof. Miroslav Pajic, Ph.D.

Die Dissertation wurde am 16.04.2024 bei der Technischen Universität München eingereicht
und durch die TUM School of Computation, Information and Technology am 25.06.2024
angenommen.

Abstract

Modern cyber-physical systems (CPS) are becoming increasingly complex, motivating data-driven techniques such as reinforcement learning (RL). Despite significant advances in recent years, the real-world applicability of RL is still very limited for many reasons, including exorbitant data requirements and safety concerns. This thesis is motivated by the development of a long-endurance solar unmanned aerial vehicle (UAV), the UIUC-TUM Solar Flyer, and the opportunities for improving its performance and capabilities through RL. Therefore, it investigates how to apply RL to three branches of CPS: Planning, Control, and Scheduling.

For planning, the thesis explores path planning for UAVs, with applications in coverage path planning and path planning for wireless data harvesting in grid-world scenarios. The problems are solved using map-based observations of the environment and RL agents learning to generalize over different targets, battery states, and number of agents. In control, the problem of real-time online learning is addressed through a cloud-edge training architecture. Furthermore, the thesis discusses an action mapping approach that can allow the integration of safety models into the learning process in continuous action spaces. It further shows how to enable it by learning to generate all feasible actions. The scheduling problem is addressed by formulating task-sets as directed acyclic graphs (DAGs) and creating static schedules by adding edges to the DAGs.

In conclusion, this thesis aims to answer the following questions for RL in CPS: (i) how to make the problems observable to enable generalization, (ii) how to include existing knowledge to accelerate training, and (iii) how to make sure that the solutions of RL are valid and safe. It paves the way for applying RL to real-world CPS, among others, enabling the future development of an intelligent version of the solar UAV.

Contents

Abstract	i
Contents	iii
List of Publications	vii
1 Introduction	1
1.1 Cyber-Physical Systems and Reinforcement Learning	1
1.2 UIUC-TUM Solar Flyer	3
1.2.1 Development	4
1.2.2 Results	7
1.3 Example Application: UAV-based Digital Agriculture as a Service	9
2 Reinforcement Learning Background	13
2.1 Markov Decision Process	13
2.2 Reinforcement Learning	14
2.2.1 Foundational Equations	15
2.2.2 Q-Table Learning	16
2.3 Deep Reinforcement Learning	16
2.3.1 Deep Learning	17
2.3.2 Function Approximation in RL	17
2.3.3 Parameterization and Reparameterization Trick	18
2.3.4 Challenges of Function Approximation in RL	18
2.4 Deep Reinforcement Learning Algorithms	18
2.4.1 Deep Q-Learning	19
2.4.2 Double Deep Q-Learning	20
2.4.3 Deep Deterministic Policy Gradient	21
2.4.4 Twin Delayed Deep Deterministic Policy Gradient	21
2.4.5 Soft Actor-Critic	22
2.4.6 Proximal Policy Optimization	24
2.5 Summary	25

3	Unmanned Aerial Vehicles – Fixed-Wing Aircraft	27
3.1	Trajectory Estimation for Geo-Fencing Applications on Small-Size Fixed-Wing UAVs	28
3.2	uavAP: A Modular Autopilot Framework for UAVs	36
4	Reinforcement Learning for Map-based Path Planning	61
4.1	UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning	62
4.2	UAV Path Planning for Wireless Data Harvesting: A Deep Reinforcement Learning Approach	69
4.3	UAV Path Planning using Global and Local Map Information with Deep Reinforcement Learning	76
4.4	Multi-UAV Path Planning for Wireless Data Harvesting With Deep Reinforcement Learning	85
5	Reinforcement Learning for Real-world Control Challenges	101
5.1	Cloud-Edge Training Architecture for Sim-to-Real Deep Reinforcement Learning	102
5.2	Learning to Generate All Feasible Actions	111
6	Reinforcement Learning for Graph-based Task Scheduling	127
6.1	Latency-Aware Generation of Single-Rate DAGs from Multi-Rate Task Sets	128
6.2	Edge Generation Scheduling for DAG Tasks Using Deep Reinforcement Learning	142
7	Discussion	159
7.1	Summary	159
7.2	The Law of the Hammer	160
7.3	Future Work	163
Appendices		165
A	Reuse Statements	167
A.1	Trajectory Estimation for Geo-Fencing Applications on Small-Size Fixed-Wing UAVs	168
A.2	uavAP: A Modular Autopilot Framework for UAVs	169
A.3	UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning	170
A.4	UAV Path Planning for Wireless Data Harvesting: A Deep Reinforcement Learning Approach	171
A.5	UAV Path Planning using Global and Local Map Information with Deep Reinforcement Learning	172

A.6 Multi-UAV Path Planning for Wireless Data Harvesting with Deep Reinforcement Learning	173
A.7 Cloud-Edge Training Architecture for Sim-to-Real Deep Reinforcement Learning	174
A.8 Learning to Generate All Feasible Actions	175
A.9 Latency-Aware Generation of Single-Rate DAGs from Multi-Rate Task Sets	176
A.10 Edge Generation Scheduling for DAG Tasks Using Deep Reinforcement Learning	177

List of Publications

Core Publications

1. M. Theile, S. Yu, O. D. Dantsker, and M. Caccamo, “Trajectory estimation for geo-fencing applications on small-size fixed-wing UAVs,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1971–1977, IEEE, 2019
2. M. Theile, O. Dantsker, R. Nai, M. Caccamo, and S. Yu, “uavAP: A modular autopilot framework for UAVs,” in *AIAA AVIATION 2020 FORUM*, p. 3268, 2020
3. M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, “UAV coverage path planning under varying power constraints using deep reinforcement learning,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1444–1449, IEEE, 2020
4. M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, “UAV path planning using global and local map information with deep reinforcement learning,” in *2021 20th International Conference on Advanced Robotics (ICAR)*, pp. 539–546, IEEE, 2021
5. M. Theile, D. Bernardini, R. Trumpp, C. Piazza, M. Caccamo, and A. L. Sangiovanni-Vincentelli, “Learning to generate all feasible actions,” *IEEE Access*, vol. 12, pp. 40668–40681, 2024

Additional Publications

6. M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, “Latency-aware generation of single-rate dags from multi-rate task sets,” in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 226–238, IEEE, 2020
7. H. Bayerlein, M. Theile, M. Caccamo, and D. Gesbert, “UAV path planning for wireless data harvesting: A deep reinforcement learning approach,” in *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2020
8. H. Bayerlein, M. Theile, M. Caccamo, and D. Gesbert, “Multi-UAV path planning for wireless data harvesting with deep reinforcement learning,” *IEEE Open Journal of the Communications Society*, vol. 2, pp. 1171–1187, 2021

9. H. Cao, M. Theile, F. G. Wyrwal, and M. Caccamo, “Cloud-edge training architecture for sim-to-real deep reinforcement learning,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9363–9370, IEEE, 2022
10. B. Sun, M. Theile, Z. Qin, D. Bernardini, D. Roy, A. Bastoni, and M. Caccamo, “Edge generation scheduling for dag tasks using deep reinforcement learning,” *IEEE Transactions on Computers*, 2024

Chapter 1

Introduction

1.1 Cyber-Physical Systems and Reinforcement Learning

Cyber-physical systems (CPS) are ubiquitous in every aspect of our modern society, encompassing a range from smartphones and intelligent vehicles to water kettles with controllable temperatures. CPS can be generally defined as computer systems that control or monitor physical components, such as electrical or mechanical hardware. This interplay allows software to compensate for hardware errors or imprecision; however, it also necessitates careful consideration of hardware constraints, particularly in terms of real-time requirements. In 2012, Kim and Kumar [11] summarized the progress of CPS in various fields. Since then, the field of CPS has evolved and expanded so rapidly and broadly that it is impossible for a single document to encapsulate all of its recent advancements comprehensively.

A popular application branch of CPS is the area of mobile robots, such as unmanned aerial vehicles (UAVs), autonomous ground vehicles (AGVs), autonomous surface vehicles (ASVs), or autonomous underwater vehicles (AUVs). The critical challenges of planning, control, and scheduling are paramount in these applications. Planning involves generating efficient paths, considering obstacle avoidance, and optimizing for speed and energy consumption. Control is crucial for executing these plans accurately, requiring real-time adaptation to environmental changes and maintaining stability under varying conditions. Scheduling focuses on efficiently allocating resources to different tasks. While crucial in multi-robot systems for coordinated operation, it is also instrumental for ensuring the real-time performance of the complex required onboard computations through task scheduling. Together, these elements are fundamental for the autonomy and effectiveness of mobile robots.

Traditionally, the solutions to these challenges can be divided into two directions: exact methods and heuristics. The former focuses on precisely modeling the problem and finding the optimal solution by solving the resulting optimization problem. The latter focuses on finding relatively simple algorithms that solve the underlying problem sufficiently well. For planning, an exact method is model predictive control (MPC) [12], in which the evolution of the system and environment is typically modeled in detail to predict future states and optimize them by adapting the sequence of control inputs. On

the other hand, heuristics often abstract away the low-level details, focusing on finding paths through probabilistic methods such as rapidly exploring random trees (RRTs) [13] or probabilistic road maps (PRMs) [14]. In control, exact methods model the system and find optimal control actions, e.g., state feedback control. However, heuristics such as PID controllers [15] are often used, as they tend to offer sufficient performance in many applications without complicated modeling. In scheduling, exact methods often require solving NP-hard optimization problems, making heuristics based on priorities, such as rate monotonic (RM) or earliest deadline first (EDF) popular [16]. In some cases, these heuristics even yield the optimal solution.

As CPS are becoming increasingly complex, exact methods are becoming either too complex to model perfectly or infeasible to solve optimally, and algorithmic heuristics no longer offer sufficient solutions. In recent years, the focus has shifted towards data-driven techniques to combat the ever-growing complexities. The idea is to learn relationships between vast quantities of data instead of modeling the problem or hand-writing algorithms to determine the relationships. These techniques have been outstandingly successful in tasks of computer vision [17] and, recently, for natural language [18, 19]. In these fields, labeled data is used to train deep neural networks to infer the correct outputs from given inputs. In computer vision, this labeled data can be the classifications for different images, and in natural language, it is often the next word in a given text. For the tasks of planning, control, and scheduling, usually labeled data does not exist, as optimal solutions to a problem are unknown, and human data is often suboptimal or expensive to collect.

The data-driven technique suitable for these challenges is reinforcement learning (RL) [20]. In contrast to the supervised learning techniques used in computer vision and natural language processing, RL does not require labeled data. Instead, RL agents learn policies through interactions with the environment. By exploring various actions in different states, RL agents aim to maximize a cumulative reward quantified as the return, often expressed through a value function. The strength of RL, specifically deep RL in which the agent is expressed through a neural network, is its ability to generalize to unseen scenarios, an essential element for CPS. RL agents are typically trained in a simulation, which is effectively a complex forward model of the system. The resulting policy is not necessarily optimal but delivers good performance even in complicated tasks. Therefore, RL could be classified as a methodology that trains through interactions with a complex model, yielding learned heuristics to perform well in challenging tasks.

While RL offers significant advantages for addressing the challenges of complex CPS, it also presents notable challenges, especially concerning safety and data efficiency. One of the primary concerns with RL is ensuring the safety of the system during both the training and deployment phases. Since RL learns through trial and error, it is required to violate safety constraints numerous times to learn to avoid them. Therefore, RL agents are typically trained in simulation and deployed to the real physical system after training. While training in simulation mitigates the safety risks during training, the difference between simulation and the real world, referred to as the *sim-to-real gap* [21], typically leads to significant decrements in performance. Additionally, given that the RL agent is usually a black box neural network, proving that a trained agent will never violate safety



Figure 1.1: The UIUC-TUM Solar Flyer aircraft equipped with solar panels flying autonomously in an agricultural environment [22].

constraints is challenging or potentially impossible.

This thesis addresses some of the challenges when applying RL to problems in CPS. A primary focus lies in formulating the problems to allow RL agents to find generalizing policies and ensure that the agents' solutions are safe and feasible. To further motivate the challenges, the following section summarizes a CPS we developed, which offers an application case study to illustrate the challenges in planning, control, and scheduling.

1.2 UIUC-TUM Solar Flyer

In recent years, UAVs have surged in popularity, primarily fueled by the desire to apply these aircraft to precision farming, infrastructure and environment monitoring, surveying and mapping, surveillance, search and rescue missions, and weather forecasting. These applications predominantly require continuous data collection, especially from visual sensors. Additionally, the collected data must be processed to acquire the mission results. Typically, there are three approaches to handling the collected data [23].

In the first approach, the data is processed *offline* and *offboard*. The data is collected and stored on the UAV during the mission and downloaded to a ground station for processing after landing [24–27]. This approach is very cost-effective since data storage is inexpensive. However, the UAV cannot adapt the mission based on real-time stimuli, making it suboptimal or sometimes infeasible for long-endurance missions. The second approach is to process the data *online* but *offboard*. For this approach, the collected data is continuously streamed down to a ground station, where the data is analyzed in real-time to remotely adjust mission plans and objectives [28, 29]. While this approach enables the UAV to react to real-time stimuli, high-bandwidth communication requires significant power and depends on communication infrastructure. Therefore, this approach

is problematic for many mission profiles, especially long-distance, long-endurance missions.

The UIUC-TUM solar flyer is designed for the third approach: collect and process the data *online* and *onboard*, only relaying the results to a ground station if necessary. This approach is ideal for long-endurance autonomous missions, offering closed-loop control without requiring unreliable and costly communication. Before our solar flyer, this approach had only been used in high-cost, classified aircraft [30] or for low-complexity objectives such as “follow me” mode [31,32]. The solar flyer is equipped with solar panels to combat the high demand for energy for propulsion, actuation, and computation. Previously, there had been many existing aircraft that use solar panels and can sustain continuous flight [33–39]. However, they cannot perform significant onboard computations beyond automating flight. Moreover, the solar flyer only comprises commercial-off-the-shelf (COTS) components. All the other long-endurance solar aircraft utilized custom airframe designs with many custom components (e.g., single-application propellers and gearboxes) [33,34]. Using only COTS components reduces the cost of the aircraft, which in turn increases accessibility to the community.

1.2.1 Development

This section briefly summarizes the development of the UIUC-TUM Solar Flyer, discussing its hardware, autopilot, and digital twin [40]. More detailed information on the autopilot and a geo-fencing application are discussed in Chapter 3.

Hardware

The UIUC-TUM Solar Flyer was designed to continuously acquire and process high-resolution visible and infrared imagery, focusing on optimizing its airframe, propulsion system, energy system, and avionics. The aircraft’s development involved a systematic trade study to select a high-efficiency, commercially available airframe with a wingspan of at least three meters [41]. F5 Models and Top Model CZ were considered, and the final choice was the F5 Models Pulsar 4E Pro, as shown in Figure 1.1. It offers a large wingspan, lightweight construction, and efficient design, incorporating a kevlar pod, carbon fiber, and balsa wood reinforced with carbon and kevlar fiber. Modifications were made to facilitate the integration of computational devices and solar arrays.

The propulsion system of the Solar Flyer was specifically designed considering the limited onboard energy. A comprehensive tool was developed to optimize the propulsion system, considering various propeller and motor combinations for specific mission profiles [42]. The propulsion system’s optimization required detailed performance data of potential motors and propellers obtained through wind tunnel testing of 40 Aero-Naut CAM carbon fiber folding propellers. The final selection, Model Motors AXi 480/1380 motor and Aero-Naut CAM 12x8 propeller, resulted in a 15% increase in propulsion efficiency compared with a common baseline combination. The energy system of the Solar Flyer comprises Gallium arsenide solar arrays, an MPPT charge controller, and a lithium-ion battery, ensuring efficient energy collection, storage, and distribution [23].

The avionics of the Solar Flyer are centered around a commercially available flight control and data acquisition system, the AIVolo FDAQ+FC [43], integrating various

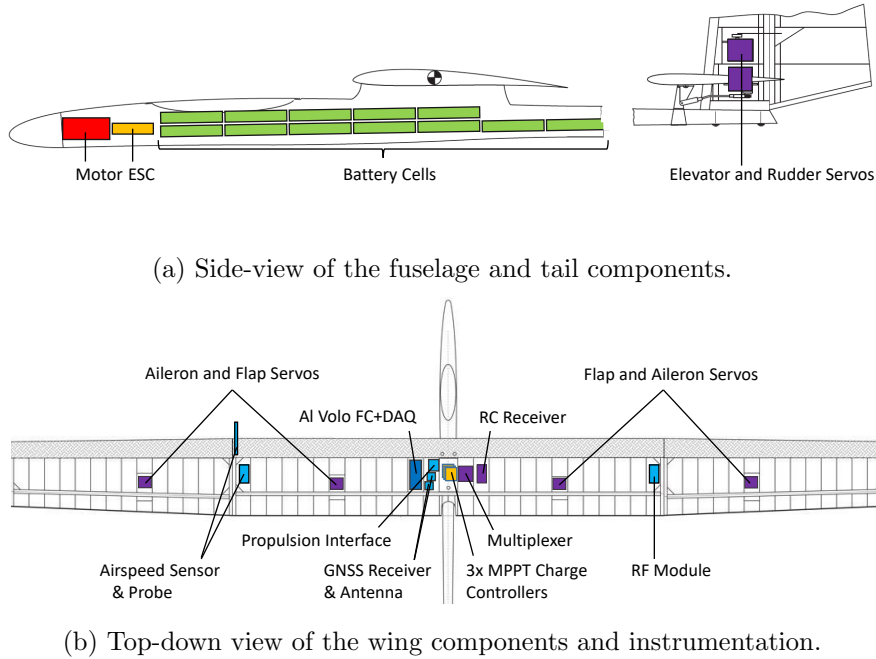


Figure 1.2: Components and instrumentation in different parts of the Solar Flyer [44].

sensors and modules for effective operation and data collection. The aircraft additionally features an RF module and various sensors, including an inertial measurement unit, GNSS, airspeed sensor, and motor sensor. The layout of the Solar Flyer’s fuselage and wing components, as shown in Figure 1.2, shows the strategic placement of these systems to maintain balance and efficiency.

Autopilot

To fully control the solar flyer’s autonomy, we developed our own autopilot framework, *uavAP* [2]. The *uavAP* is a modular framework that enables rapid prototyping of planning, control, and communication modules. The modules are executed in separate processes to allow software failure isolation, as shown in Figure 1.3. Additionally, a Watchdog process synchronizes the modules on startup and monitors the state of the processes. An API provides an interface for external processes to supply sensor data and read out action commands. The autopilot implements a control stack to follow waypoint-based trajectories autonomously. Additionally, it offers the ability to automate maneuver sequences that can be used to characterize the aircraft dynamics [40, 45]. A safety layer detects when a boundary violation is imminent and aborts the maneuvers preemptively [1].

The ground station, *uavGS*, was designed to monitor, command, and tune the autopilot. The ground station interface has a modular layout that can be customized for

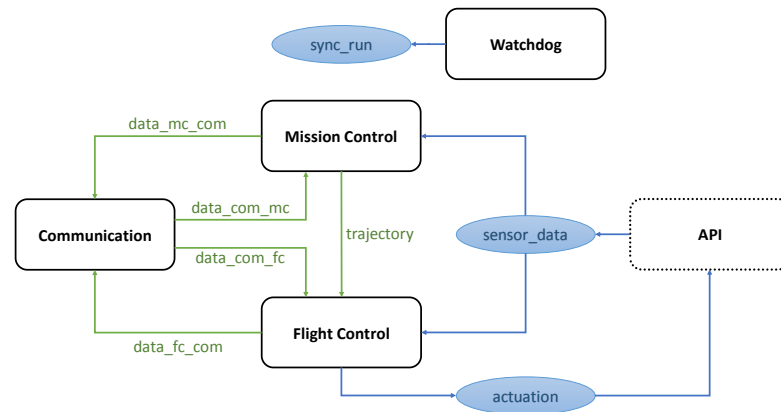


Figure 1.3: Processes and their default connections in uavAP [2].

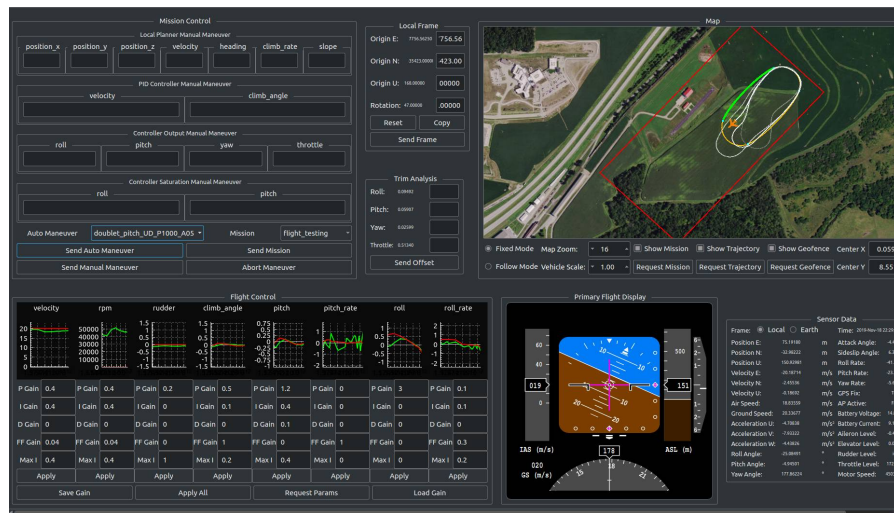


Figure 1.4: Ground station design for monitoring the autopilot and tuning parameters [2].

each use case. Figure 1.4 shows a layout used for flight testing for system identification [40]. The top right shows an overhead map with the aircraft's current position, its planned trajectory in yellow with the current path section in green, and safety bounds in red. The bottom right shows a primary flight display and the current sensor data of the UAV. The bottom left shows the status of all the PID controllers with their target values in red and the current value in green. Additionally, it allows for changing the PID parameters and sending them to the UAV for tuning. The top left is specifically designed for the mission of system identification. It allows to override target values in the control stack to perform specific maneuvers. Additionally, it allows one to select from a list of predefined maneuver sequences that can be used to determine airplane characteristics.

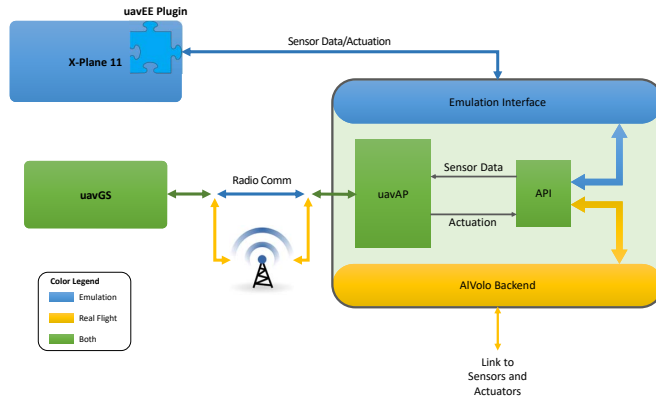


Figure 1.5: Architecture of the UAV emulation environment – uavEE [46].

Digital Twin

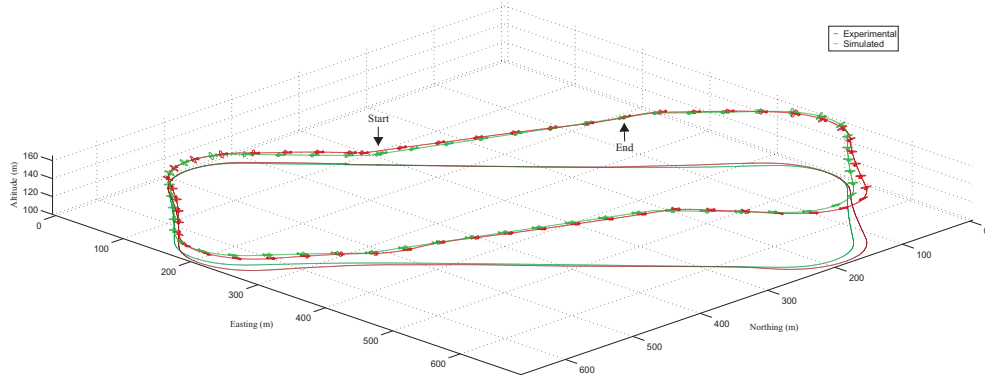
To test the autopilot software and computational hardware before deploying it on a real aircraft, we developed the UAV emulation environment, uavEE [46]. The general architecture of uavEE is shown in Figure 1.5, in which the colors of the components indicate if they are present in emulation, real flight, or both. In the center is the uavAP autopilot with its API, which exchanges sensor data and control commands with either an AlVolo Backend [43] in real flight or an emulation interface during emulation. While the AlVolo Backend aggregates sensor data from physical sensors and sends control commands to the corresponding actuators and motors, the emulation interface communicates with a plugin in a flight simulator. The flight simulator commonly in use is X-Plane 11 [47], for which uavEE provides a plugin. To test the ground station uavGS, in emulation, the connection between the autopilot and ground station is replaced with serial or UDP connections. Depending on the stages of development, the different components can run on one device to only test the software or on different devices to test the computational hardware.

As a primary concern of a long-endurance solar UAV is energy consumption, we further developed a power model that estimates the propulsion power from the current state of the aircraft [46, 48]. The power model has been further augmented in [49] by estimating the solar power generation depending on the UAV position, attitude, time of day, and day of the year. In [50], these models were used to simulate the power consumption and generation during long-endurance flights.

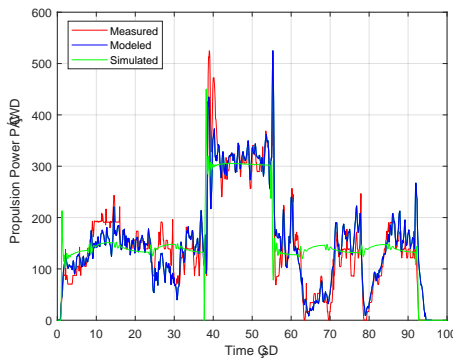
1.2.2 Results

uavEE accuracy

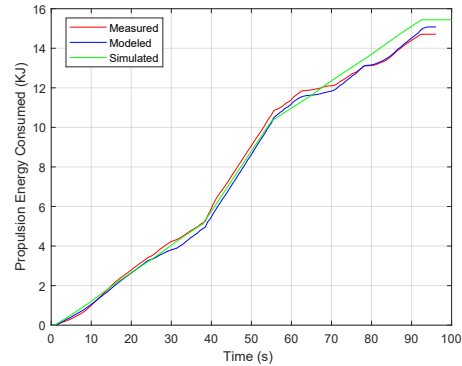
Initial flight tests were conducted with an Avistar Elite trainer aircraft, and the corresponding simulation used the FS One[®] flight simulator [51]. A comparison of trajectories between the real flight and the simulation (as shown in Figure 1.6a) indicates that the trajectory-following behavior in the uavEE closely mirrors that of the actual flight. This



(a) Flight path of an experimental (red) and simulated (green) flight. For reference, the aircraft is plotted every 2 seconds.



(b) Propulsion power



(c) Energy consumed

Figure 1.6: Comparison of experimentally measured (red), experimentally modeled (blue), and simulated (green) results using the propulsion power model [46].

similarity suggests that the emulation environment effectively replicates the real-world flight path of the aircraft, encompassing various maneuvers such as turns, climbs, and straight flights.

Regarding power consumption, focusing on a specific interval (100s to 200s ticks), compares the measured power during the flight (red) with the power model [46, 48] estimations (blue) derived from inertial and GPS data. The findings, illustrated in Figures 1.6b and 1.6c, reveal a near congruence between the estimated and measured power for the given trajectory, despite some initial differences attributed to the wind-induced angle of attack.

Further, when applying the same power model to simulated data, the power and energy curves (green in the figures) exhibit less noise compared to the measured power. Although there is a slight overestimation of consumed energy in the simulator for the specific trajectory extract, these disturbances generally cancel out over the total energy consumption. This indicates that the uavEE can accurately estimate power and energy consumption, paralleling real-flight conditions.

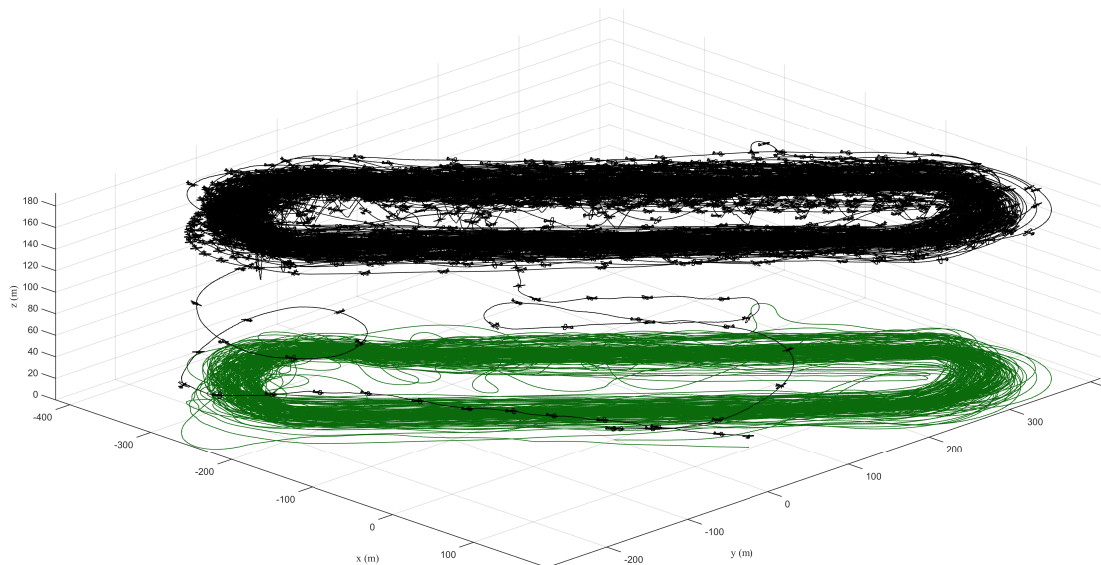


Figure 1.7: Trajectory of the UIUC-TUM Solar Flyer during an 8-hour flight under cloudy and gusty conditions [52].

Overall, the results demonstrate that the power model yields highly accurate results, affirming the effectiveness of the emulation environment in replicating real flight conditions in terms of both trajectory adherence and power consumption.

Long-endurance solar flight

As discussed in [52], on 31 August 2020, the UIUC-TUM Solar Flyer completed an 8-hour flight under non-ideal conditions, taking off at 7:55 AM and landing at 3:59 PM. The flight took place with variable weather, featuring partly cloudy to cloudy skies and winds ranging from 8 to 24 km/h (with gusts up to 32 km/h), shifting from east to south-west. The aircraft followed a 650×150 meter race track flight pattern at an altitude of 100 meters and an airspeed of 11 m/s, successfully completing 188 laps. Despite the challenging conditions, this flight resulted in an effective coverage area of 14.1 km^2 , equating to a coverage rate of 1.8 km^2 per hour. The trajectory of this flight is illustrated in Figure 1.7, depicting the aircraft's position at 5-second intervals.

1.3 Example Application: UAV-based Digital Agriculture as a Service

This thesis uses a conceptual example of a solar UAV to illustrate and motivate various reinforcement learning solutions. The example application focuses on vegetation monitoring, a significant aspect of smart farming or digital agriculture due to its role in

data gathering [53]. UAVs equipped with sensors are valuable in this domain, capable of delivering timely, high-resolution data [54]. Numerous studies have explored UAVs for vegetation, livestock monitoring, and forestry, with comprehensive surveys in [55–57]. Further research has investigated UAVs as agrochemical distribution systems for specific crops such as artichokes [58] and cotton [59].

Consider utilizing a long-endurance solar aircraft with onboard computation capabilities, such as the UIUC-TUM Solar Flyer, for applications such as vegetation monitoring. The optimal use case of such a UAV would not be that every farmer purchases and operates its own, but instead that a company offers vegetation monitoring as a service to different farmers in an area. In that scenario, farmers could decide asynchronously which parts of their plot they want inspected, if any. After the aircraft inspected the area, the farmer would get an immediate detailed analysis and could apply measures according to the needs of the plants.

This use case results in challenging requirements for the company operating the UAV. While the UAV operates in the same region consisting of landing zones, obstacles, and no-fly zones, the area coverage target changes daily. This yields a coverage path planning (CPP) problem for which the path with the lowest energy consumption needs to be found to maximize utilization. Additionally, the UAV must fly reliably during varying weather conditions, maximizing power efficiency. Finally, the direct vegetation analysis feedback for the farmers requires the computation to be handled while computing the optimal path and control command. To summarize, the main challenges in this application include:

1. Creating a path for the UAV such that the sensors cover the changing target zones, the UAV avoids obstacles and designated no-fly zones, and ensures a return to a landing zone when the battery is low.
2. Efficiently controlling the UAV to follow a planned trajectory under varying environmental conditions, minimizing power consumption, and maximizing solar power generation.
3. Managing all computations for planning, control, and data processing online and onboard on a resource-constrained platform.

Each of these challenges presents a potential area for applying reinforcement learning. The first is an instance of coverage path planning (CPP), a proven NP-hard problem [60] that poses scalability issues for long-duration missions such as regional vegetation monitoring. The second challenge arises from the UAV model’s dependency on environmental factors like wind, solar intensity, and thermals, for which model-free reinforcement learning can offer an efficient solution. The third challenge involves scheduling on potentially heterogeneous multi-core platforms, another NP-hard problem with scalability issues as the number of tasks on the UAV increases.

This thesis delves into these challenges, identifying and addressing the underlying issues. In Chapter 2, the reinforcement learning background is provided to give a general understanding and some details for the most popular algorithms. Chapter 3 elaborates on the various challenges in CPS by discussing the development of an autopilot framework

and a geo-fencing application for fixed-wing UAVs. For the UAV path planning problems, Chapter 4 focuses on formulating them to allow an RL agent to generalize. The problems of grid-world coverage path planning and path planning for data harvesting are solved by expressing the environment as a map and training an RL agent to find the shortest path, generalizing over different problem instances. It also shows that the map-based environment observation can be used for multi-agent scenarios. For control, two problems are investigated in Chapter 5. The first centers on the challenge of pretraining an agent in a simulation and continuing its training on the actual hardware. It establishes a cloud-edge training architecture that is evaluated for an inverted pendulum system. The second is how to avoid unsafe control commands in a manner that benefits the learning agent. It introduces an action mapping scheme for filtering unsafe actions. Chapter 6 addresses the scheduling problem by formulating different scheduling problems as directed acyclic graph (DAG) problems. The chapter then offers a novel solution for DAG scheduling that trains an RL agent to iteratively add edges to the graph until it is statically schedulable and real-time guarantees can be made. Finally, Chapter 7 discusses the contributions, provides a general overview of the applicability of RL in CPS, and elaborates on future research directions.

Chapter 2

Reinforcement Learning Background

This chapter provides the mathematical background for reinforcement learning (RL). It starts by introducing various forms of Markov Decision Processes (MDPs) that are used to model systems for RL. In the following sections, the fundamentals of RL are introduced, followed by an introduction to deep RL. The chapter finishes with a detailed description of the most common RL algorithms.

2.1 Markov Decision Process

When formulating problems for RL, they usually need to be expressed as Markov decision processes (MDPs) [61]. An MDP describes a system through states and actions, where the state evolves according to a transition function. The system is an MDP if it has the Markov property, for which the transition function can only depend on the current state and the applied action and be independent of past states. All MDP variants consist of a state space \mathcal{S} , an action space \mathcal{A} , and a reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that yields rewards based on the state and action of the system. In some cases, the reward function also depends on the state of the system after applying the action, i.e., $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$. The system evolves according to a transition function, which can be deterministic as $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ or stochastic as $P : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathcal{S})$, where $\mathcal{P}(\mathcal{S})$ stands for the space of all distributions over \mathcal{S} , yielding a deterministic or stochastic MDP. If the state of the system is not directly measured but rather indirectly observed, the system becomes a partially observable MDP (POMDP). In that case, an observation space Ω is defined, and a probabilistic observation function $O : \mathcal{S} \rightarrow \mathcal{P}(\Omega)$ maps the state to a distribution over possible observations. All MDP variants typically have a discount factor γ that determines the importance of future rewards compared with current rewards. Table 2.1 provides a summary of these MDP variants, and the corresponding symbols are explained in Table 2.2. For simplicity of notation, the following focuses on fully observable stochastic MDPs.

The objective in an MDP is to maximize the cumulative discounted reward called *return*, often abbreviated with an R . However, to avoid confusion with the reward or reward function, we directly utilize the RL term *value function* V . To maximize the value function, a different action must be applied to each state encountered in the MDP.

MDP Tuple	Type
$(\mathcal{S}, \mathcal{A}, R, T, \gamma)$	Deterministic MDP
$(\mathcal{S}, \mathcal{A}, R, P, \gamma)$	Stochastic MDP
$(\mathcal{S}, \mathcal{A}, R, P, \Omega, O, \gamma)$	POMDP

Table 2.1: Common types of Markov decision processes.

Symbol	Mapping	Description
\mathcal{S}	-	State space
\mathcal{A}	-	Action space
R	$\mathcal{S} \times \mathcal{A}(\times \mathcal{S}) \rightarrow \mathbb{R}$	Reward function
T	$\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$	Deterministic transition function
P	$\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathcal{S})$	Probabilistic transition function
Ω	-	Observation space
O	$\mathcal{S} \rightarrow \mathcal{P}(\Omega)$	Probabilistic observation function
γ	-	Discount factor $\in [0, 1] \subset \mathbb{R}$

Table 2.2: Elements in the different MDP types.

Therefore, a *policy* π is introduced that maps the current state to an action. The policy can either be deterministic as $\pi : \mathcal{S} \rightarrow \mathcal{A}$ or stochastic as $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, with $\mathcal{P}(\mathcal{A})$ being the space of all distributions over the action space. For generality, we assume that the policy is stochastic in the following. With the policy, the value function can be defined as

$$V^\pi(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid a_t \sim \pi(\cdot | s_t), s_{t+1} \sim P(\cdot | s_t, a_t), s_0 = s \right], \quad (2.1)$$

which is the expectation of the cumulative discounted sum of rewards when applying actions according to the policy, the system evolving according to the transition function, and the initial state being s . Consequentially, the objective of an MDP is to find a policy that maximizes the value function for all states as

$$\pi^* = \arg \max_{\pi} V^\pi(s), \quad \forall s \in \mathcal{S}. \quad (2.2)$$

RL is a tool to find that policy by interacting with the MDP. In RL, it is common to simplify the temporal notation from s_t and s_{t+1} to s and s' . Whenever possible, this notation is adopted.

2.2 Reinforcement Learning

Reinforcement learning (RL) aims to learn a policy to maximize the value function in an MDP through interactions with the environment. Figure 2.1 gives a visualization of this interaction loop. It shows the RL *agent* observe a state s and perform an action a , which yields a reward r and advances the environment to state s' . To find a good policy,

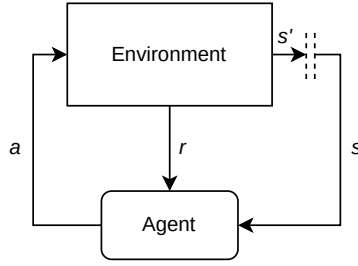


Figure 2.1: Sketch of the interaction loop in RL.

the agent needs to explore various trajectories, for which it needs to find a compromise between exploration and exploitation. The following details the foundational equations and discusses different strategies.

2.2.1 Foundational Equations

The objective in RL is to maximize the value in (2.1), which is difficult to maximize directly. The most common method to approach this maximization is to formulate a Q-value

$$Q(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} \left[\max_{a' \in \mathcal{A}} Q(s', a') \right] \quad (2.3)$$

that aims to explicitly define the value of each specific action at a given state. It is a recursive function adding the immediate reward obtained for performing action a in state s with the γ -discounted expectation of the highest Q-value in the next state. Here, the expectation is over the stochastic transition function. The corresponding policy is simply using the action with the highest Q-value, i.e.,

$$\pi(s) = \arg \max_{a \in \mathcal{A}} Q(s, a). \quad (2.4)$$

In continuous action spaces, the definition of the Q-value in (2.3) is challenging, as the maximal Q-value at the next state is hard to find exactly. However, since the policy aims to maximize the Q-value, it can be used directly in its definition as

$$Q^\pi(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} [Q^\pi(s', \pi(s'))], \quad (2.5)$$

making the Q-value dependent on the policy. To facilitate exploration, the policy is often explicitly stochastic, yielding the most general Q-value definition as

$$Q^\pi(s, a) = R(s, a) + \gamma \mathbb{E}_{a' \sim \pi(\cdot | s'), s' \sim P(\cdot | s, a)} [Q^\pi(s', a')]. \quad (2.6)$$

This Q-value definition includes the definition in (2.5) for deterministic policies and the definition in (2.3) for deterministic policies in discrete action spaces. Using the definition of the Q-value in (2.6), the value function from (2.1) can be written as

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot | s)} [Q^\pi(s, a)], \quad (2.7)$$

providing the relationship between the Q-value and the value.

For some algorithms, it is necessary to estimate an *advantage* function, which indicates the advantage or disadvantage when performing a specific action at a given state compared to the expectation of the current policy. It is simply defined as

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s), \quad (2.8)$$

i.e., the difference between the Q-value and the value. Given these equations, the simplest approach for RL can be introduced: Q-table learning.

2.2.2 Q-Table Learning

The simplest approach to Q-learning is to learn the Q-value as a table containing the current estimate of the Q-value for each state-action pair. The values in the table can then be updated after each interaction with the environment according to

$$Q(s, a) \leftarrow (1 - \alpha_t)Q(s, a) + \alpha_t \left(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') \right), \quad (2.9)$$

in which s is the state before the interaction, a is the action applied, $r = R(s, a)$ is the reward received, $s' \sim P(\cdot|s, a)$ is the next state after the interaction and α_t is a learning rate that can vary throughout the training process. In a fully deterministic environment, a learning rate of $\alpha_t = 1$ is optimal [20]. According to Watkins et al. [62], in discrete environments, the Q-table method is guaranteed to converge to the ground-truth Q-values if

1. the learning rate approaches zero throughout training with $\sum_{t=1}^{\infty} \alpha_t = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$;
2. all state-action pairs are visited infinitely often.

Practically, most environments are continuous, and the Q-table increases in size rapidly with increased dimensionality of the state and action spaces. Additionally, the Q-table cannot generalize, meaning that it cannot deduce Q-values for state-action pairs it did not observe from those it did observe. The solution to these problems is to approximate the Q-value with deep neural networks (DNNs) instead of a table, leading to deep reinforcement learning (DRL). While transitioning to deep neural networks complicates the proof of convergence, it significantly enhances applicability in diverse environments and leverages the DNNs' inherent capacity for generalization in RL.

2.3 Deep Reinforcement Learning

The idea in DRL is to approximate the relevant functions from Section 2.2.1 using universal function approximators, such as deep neural networks. The following provides the critical ideas of deep learning, followed by the general concepts of using deep learning in RL.

2.3.1 Deep Learning

In deep learning, deep neural networks (DNNs) are used as function approximators for various input-output relationships [63]. The DNNs are parameterized with trainable parameters, which are trained using backpropagation [64].

Specifically, a *dense* layer, or *fully-connected* layer can be defined as

$$\mathbf{l}_{i+1} = \sigma_i(W_i \mathbf{l}_i + \mathbf{b}_i), \quad (2.10)$$

with an input $\mathbf{l}_i \in \mathbb{R}^{n_i}$ and output $\mathbf{l}_{i+1} \in \mathbb{R}^{n_{i+1}}$. The parameters of the layer are the weight matrix $W_i \in \mathbb{R}^{n_{i+1} \times n_i}$ and a bias $\mathbf{b}_i \in \mathbb{R}^{n_{i+1}}$ and some non-linear activation function $\sigma_i : \mathbb{R}^{n_{i+1}} \rightarrow \mathbb{R}^{n_{i+1}}$. Depending on the use case, this activation function is commonly a rectified linear unit (ReLU), sigmoid, or tanh. The activation functions need to be non-linear to allow the neural network to approximate non-linear functions. For a DNN with m layers, the input \mathbf{x} and output \mathbf{y} can be included as

$$\mathbf{l}_0 = \mathbf{x}, \quad \mathbf{y} = \mathbf{l}_m. \quad (2.11)$$

These layers are commonly summarized into one function definition as

$$\mathbf{y} = f_\theta(\mathbf{x}), \quad \theta = (W_0, b_0, W_1, b_1, \dots, W_{m-1}, b_{m-1}). \quad (2.12)$$

Given a set of ground-truth data containing data points of the format $(\mathbf{x}, \mathbf{y}_{\text{gt}})$ a loss function can be defined such as a mean-squared error loss

$$L_\theta(\mathbf{x}, \mathbf{y}_{\text{gt}}) = \|f_\theta(\mathbf{x}) - \mathbf{y}_{\text{gt}}\|^2. \quad (2.13)$$

Depending on the use case, other loss functions can be used, such as cross-entropy for classification tasks. Using backpropagation [64], a gradient of the loss to the parameters can be computed, and the parameters can be updated according to

$$\theta \leftarrow \theta - \alpha \nabla_\theta L_\theta(\mathbf{x}, \mathbf{y}_{\text{gt}}), \quad (2.14)$$

such that the network approximates the relationship of the data points after multiple training steps.

To process other data types, various layers were defined, such as convolution layers [17] for images, attention layers [18] for language tokens, or recurrent long-short term memory (LSTM) cells [65] to add memory and many more. In the following, these techniques are used to approximate the fundamental functions of RL.

2.3.2 Function Approximation in RL

Depending on the MDP environment, different functions must be approximated with DNNs for DRL. If the action space is discrete, it can be sufficient to approximate the Q-value in (2.3) by processing the state as an input and yielding the Q-value estimate for each action individually at corresponding output neurons. In that case, the Q-value at a given state is directly estimated for each action, and the best one can be chosen

according to (2.4) as done in [66]. However, if the action space is continuous, this is impossible (unless the action space is discretized, e.g., Agent C51 in [67]). In that case, training two networks, an actor and a critic [68] is common. The critic estimates the value function in (2.1) or Q-value in (2.6), and the actor tries to maximize the Q-value estimate or an advantage. Since the actor is learning the policy function, which can be stochastic, it often parameterizes a parametric stochastic function.

2.3.3 Parameterization and Reparameterization Trick

Most neural networks are deterministic functions. The most common method to express a stochastic policy $\pi_\phi(a|s)$ using a neural network with parameters ϕ is to parameterize a known distribution with the network’s output. In most applications, the output of the neural network is the mean (μ) and standard deviation (σ) of a Gaussian $\mathcal{N}(\mu, \sigma^2)$. This representation significantly limits the space of possible policies, as it assumes that all necessary action policies can be represented through a Gaussian distribution. However, it has worked in many applications and is thus the de facto standard.

Given an action sample $a \sim \pi_\phi(\cdot|s)$, many RL algorithms are required to compute the gradient $\nabla_{\theta} a$ with respect to the policy’s parameters. If the policy parameterizes a Gaussian through its mean and standard deviation, the stochastic operation of the Gaussian makes the gradient computation challenging. Instead, a *reparameterization trick* is applied, in which the action is expressed through

$$a = \mu_\theta(s) + \sigma_\theta(s) \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1), \quad (2.15)$$

in which ϵ is sampled from a standard Gaussian. This trick treats the stochasticity of the policy as a constant for the gradient computation, allowing the gradient to propagate through the parameterization of the mean and standard deviation.

2.3.4 Challenges of Function Approximation in RL

In general, the first problem of function approximation in RL is that the ground truth target value required for the loss computation in (2.13) is *unknown*. Therefore, the target value needs to be estimated, commonly done using the same or similar network as the one being trained, called *bootstrapping*. When applying bootstrapping, the target value continuously changes during training, leading to a moving target problem and ultimately causing stability problems. The second problem is that the stochastic gradient descent in (2.14) assumes the data points are identically and independently distributed (iid). Since the RL agent learns from consecutive interactions with the environment, the resulting consecutive data points are strongly correlated, breaking this assumption and leading to further instabilities. The following algorithms are solving these problems in different ways, enabling the successful training of DRL agents.

2.4 Deep Reinforcement Learning Algorithms

In recent years, a variety of DRL algorithms have been developed. This section discusses the most relevant ones, with a summary in Table 2.3.

Algorithm	On/Off-policy	Action space	Full name
DQN	off	disc.	Deep Q-network
DDQN	off	disc.	Double deep Q-network
DDPG	off	cont.	Deep deterministic policy gradient
TD3	off	cont.	Twin delayed DDPG
SAC	off	cont.	Soft actor-critic
PPO	on	cont. + disc.	Proximal policy optimization

Table 2.3: Commonly used reinforcement learning algorithms and their characteristics.

2.4.1 Deep Q-Learning

In 2015, Mnih *et al.* [69] established combining three components to facilitate human-level performance with reinforcement learning in Atari games. The three components are:

1. **Function approximation using deep neural networks:** Approximating the Q-function with a neural network with parameters θ .
2. **Utilizing experience replay:** Using a buffer of experiences collected by the agent throughout training and sampling random batches for each training step broke the correlation between consecutive update steps.
3. **Using target networks for bootstrapping:** By using a target network with moving average filtered parameters of the main network, called $\bar{\theta}$, the target value for the Q-network was stabilized.

The loss function for the Q-network update is given by

$$L(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(\underbrace{Q_\theta(s, a) - (r + \gamma \max_{a' \in \mathcal{A}} Q_{\bar{\theta}}(s', a'))}_{\text{Target value}} \right)^2 \right], \quad (2.16)$$

in which the function approximation can be seen by Q_θ , the experience replay buffer as \mathcal{D} , and the target network in the target value as $Q_{\bar{\theta}}$. The target network is either updated periodically by copying over the parameters of the main network or with a moving average as

$$\bar{\theta} \leftarrow (1 - \tau)\bar{\theta} + \tau\theta, \quad (2.17)$$

with $\tau \in (0, 1]$ being the filter constant. By sampling from the replay memory, computing the gradient of the loss with respect to the parameters of the main network, and applying a gradient step to the network, the authors first achieved human-level performance in many Atari games.

Exploration

The simplest exploration strategy in discrete action spaces is ϵ -greedy exploration. In ϵ -greedy, the action applied during training is

$$a = \begin{cases} \text{uniform}(\mathcal{A}), & \text{with probability } \epsilon \\ \operatorname{argmax}_{a \in \mathcal{A}} Q_{\theta}(s, a), & \text{with probability } 1 - \epsilon, \end{cases} \quad (2.18)$$

taking a uniformly random action with probability ϵ and the greedy best action otherwise. The parameter ϵ can be set to a large value at the beginning and gradually decreased throughout training.

While very simple to implement, ϵ -greedy has significant shortcomings. The main problem is that the exploration action is independent of the state, making actions with catastrophic outcomes happen throughout training, even after the agent learned to avoid them. Another issue is that it is difficult to find the appropriate value for the decay rate ϵ , as it is highly problem-dependent.

The softmax-exploration strategy aims to address these issues by making the probability of each action dependent on the Q-value estimate, resulting in the stochastic exploration policy

$$\pi(a|s) = \frac{\exp(Q_{\theta}(s, a)/\beta)}{\sum_{a' \in \mathcal{A}} \exp(Q_{\theta}(s, a')/\beta)} \quad (2.19)$$

The benefit is that catastrophic actions with a significantly lower Q-value are chosen rarely once the agent learns their impact. While the parameter β needs to be tuned according to the difference in Q-values, it can often remain constant throughout training.

2.4.2 Double Deep Q-Learning

A problem plaguing DQN is the overestimation of the target Q-value. Intuitively this can be understood by considering Q_{θ} but also $Q_{\bar{\theta}}$ as a noisy estimate of Q. Therefore, the max operator in the target value favors actions with overestimated Q-values rather than underestimated values. Due to bootstrapping, the Q-network follows the target value, generally overestimating Q-values and reinforcing the overestimation in the next update step.

To combat this overestimation issue, in 2016 Van Hasselt *et al.* [70] introduced the following alternative loss function:

$$L(\theta) = \mathbb{E}_{(s, a, r, s') \sim \mathcal{D}} \left[\left(Q_{\theta}(s, a) - (r + \gamma Q_{\bar{\theta}}(s', a')) \right)^2 \mid a' = \operatorname{argmax}_{a' \in \mathcal{A}} Q_{\theta}(s', a') \right] \quad (2.20)$$

In that proposed loss function, the selection of the next action is conducted by the normal network Q_{θ} , but the value of that action is estimated by the target network $Q_{\bar{\theta}}$. This decoupling was enough in many cases to limit overestimation and improve the agent's learning stability and performance. For exploration, the same strategies as in DQN can be used.

2.4.3 Deep Deterministic Policy Gradient

DQN and DDQN are applicable only in discrete action spaces, where the Q-value for each action at a given state can be enumerated, and the maximal one can be explicitly selected. However, in continuous action spaces, such as those encountered in control problems within CPS—for example, torque control of electric motors—this explicit enumeration is not feasible. Therefore, the determination of the maximum Q-value for a given state is not straightforward in these scenarios. Many reinforcement learning algorithms adopt an actor-critic architecture to address this limitation in continuous action spaces. A critic-network estimates a value function in this framework, while the actor-network infers actions based on given states.

In deep deterministic policy gradient (DDPG) introduced by Lillicrap *et al.* [71] in 2015, the critic-network with parameters θ estimates the Q-value for a state-action pair, and the actor π_ϕ with parameters ϕ deterministically infers an action from a state. Like in DQN, the loss function of the critic is defined as

$$L_Q(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(Q_\theta(s, a) - (r + \gamma Q_{\bar{\theta}}(s', \pi_{\bar{\theta}}(s'))) \right)^2 \right], \quad (2.21)$$

with the action in the next state chosen by the actor's target network.

To train the actor-network, DDPG formulates the objective as

$$\max_{\phi} \mathbb{E}_{s \sim \mathcal{D}} [Q_\theta(s, \pi_\phi(s))], \quad (2.22)$$

i.e., the actor aims to maximize the Q-value estimate of the critic-network for the actor's action given states from the replay buffer. The actor is optimized using the deterministic gradient

$$\nabla_{\phi} Q_\theta(s, \pi_\phi(s)) = \nabla_a Q_\theta(s, a) \nabla_{\phi} \pi_\phi(s), \quad (2.23)$$

taking the gradient of the critic with respect to the action, followed by the gradient of the actor with respect to its parameters θ . The target networks for the actor and critic are updated like in DDQN through a moving average. Using this method, learning action policies for continuous action spaces was possible.

Exploration

In the DDPG algorithm, exploration is typically facilitated through additive noise, with the original work recommending Ornstein-Uhlenbeck (OU) noise for its mean-reverting qualities that mimic the correlated noise in physical control systems. However, recent implementations often prefer Gaussian noise for its simplicity and effectiveness across various applications. Gaussian noise provides necessary exploration without OU noise's complexity and hyperparameter tuning.

2.4.4 Twin Delayed Deep Deterministic Policy Gradient

While DDPG was a popular algorithm used for many problems with continuous action spaces, it had issues with stability and overestimation. To improve the algorithm, in 2018,

Fujimoto *et al.* established twin delayed deep deterministic policy gradient (TD3) [72]. TD3 introduced the following key changes compared to DDPG:

1. **Two critic-networks:** Employing two separate critic-networks and bootstrapping with the lower critic value to reduce overestimation bias.
2. **Additive noise in the next action:** Adding noise to the action at the next state for stabilized value estimates.
3. **Delayed training of the actor:** Updating the actor-network later and less frequently than the critic-networks to ensure policy updates are based on reliable value estimates.

Through these updates, TD3 requires six neural networks, two critics, one actor, and a target network for each. The two critics are updated according to the loss

$$L_Q(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[(Q_{\theta_i}(s, a) - y(r, s'))^2 \right], \quad i = 1, 2, \quad (2.24)$$

in which the target value is given by

$$y(r, s') = r + \gamma \min_{j \in \{1,2\}} Q_{\bar{\theta}_j}(s', \pi_{\bar{\phi}}(s') + \text{clip}(\epsilon, -c, c)), \quad \epsilon \sim \mathcal{N}(0, \sigma). \quad (2.25)$$

In this target value, clipped Gaussian noise is added to the next action and the lower of the two critic values is taken. These additions improved DDPG significantly, making TD3 the standard implementation of DDPG.

2.4.5 Soft Actor-Critic

A branch of RL called maximum entropy learning aims to solve the exploration problem by learning to balance performance and the entropy of the policy, in which entropy quantifies the stochasticity of the action. The most popular maximum entropy learning algorithm is soft actor-critic (SAC), introduced by Haarnoja *et al.* in 2018 [73].

In maximum entropy learning, the value function in (2.1) is extended to

$$V^\pi(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t \left(R(s_t, a_t) + \alpha H(\pi(\cdot | s_t)) \right) \mid a_t \sim \pi(\cdot | s_t), s_{t+1} \sim P(\cdot | s_t, a_t), s_0 = s \right], \quad (2.26)$$

by including an entropy term, where the entropy is defined as

$$H(p) = \mathbb{E}_{x \sim p} [- \log p(x)]. \quad (2.27)$$

By adding the entropy term, the value depends not only on the maximum accumulation of rewards through the policy but also on its stochasticity. This objective incentivizes the agent to be as stochastic as possible while maintaining close to optimal performance.

According to the new definition of the value function, the Q-value changes to

$$Q(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} \left[\mathbb{E}_{a' \sim \pi(\cdot | s')} [Q(s', a') - \alpha \log \pi(a' | s')] \right]. \quad (2.28)$$

In the original SAC paper [73], the authors propose to utilize an additional network with parameters ψ to approximate the soft value function

$$V_\psi(s) = \mathbb{E}_{a \sim \pi_\phi(\cdot|s)} [\mathbb{Q}_\theta(s, a) - \alpha \log \pi_\phi(a|s)] \quad (2.29)$$

based on the Q-function approximation \mathbb{Q}_θ and the policy π_ϕ . However, most implementations of SAC, among others, the authors' code [74] and a popular RL documentation website [75] approximate the soft value function with a single policy sample. This simplification leads to the following loss function for the critic:

$$L_Q(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[(\mathbb{Q}_{\theta_i}(s, a) - y(r, s'))^2 \right], \quad i = 1, 2, \quad (2.30)$$

in which the target value is computed as

$$y(r, s') = r + \gamma \left(\min_{j=1,2} \mathbb{Q}_{\bar{\theta}_j}(s', a') - \alpha \log \pi_\phi(a'|s') \right), \quad a' \sim \pi_\phi(\cdot|s'), \quad (2.31)$$

similar to the one in TD3.

The target of the policy is an energy function based on the Q-values given through

$$p(a) = \frac{\exp(\mathbb{Q}_\theta(s, a))}{Z(s)}, \quad Z(s) = \int_{\mathcal{A}} \exp(\mathbb{Q}_\theta(s, a')) da', \quad (2.32)$$

which is similar to the softmax exploration strategy in discrete action spaces. To train the policy to match this distribution, SAC uses the reverse Kullback-Leibler (KL) divergence to formulate the loss as

$$L_\pi(\phi) = \mathbb{E}_{s \sim \mathcal{D}} \left[\text{D}_{\text{KL}} \left(\pi_\phi(\cdot|s) \parallel \frac{\exp(\mathbb{Q}_\theta(s, \cdot))}{Z_\theta(s)} \right) \right]. \quad (2.33)$$

The benefit of using the reverse KL divergence is that when taking the gradient with respect to the policy parameters ϕ , the typically intractable partition function $Z_\theta(s)$ is eliminated as it is independent of ϕ and a . By reparameterizing the policy with a function

$$f_\phi(\epsilon; s) \quad (2.34)$$

the loss of the actor can be given as

$$L_\pi(\phi) = \mathbb{E}_{s \sim \mathcal{D}, \epsilon \sim \mathcal{N}} [\log \pi_\phi(f_\phi(\epsilon; s)|s) - \mathbb{Q}_\theta(s, f_\phi(\epsilon; s))], \quad (2.35)$$

in which θ can either be θ_1 or the lower Q-value of both critics is used.

A key advantage of SAC over DDPG and TD3 is that the exploration induced by the entropy of the policy is state-dependent. Therefore, the agent can decrease entropy in critical situations where some actions may lead to catastrophic outcomes, which is impossible in the DDPG algorithms. The difference between SAC and DDPG is thus analog to the difference between the soft-max and epsilon-greedy exploration strategies in DQN.

2.4.6 Proximal Policy Optimization

An alternative to the off-policy algorithms before is the proximal policy optimization (PPO) algorithm introduced by Schulman *et al.* in 2017 [76]. PPO is an on-policy algorithm, leading to a significantly different training procedure. Like DDPG, TD3, and SAC, PPO is an actor-critic algorithm, but the critic estimates the state-value function $V_\theta(s)$ instead of the state-action-value function $Q_\theta(s, a)$. The critic estimates the value of the current policy, and thus, experiences collected by other policies, such as older versions of the current policy, cannot be used for training. Therefore, instead of storing experiences collected throughout training in a replay memory, PPO collects a predefined number of interactions using the current policy, called a *rollout*. Using this rollout, the actor and critic are updated, after which it is discarded, and a new one is collected using the updated policy.

Specifically, after conducting a rollout of T interactions with the environment, recording states s_t , actions a_t , and rewards r_t an *advantage* of each action is estimated using the generalized advantage estimate [77] as

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \dots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \quad (2.36)$$

where

$$\delta_t = r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t). \quad (2.37)$$

The advantage quantifies if an action led to a better or worse outcome compared to the average performance of the policy. In this estimate, λ is a factor that balances the Monte Carlo value estimate for $\lambda = 1$ and the fully bootstrapped estimate for $\lambda = 0$ to find an appropriate variance-bias trade-off.

This advantage estimate is then used in the actor loss as

$$L(\phi) = \hat{\mathbb{E}}_t \left[\min\{\rho_t(\phi)\hat{A}_t, \text{clip}(\rho_t(\phi), 1 - \epsilon, 1 + \epsilon)\hat{A}_t\} \right], \quad (2.38)$$

as the expectation over the rollout with

$$\rho_t(\phi) = \frac{\pi_\phi(a_t|s_t)}{\pi_{\phi_{\text{old}}}(a_t|s_t)} \quad (2.39)$$

as the ratio between the currently trained actor and the actor used to interact with the environment to collect the rollout. The clip function clips the first value between its second and third argument. Clipping the probability ratio of the trained policy to the rollout policy prevents the policy from deviating too much during training on one rollout. It keeps the current policy “proximal” to the old policy, giving PPO its name. An additional loss on the policy’s entropy can be added to facilitate exploration and prevent the policy from collapsing into a deterministic policy.

The critic is trained to estimate the expected value of the actor. The target of the critic can be set to the actual discounted cumulative sum of rewards

$$\hat{V}_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{T-t} r_{T-1} + V_\theta(s_T), \quad (2.40)$$

which is bootstrapped only at the end of the rollout. Alternatively, the target can be set to

$$\hat{V}_t = \hat{A}_t + V_\theta(s_t), \quad (2.41)$$

which is a popular implementation, often yielding better performance. The reason may be that the total cumulative sum is strongly biased towards one action sequence by the actor, which may not represent its average performance. The alternative version in (2.41) incorporates intermediary estimates of the value function, reducing the bias.

2.5 Summary

After formulating problems as MDPs, DRL can be used to find a policy to maximize the value functions that reflect the problems' objectives. Even though DRL does not offer guarantees for finding an optimal policy, it is a promising tool that can potentially find better solutions than hand-crafted heuristics or model-based techniques. While the basic idea of RL is decades old, DRL has only been enabled in the last decade through advances in computing, with rapid development in algorithms and applications. The tool DRL is available to a large community worldwide through open-access publications of algorithms and techniques and publicly available code repositories.

DRL has been successfully applied in ever-increasingly complex environments, starting from Atari games [69] to Go [78] and other board games [79], and very complex video games such as Dota [80] and Starcraft [81]. In the area of CPS, it has also been applied to, e.g., UAV attitude control [82], dexterous in-hand manipulation [83], and locomotion of four-legged robots [84], and many more. However, significant challenges remain to overcome for the widespread adoption of DRL as a tool for problems in CPS and other fields. Among these challenges are the enormous requirement for interaction data, lack of safety assurance during and after training, and general explainability of decision-making. The works within this thesis aim to address these challenges to make DRL more applicable to CPS.

Chapter 3

Unmanned Aerial Vehicles – Fixed-Wing Aircraft

3.1 Trajectory Estimation for Geo-Fencing Applications on Small-Size Fixed-Wing UAVs

Reference

M. Theile, S. Yu, O. D. Dantsker, and M. Caccamo, “Trajectory estimation for geo-fencing applications on small-size fixed-wing UAVs,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1971–1977, IEEE, 2019
DOI: <https://doi.org/10.1109/IROS40897.2019.8967579>

Abstract

The steadily increasing popularity of Unmanned Aerial Vehicles (UAVs) is creating new opportunities in diverse fields of technology and business. However, this increase of popularity also raises safety concerns. To tackle the primary concern of keeping the UAV inside a designated region, a novel trajectory estimation algorithm for geo-fencing applications is proposed. We derive the Beta-Trajectory that takes into account constraints in curvature as well as constraints in the change of curvature, which is bounded by the maximum roll-rate of the aircraft. We incorporate the Beta-Trajectory into a geo-fencing algorithm. By using our open-source uavAP autopilot, the applicability and necessity of accurate trajectory estimation algorithms for geo-fencing applications are shown on small fixed-wing aircraft. The model and algorithm are validated in high-fidelity simulations as well as in real flight testing.

Contributions to this paper

- Derivation of the Beta-Trajectory
- Implementation in the autopilot framework
- Autopilot configuration and monitoring during flight testing
- Shared paper writing

Copyright

© 2019 IEEE. Reprinted, with permission, from Mirco Theile, Simon Yu, Or D Dantsker, and Marco Caccamo, “Trajectory Estimation for Geo-Fencing Applications on Small-Size Fixed-Wing UAVs”, 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), November 2019.

See Appendix A.1 for the reuse statement. The following shows the accepted version.

Trajectory Estimation for Geo-Fencing Applications on Small-Size Fixed-Wing UAVs

Mirco Theile^{1*}, Simon Yu^{2*}, Or D. Dantsker¹ and Marco Caccamo¹

Abstract—The steadily increasing popularity of Unmanned Aerial Vehicles (UAVs) is creating new opportunities in diverse fields of technology and business. However, this increase of popularity also raises safety concerns. To tackle the primary concern of keeping the UAV inside a designated region, a novel trajectory estimation algorithm for geo-fencing applications is proposed. We derive the Beta-Trajectory that takes into account constraints in curvature as well as constraints in the change of curvature which is bounded by the maximum roll-rate of the aircraft. We incorporate the Beta-Trajectory into a geo-fencing algorithm. By using our open-source uavAP autopilot, the applicability and necessity of accurate trajectory estimation algorithms for geo-fencing applications are shown on small fixed-wing aircraft. The model and algorithm are validated in high-fidelity simulations as well as in real flight testing.

I. INTRODUCTION

In recent years, we have seen an uptrend in the popularity of Unmanned Aerial Vehicles (UAVs) driven by the desire to apply these aircraft to areas such as precision farming, infrastructure and environment monitoring, surveillance, surveying, and mapping, search and rescue missions, weather forecasting, and much more. All the above application scenarios require the aircraft to safely interact with the surrounding humans, environments, and other aircraft. Therefore, unmanned aircraft should be constrained to a designated area or space defined by a geo-fence.

For rotary aircraft, such as quadcopters, the task of staying inside the geo-fence is relatively simple since those type of aircraft are capable of stopping in mid-air and turning around with zero translational velocity. For fixed-wing aircraft, on the other hand, such execution of maneuvers is impossible as they need to maintain a minimum velocity in order to stay airborne. Consequently, a proper kinematic model for fixed-wing aircraft is required to determine the feasibility of a trajectory as well as the exact time for the initiation of an evasion maneuver. Most analytical kinematic models only constrain the maximum curvature of a trajectory, namely a Dubin's Curve ([1], [2]). In the context of geo-fencing, a constraint in the change of curvature has been widely ignored in the literature. The main contributions of this work are:

- 1) A precise kinematic model for a fixed-wing aircraft with constrained roll rate.

*The first two authors contributed equally to this work.

¹Mirco Theile, Or D. Dantsker, and Marco Caccamo are with the TUM Department of Mechanical Engineering, Technical University of Munich, Germany {mirco.theile, or.dantsker, mcaccamo}@tum.de

²Simon Yu is with the Dept. of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL jundayu2@illinois.edu

- 2) A geo-fencing algorithm using the model to avoid boundaries and stay in a designated area.
- 3) An implementation of the model and algorithm using our open-source uavAP autopilot.
- 4) An evaluation of the model and algorithm using high-fidelity simulators as well as real flight data.

To the best of our knowledge, this is the first geo-fencing algorithm that takes into account the constraint in the change of curvature.

A. Rationale and Related Work

The related work on geo-fencing applications mainly focuses on multicopters ([3]–[8]). As previously mentioned, the kinematic model of a fixed-wing aircraft is fundamentally different from that of a multicopter. Therefore, geo-fencing methods derived for multicopters cannot directly be adopted for fixed-wing aircraft. The authors in [7] look at both types of aircraft. They argue that the trajectories for a fixed-wing aircraft form a symmetric fan pattern around the velocity vector. This fan pattern, however, is based on the instantaneous change in roll, which we show, is not applicable for fixed-wing aircraft.

In related work done by [8], the velocity control of the aircraft is assumed to be instantaneous, meaning that its roll rate can be as high as infinity, giving an instantaneous change of roll angle. The work done by [9] demonstrates the issue of such an assumption by presenting discontinuity in curvature when flying with a circle-line-circle Dubin's path. In real-life scenarios, on the other hand, the curvature, as well as the roll rate of the aircraft, are limited. Finally, a variety of geo-fencing strategies are introduced in the related work. For instance, [8] limits the control space of the aircraft instead of completely overriding its mission when close to the geo-fence. In this work, on the other hand, the geo-fencing technique is similar to the technique mentioned in [7], which defines a soft boundary at a maximum distance that the aircraft can travel once the threat of breaking the geo-fence is detected. The authors in [10] developed an algorithm that defines different levels of safe zones to determine the necessity of an evasive maneuver. They use the Dubin's curve as their kinematic model and use slack variables to account for the deviation from it.

The authors in [11] describe a kinematic model that takes into account a constraint in the change of curvature for the purposes of optimal path planning. The approximation in this approach is only accurate for up to 30° of roll angle, which is not applicable for a geo-fencing algorithm where the

Aircraft Model	Wing-span (m)	Cruise Velocity (m/s)	Max Roll (°)	Max Roll Rate (°/s)	0° Roll Deviation (%)	Max Roll Deviation (%)
Avistar	1.6	22.0	45.0	30.0	37.3	133.3
Pulsar	4.0	12.0	45.0	33.8	60.5	215.5
Cessna 177	10.8	53.6	45.0	54.5	8.4	30.2
Boeing 747	59.6	263.0	30.0	19.1	1.8	6.8

TABLE I: The deviation is given for two types of rolling maneuvers, from 0° to the maximum roll angle and from negative maximum roll to positive maximum roll angle. Data taken from: Cessna 177 [14], Boeing 747 [15], Pulsar sailplane from our flight measurements, and Avistar which is constrained to protect on-board equipment; the notion of the deviations is described in Section II.

maximum turn capability has to be exploited. We compare our approach to this approach as well as the Dubin’s curve and show that we have higher accuracy than both of them while maintaining the computability as in [11].

Table I shows the deviation, the ratio of the distance between predicted orbit centers and the orbit radius (Figure 1), of the Dubin’s Curve from our Beta-Trajectory given four example aircraft. The deviation shown is higher when the total velocity is lower, which is crucial for small fixed-wing UAVs such as the Pulsar by F5Models sailplane [12], [13] with a deviations of 60 – 215%. In addition, the deviation of middle-size (general aviation) aircraft like the Cessna 177 of 8 – 30% is also significant and should be considered in trajectory planning. For large size aircraft like the Boeing 747, however, this effect can mostly be ignored.

This work is structured as follows: Section II contains the derivation of the Beta-Trajectory model and the percent deviation of the Dubin’s Curve showed in Table I. Section III defines the geo-fencing algorithm using the derived Beta-Trajectory model. In Section IV, the model and the algorithms are validated using the simulator and real flight data. Finally, Section V concludes this work and gives an outline of future work.

II. DERIVATION OF KINEMATIC MODEL

This section shows the derivation of the Beta-Trajectory, describing the frames of reference and presenting the deviations of the Dubin’s Curve from the Beta-Trajectory as contrast metrics. Additionally, we discuss the difference between the Beta-Trajectory and the approach in [11]. The underlying assumptions for the Beta-Trajectory are the following:

- 1) A rolling maneuver utilizes an approximately constant roll rate.
- 2) The aircraft’s altitude and velocity stay constant during the maneuver.
- 3) The effects of wind are set to zero.

The argument for assumption 2 is that a change in altitude would add additional constraints to the system due to maximum and minimum altitude bounds. Furthermore, an autopilot is capable of keeping the altitude constant during a rolling maneuver by applying a corresponding pitching actuation. Regarding assumption 3, this algorithm

is developed for small fixed-wing aircraft which often lack the necessary instrumentation to evaluate wind speed and direction. Therefore the presented approach utilizes a slack variable to account for the unknown wind effects. Modeling wind as a steady, uniform flow-field as done in [11] would only be a superposition between a translational motion and the Beta-Trajectory.

A. Beta-Trajectory

The derivation of the trajectory with constant roll rate is based on the relation between the aircraft’s roll angle ϕ and its yaw rate $\dot{\psi}$. Their relation can be expressed as follows:

$$\dot{\psi}(t) = -\frac{g}{V} \tan \phi(t) \quad (1)$$

where g is the gravitational constant and V is the total velocity of the aircraft. The relation demonstrates the contribution of the rotated lift-force to the centripetal force acting on the aircraft. The constant roll angle rate $\dot{\phi}_c$ influences the roll angle ϕ by

$$\phi(t) = \dot{\phi}_c t + \phi_0 \quad (2)$$

where ϕ_0 is the initial roll angle of the aircraft. Since we assume constant altitude during flights, the velocity and position of the aircraft can be expressed in two dimensions. To simplify the derivations of the trajectory, we define the velocity, as well as the position, in the complex plane. Hence

$$\begin{aligned} \dot{c}(t) &= v(t) = v_x(t) + iv_y(t) \\ c(t) &= x(t) + iy(t) \end{aligned}$$

Using the complex form, we can then express the velocity using Euler’s Equation as

$$v(t) = V e^{i\psi(t)} \quad (3)$$

where $\psi(t)$ can be found by integrating (1). The full derivation of the kinematic model can be found in our technical report [16]. The following results are obtained by integrating the yaw rate and velocity. The position of the aircraft flying with constant roll rate can be expressed using the complex Incomplete Beta Function defined as the following:

$$B(x; a, b) = \int_0^x y^{a-1} (1-y)^{b-1} dy \quad (4)$$

where $B : \mathbb{R} \times \mathbb{C} \times \mathbb{C} \mapsto \mathbb{C}$. By using the Incomplete Beta Function, we are able to describe every point on the trajectory as a function of the roll angle of the aircraft:

$$\beta(\phi) = \text{sign}(\phi) \frac{V}{2\dot{\phi}_c} [B(1; a, b) - B(\cos^2 \phi; a, b)] \quad (5)$$

where $\text{sign}()$ is the sign function that returns the sign of its argument. The parameters a and b for the Incomplete Beta Function are defined as follows:

$$a = \frac{1}{2} + i \frac{g}{2V\dot{\phi}_c}, \quad b = \frac{1}{2}$$

Additionally, the yaw angle of the aircraft at each roll angle can be calculated using

$$\psi(t) = \frac{g}{V\dot{\phi}_c} \ln \cos \phi(t) \quad (6)$$

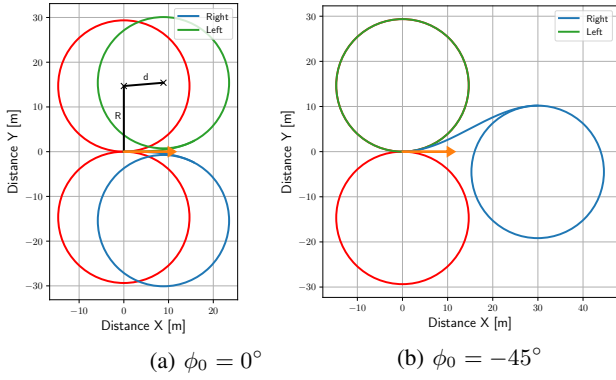


Fig. 1: Contrast between Dubin's Path (red) and Beta-Trajectory (green and blue, left and right turn, respectively); Orange arrow indicates initial aircraft position and yaw angle. Data is taken from the Pulsar sailplane in Table I.

The equations for the position and yaw angle of the aircraft are relative to a trajectory frame. In this frame, the initial position and yaw angle of the aircraft are the following:

$$c_0 = \text{sign}(\phi_0) \frac{V}{2\dot{\phi}_c} [B(1; a, b) - B(\cos^2 \phi_0; a, b)] \quad (7)$$

$$\psi_0 = \frac{g}{V\dot{\phi}_c} \ln \cos \phi_0 \quad (8)$$

The full trajectory of an aircraft with a constrained roll rate as well as a constrained maximum roll angle ϕ_{max} can be described as a disjunction of the Beta-Curve B and an orbit \mathcal{O} . We define

$$B(\phi_0, \phi_{max}) = \{\beta(\phi) \mid \forall \phi \in [\phi_0, \phi_{max}]\} \quad (9)$$

depending on the initial and maximum roll angle. The orbit

$$\mathcal{O}(c_{center}, R) = \{c \in \mathbb{C} \mid \|c - c_{center}\| = R\} \quad (10)$$

is defined by its radius R and center point c_{center} . The radius is calculated from the curvature in (1) as

$$R = \left\| \frac{V^2}{g \tan \phi} \right\| \quad (11)$$

The center point is geometrically calculated by adding R perpendicularly to the last point of the Beta-Curve in the roll direction, thus

$$c_{center}(\phi) = \beta(\phi) - i \text{sign}(\phi) R e^{i\psi(\phi)} \quad (12)$$

Consequently, the full Beta-Trajectory is expressed through the disjunction of the Beta-Curve and orbit as

$$T(\phi_0, \phi_{max}) = \mathcal{O}(c_{center}(\phi_{max}), R) \cup B(\phi_0, \phi_{max}) \quad (13)$$

Two example trajectories can be seen in Figure 1 using Pulsar sailplane data from Table I.

B. Frames of Reference

The definition of the Beta-Trajectory in (13) is based on the usage of different frames of reference. To apply the description of the trajectory to a geo-fencing context, a transformation needs to be established. In this work, we

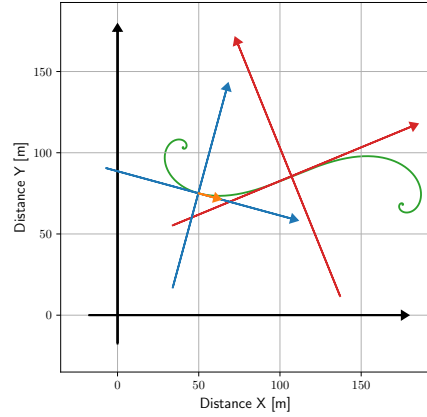


Fig. 2: Frames of reference: Red is the Trajectory-Frame, centered at $\beta(\phi = 0)$; Blue is the Aircraft-Frame, centered at the aircraft position c_0^E ; Black is the Earth-Frame, which defines global positions (GPS); Green is an example Beta-Curve $B(-\frac{\pi}{2}, \frac{\pi}{2})$ with positive roll rate.

describe three frames of reference, illustrated in Figure 2. First is the Trajectory-Frame, described as c^T for position and ψ^T for yaw angle where each point on the trajectory can be described using (5). This frame is centered around the point at which the roll angle of the aircraft is zero, hence, $\beta^T(\phi = 0) = 0 + i0$, as well as $\psi^T(\phi = 0) = 0$.

The second frame is the Earth-Frame, described by c^E and ψ^E and based on the GPS coordinates. In this work, the Earth-Frame is the UTM-Frame in East-North-Up (ENU) notation where the zero yaw angle is in the east direction. The third frame is the Aircraft-Frame, described by c^A and ψ^A and centered around the current aircraft position where the current aircraft yaw angle is defined as zero.

To transform from the Trajectory-Frame to the Earth-Frame, the Aircraft-Frame serves as an intermediary for the conversion. With respect to the Earth-Frame, the Aircraft-Frame is offset by the current aircraft position c_0^E and rotated by the current aircraft yaw angle ψ_0^E . Additionally, the Aircraft-Frame is defined in the Trajectory-Frame with the offset c_0^T from (7) and the rotation ψ_0^T from (8). The transformation between the frames can be described in the complex plane. The transformation from the Trajectory-Frame to Aircraft-Frame can be expressed as

$$c^A = e^{-i\psi_0^T} (c^T - c_0^T) \quad (14)$$

$$\psi^A = \psi^T - \psi_0^T \quad (15)$$

Similarly the transformation from Earth-Frame to Aircraft-Frame is

$$c^A = e^{-i\psi_0^E} (c^E - c_0^E) \quad (16)$$

$$\psi^A = \psi^E - \psi_0^E \quad (17)$$

Combining (14) and (16) as well as (15) and (17) yields

$$c^E = e^{i\psi_0^E} e^{-i\psi_0^T} (c^T - c_0^T) + c_0^E \quad (18)$$

$$= e^{i(\psi_0^E - \psi_0^T)} (c^T - c_0^T) + c_0^E \quad (19)$$

$$\psi^E = \psi^T - \psi_0^T + \psi_0^E \quad (20)$$

as the transformation from the Trajectory-Frame to the Earth-Frame.

For simplification of the following equations, the frame notation is mostly avoided. An equation without frame indication assumes all the arguments are in the same frame. Since the frames represent linear and orthonormal transformations, an equation holding in one frame will hold in other frames.

C. Comparison to Other Approaches

To illustrate the necessity of the kinematic model, we compare it to the Dubin's-Curve model in [2], which is primarily used in the literature. For Dubin's-Curves, only the curvature is constrained, but not the change in curvature determined by the maximum roll rate. As seen in Figure 1, the trajectory deviates from the orbit predicted by the Dubin's-Curve. We define the percent deviation as the ratio between the distance of the center points d and the radius R . The radius R is calculated using (11) and the distance d is calculated using the two center points:

$$d = \|c_{center}(\phi_{max}) - c_{dubin}\| \quad (21)$$

Calculation for c_{dubin} is similar to (12) except that it is set next to the initial position. Hence

$$c_{dubin} = \beta(\phi_0) - i \operatorname{sign}(\dot{\phi}_c) \operatorname{Re}^{i\psi(\phi_0)} \quad (22)$$

Using the above equations, the deviation values in Table I are calculated.

The authors in [11] linearize the $\tan()$ in (1) to yield

$$\dot{\psi}(t) \approx -\frac{g}{V} \phi(t) \quad (23)$$

which they call Continuous-Curvature Convected Dubins-Curve (CCC-Dubin). This results in a representation of the position on the trajectory using Fresnel integrals, which can only be solved numerically, similar to the Beta Incomplete Function. The linearization introduces deviations at roll angles greater than 30° , which we show in Section IV.

III. GEO-FENCING

We define the geo-fence F as a convex¹ polygon described by N fence segments where each segment F_k is defined as

$$F_k = \{c \in \mathbb{C} \mid \operatorname{Re}\{\bar{n}_k c\} = b_k\} \quad (24)$$

where n_k is the unit normal vector of the line pointing into the polygon and b_k is the offset. \bar{n}_k is the complex conjugate of n_k and $\bar{n}_k c$ represents the inner product of n_k and c in the complex plane. Using the definitions, a *safe* area with respect to F_k can be expressed as

$$S_k = \{c \in \mathbb{C} \mid \operatorname{Re}\{\bar{n}_k c\} > b_k\} \quad (25)$$

which describes the *safe* half-space in \mathbb{C} . We add a slack s as a safety margin to account for factors such as the wind, resulting in

$$S_k(s) = \{c \in \mathbb{C} \mid \operatorname{Re}\{\bar{n}_k c\} > b_k + s\} \quad (26)$$

¹For concave polygons this approach can be used as well but requires additional steps.

The safe area of the full geo-fence F is then described by

$$S(s) = \bigcap_{\forall k} S_k(s) \quad (27)$$

A. Safety of Beta-Trajectory

We define the Beta-Trajectory T as *safe* if it satisfies

$$T(\phi_0, \phi_{max}) \subset S(s) \quad (28)$$

For the trajectory to fully lie in the safe area, both the orbit and the Beta-Curve need to lie in the safe area. For the orbit, the condition can be written as

$$\mathcal{O}(c_{center}(\phi_{max}), R) \subset S(s) \quad (29)$$

and trivially simplified to

$$c_{center}(\phi_{max}) \in S(s + R) \quad (30)$$

which can be immediately verified.

To evaluate

$$B(\phi_0, \phi_{max}) \subset S(s) \quad (31)$$

it is necessary to check each fence segment F_k individually. The Beta-Curve lies fully in $S_k(s)$ if its outer most point, a critical point, in the direction of a fence segment F_k lies in $S_k(s)$. This critical point is found through the following minimizer:

$$\phi_{crit,k} = \arg \min_{\phi \in [\phi_0, \phi_{max}]} \operatorname{Re}\{\bar{n}_k \beta(\phi)\} \quad (32)$$

This equation could be solved analytically. However, there is an intuitive solution simplifying the problem. Since $B(\phi_0, \phi_{max})$ is a smooth function describing the aircraft trajectory heading towards and away from the fence, potential critical points on the trajectory are the points where the aircraft is flying parallel to the fence. The start and end point of $B(\phi_0, \phi_{max})$, i.e. the current position and the start of the orbit can be evaluated individually.

The direction of the fence segment F_k is defined by its normal vector n_k as

$$\psi_k(l) = \arg\{in_k\} + l\pi, \quad l \in \mathbb{Z} \quad (33)$$

meaning that the direction is the 90-degree rotated normal vector and all of its rotationally coinciding vectors. Those coinciding vectors are significant since the Beta-Trajectory has the form of a spiral and, thus, can have multiple parallels. In the Trajectory-Frame, we can solve for the corresponding roll angle by inverting (6). The inversion yields

$$\phi = \pm \cos^{-1}\left(e^{\frac{\phi_c V}{g} \psi^T}\right) := \lambda(\psi^T) \quad (34)$$

Consequently, the critical roll angles on the $B(\phi_0, \phi_{max})$ with respect to fence segment F_k are

$$\Phi_k = \{\phi = \lambda(\psi_k^T(l)) \mid \forall l \in \mathbb{Z}, \phi \in [\phi_0, \phi_{max}]\} \quad (35)$$

where the global minimum of (32) satisfies $\phi_{crit,k} \in \Phi_k$. Note that if $\lambda(\psi_k^T(l)) > \phi_{max}$, so is $\lambda(\psi_k^T(l+1))$, simplifying the search for critical roll angles. We can write that

$$B(\phi_0, \phi_{max}) \subset S_k(s) \quad (36)$$

$$\text{iff } \beta(\phi) \in S_k(s), \quad \forall \phi \in \Phi_k \quad (37)$$

To conclude, the trajectory $T(\phi_0, \phi_{max})$ is *safe* if

$$\beta(\phi) \in S_k(s), \forall \phi \in \Phi_k, \forall k \in \{1, \dots, N\} \\ \wedge \\ c_{center}(\phi_{max}) \in S(s + R)$$

This relation shows that for the geo-fencing algorithm only a few distinct points on the predicted trajectory have to be evaluated.

B. Implementation

Algorithm 1: Evaluate Safety

Input: T
Output: Safety of T
Data: aircraftState, F
orbitCenter = getOrbitCenter(T);
forall $F_k \in F$ **do**
 if orbitCenter $\notin S_k(s + R)$ **then**
 return *unsafe*;
 $\Phi =$ getCriticalRolls(T, F_k);
 for $\phi \in \Phi$ **do**
 if $\beta(\phi) \notin S_k(s)$ **then**
 return *unsafe*;
return *safe*;

The following geo-fencing algorithm is used to initiate an evasive maneuver if the aircraft’s current position, attitude, and velocity indicate that a violation of the geo-fence is imminent. To determine if a violation is imminent, we consider two evasive maneuvers: turning fully to the left or right with the constant roll rate up to the maximum roll angle. We say that the aircraft will inevitably break the fence if the resulting trajectories of the evasive maneuvers violate the geo-fence.

At every time step, both trajectories are generated and evaluated. Algorithm 1 implements the derivation in the beginning of this section. It returns *unsafe* if a given trajectory violates any of the fence segments and *safe* otherwise. If both trajectories are classified as *unsafe*, an evasive maneuver is executed in the direction that was last classified as *safe*. The evasive maneuver ends when the aircraft is flying away from the violated fence segment.

The implementation of the algorithms in uavAP autopilot makes use of an overriding framework that enables modules to directly override the targets of the controller. When initiating an evasive maneuver, the geo-fencing module overrides the controller roll target to the maximum roll angle in the corresponding direction. To generate the trajectories and evaluate the β function in (5), the Arb library [17] is used. The Arb library uses ball arithmetic to solve real and complex functions such as the incomplete beta function.

IV. EVALUATION

In order to evaluate the proposed kinematic model and the geo-fencing algorithm, we conducted evaluations in

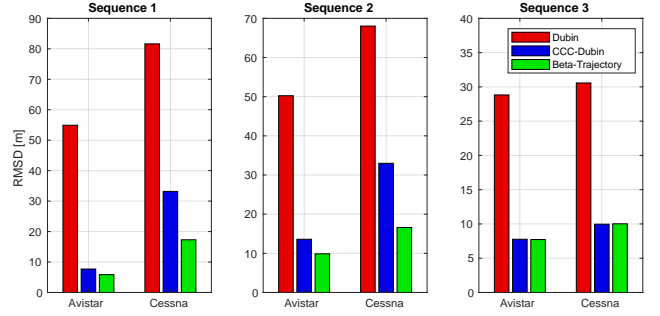


Fig. 3: Root-mean-square deviation of the trajectory predictions of the three approaches for three 30 second maneuvering sequences, comparing in two flight simulators, Trainer in FS One® and Cessna in X-Plane® 11; note the different scale of the y-axis.

simulations as well as in real flights. In this section, we describe the experimental setup, followed by an evaluation and comparison of the Beta-Trajectory, and concluding with an experimental geo-fenced flight.

A. Experimental Setup

To evaluate the Beta-Trajectory, we compare it to the Dubin’s-Curve in [9] as well as the Continuous-Curvature Connected Dubin’s-Curve (CCC-Dubin) in [11]. The uavAP autopilot is instructed to fly sequences of roll maneuvers in two different flight simulators, namely the high-fidelity FS One® Flight Simulator ([18], [19]) as well as the X-Plane® 11 ([20]). The aircraft used are a Trainer in FS One® similar to the Avistar Elite and the Cessna 172 in X-Plane®11 similar to the Cessna 177. The simulators are connected through the uavEE emulation environment described in [21].

The geo-fencing algorithm is evaluated by flying an aircraft inside a defined geo-fence. The hardware used for the actual flight is composed of an aircraft and computational hardware. A fixed-wing trainer-type radio control aircraft, the Great Planes Avistar Elite built for previous avionics development [21]–[24], is used for the evaluation. The Avistar has a 1.59 m wingspan and a mass of 3.92 kg; it has the following control surfaces: two ailerons (roll), two flaps, one elevator (pitch), and one rudder (yaw). The specifications of the aircraft can be found in [23]. The aircraft was instrumented with an Al Volo FC+DAQ 400 Hz flight computer and data acquisition system [25], which integrates the open-source uavAP autopilot [13]. The uavAP autopilot is based on a modular and configurable framework that allows for easy integration of different planning and control algorithms. For detailed information about uavAP, the interested reader is directed to the GitHub page².

B. Trajectory Prediction

To evaluate the prediction accuracy of the Beta-Trajectory, the autopilot is instructed to fly a sequence of roll maneuvers, alternating from right to left, for 30 seconds in total. The

²<https://github.com/theilem/uavAP.git>

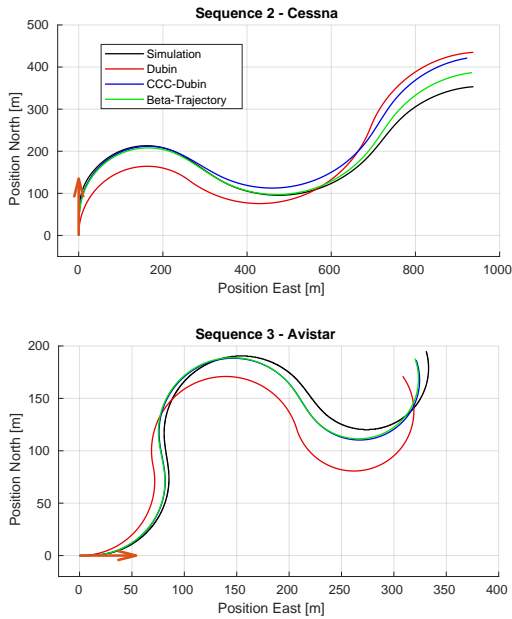


Fig. 4: Two simulated roll maneuver sequences recorded and compared to the three prediction approaches; the orange arrow shows the initial position and heading.

trajectory is recorded and compared to a prediction using the three approaches. The predictions are based on the roll sequences, the aircraft velocity, and the estimated roll rate. The performance of the respective approaches is evaluated based on the deviation to the simulated trajectory. The deviation is calculated as the root-mean-squared deviation/distance (RMSD) of the trajectories. Figure 3 shows the deviation for three different sequences executed with the Avistar and the Cessna. While sequence 1 and sequence 2 used roll targets from the whole spectrum of roll angles (-45° to 45°), sequence 3 is constrained to angles from only -30° to 30° , the linearization limits of the CCC-Dubin’s curve. This is to show that the deviation in CCC-Dubin’s curves arise from the linearization of the $\tan(\cdot)$. It can be seen that both the CCC-Dubin approach as well as the Beta-Trajectory outperform the Dubin’s approach in the sequences. In sequences 1 and 2, the Beta-Trajectory performs better than the CCC-Dubin approach while they show equal performance in sequence 3.

Figure 4 shows two examples from the sequences, first the sequence 2 using the Cessna and second the sequence 3 using the avistar. Due to the low roll angles in sequence 3, the CCC-Dubin and Beta-Trajectory give equal predictions. In sequence 2, however, a significant drift towards the end of the trajectory can be observed in the CCC-Dubin prediction. The Dubin approach deviates immediately and demonstrates that it is not suitable for short horizon predictions.

C. Geo-Fencing

Since it predicts the simulator path with high accuracy, the Beta-Trajectory is incorporated into the geo-fencing algorithm and deployed it onto a real aircraft. The autopilot on the aircraft was instructed to fly out of the designated

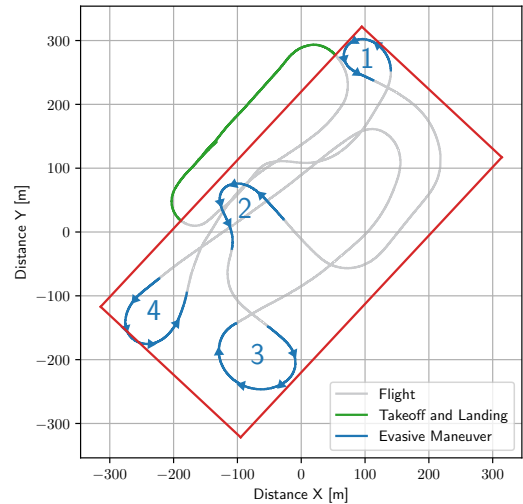


Fig. 5: Real flight path of the Avistar deployed with the uavAP autopilot and the geo-fencing algorithm; Red shows the geo-fence. Blue shows the evasive maneuvers labeled from 1-4. Slack value is 5 meters and velocity is 20 m/s.

area defined by the geo-fence. The higher level geo-fencing algorithm observes the flight and intervenes the current flight path when necessary. The result can be seen in Figure 5. The slack in Equation (26) was set low to emphasize the effect. The aircraft is kept inside the geo-fence with two prominent deviations. The first deviation is reflected by evasive maneuver 4 where the aircraft slightly overshoots the fence. The second deviation occurs in evasive maneuver 2, which has a greater safety margin than the others. These two deviations are attributed to the effects of wind. Since the initial assumption presumes no wind, the deviations due to winds and gusts are expected. However, these slight deviations show the algorithm’s ability to decently perform in windy situations despite a low value for the slack. Future work regarding wind integration will improve the results even further.

V. CONCLUSION AND FUTURE WORK

In this work, we show the applicability and the impact of the Beta-Trajectory as a new kinematic model for fixed-wing aircraft. The trajectory is derived from the maximum roll angle as well as the maximum roll rate constraints. Using the newly derived kinematic model, we develop a geo-fencing algorithm that aims to keep the aircraft in a designated area. The theoretical derivation of the kinematic model and the algorithms are validated in high-fidelity simulations using the uavEE emulation environment as well as in real flights. The source code related to this work is available in the open-source autopilot uavAP.

For future work, the Beta-Trajectory can be used for path and trajectory planning algorithms. The model can be updated to incorporate a change of altitude which would affect the relationship between the roll angle and yaw rate. In order to make use of instrumentation that estimates wind speed and direction, wind effects need to be added to the geo-

fencing algorithm. As mentioned previously, wind effects can be incorporated as a superposition to the Beta-Trajectory. The geo-fencing algorithm also needs to be adapted to calculate critical roll angles based on the course angle of the aircraft, which represents the direction of the total aircraft velocity vector including the wind effects.

ACKNOWLEDGMENTS

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant number CNS-1646383. Marco Caccamo was also supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

- [1] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957.
- [2] I. Lugo-Crdenas, G. Flores, S. Salazar, and R. Lozano, "Dubins path generation for a fixed wing uav," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 339–346, May 2014.
- [3] M. N. Stevens and E. M. Atkins, "Multi-mode guidance for an independent multicopter geofencing system," in *16th AIAA Aviation Technology, Integration, and Operations Conference*, AIAA Paper No. 2016-3150, AIAA AVIATION Forum, Jun. 2016.
- [4] S. Zhang, D. Wei, M. Q. Huynh, J. X. Quek, X. Ma, and L. Xie, "Model predictive control based dynamic geofence system for unmanned aerial vehicles," in *AIAA Paper No. 2017-0675, AIAA Infotech @ Aerospace*, Jan. 2017.
- [5] M. N. Stevens, B. Coloe, and E. M. Atkins, "Platform-independent geofencing for low altitude uas operations," in *AIAA Paper No. 2015-3329, 15th AIAA Aviation Technology, Integration, and Operations Conference, AIAA AVIATION Forum*, Jun. 2015.
- [6] H. T. Dinh, M. H. C. Torres, and T. Holvoet, "Dancing uavs: Using linear programming to model movement behavior with safety requirements," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 326–335, June 2017.
- [7] E. T. Dill, S. D. Young, and K. J. Hayhurst, "Safeguard: An assured safety net technology for uas," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pp. 1–10, Sept 2016.
- [8] T. Gurriet and L. Ciarletta, "Towards a generic and modular geofencing strategy for civilian uavs," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 540–549, June 2016.
- [9] M. Shanmugavel, A. Tsourdos, B. White, and R. bikowski, "Co-operative path planning of multiple uavs using dubins paths with clothoid arcs," *Control Engineering Practice*, vol. 18, no. 9, pp. 1084 – 1092, 2010.
- [10] A. J. Bateman, W. Gressick, and N. Gandhi, "Application of run-time assurance architecture to robust geofencing of suas," in *AIAA Paper No. 2018-1985, AIAA Infotech@ Aerospace*, Jan. 2018.
- [11] L. Techy, C. A. Woolsey, and K. A. Morgansen, "Planar path planning for flight vehicles in wind with turn rate and acceleration bounds," in *2010 IEEE International Conference on Robotics and Automation*, pp. 3240–3245, IEEE, 2010.
- [12] O. D. Dantsker, M. Theile, M. Caccamo, and R. Mancuso, "Design, development, and initial testing of a computationally-intensive, long-endurance solar-powered," in *AIAA Paper No. 2018-4217, AIAA Applied Aerodynamics Conference, Atlanta, Georgia*, Jun. 2018.
- [13] Real Time and Embedded System Laboratory, University of Illinois at Urbana-Champaign, "Solar-Powered Long-Endurance UAV for Real-Time Onboard Data Processing." <http://rtsl-edge.cs.illinois.edu/UAV/>, Accessed Sep. 2018.
- [14] D. L. Kohlman, "Flight test data for a cessna cardinal. [steady state performance and fixed stick dynamic stability characteristics]," *NASA Contractor Report CR-2337*, Jan 1 1974.
- [15] D. Geleyns, "Alternative flight control in civil aviation," Master's thesis, Delft University of Technology, the Netherlands, 2016.
- [16] M. Theile and S. Yu, "Kinematic Model for Fixed-Wing Aircraft with Constrained Roll-Rate," tech. rep., University of Illinois at Urbana-Champaign, Department of Computer Science, Sep. 2018.
- [17] Fredrik Johansson, "Arb - a C library for arbitrary-precision ball arithmetic." <http://arblib.org/>, Accessed Sep. 2018.
- [18] M. S. Selig, "Real-time flight simulation of highly maneuverable unmanned aerial vehicles," *Journal of Aircraft*, vol. 51, pp. 1705–1725, Nov.-Dec. 2014.
- [19] M. Selig, "Modeling propeller aerodynamics and slipstream effects on small uavs in realtime," in *AIAA Paper No. 2010-7638, AIAA Atmospheric Flight Mechanics Conference, Toronto, Ontario, Canada*, Aug. 2010.
- [20] Laminar Research, "X-Plane 11." <http://www.x-plane.com/>, Accessed Mar. 2019.
- [21] M. Theile, O. D. Dantsker, R. Nai, and M. Caccamo, "uavee: A modular, power-aware emulation environment for rapid prototyping and testing of uavs," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Hakodate, Japan*, Aug. 2018.
- [22] R. Mancuso, O. D. Dantsker, M. Caccamo, and M. S. Selig, "A low-power architecture for high frequency sensor acquisition in many-DOF UAVs," in *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPs)*, Berlin, Germany, Apr. 2014, pp. 103–114, 2014.
- [23] O. D. Dantsker, R. Mancuso, M. S. Selig, and M. Caccamo, "High-frequency sensor data acquisition system (sdac) for flight control and aerodynamic data collection research on small to mid-sized uavs," in *AIAA Paper No. 2014-2565, AIAA Applied Aerodynamics Conference, Atlanta, Georgia*, Jun. 2014.
- [24] O. D. Dantsker, M. Theile, and M. Caccamo, "A high-fidelity, low-order propulsion power model for fixed-wing electric unmanned aircraft," in *AIAA Paper No. 2018-5009, AIAA/IEEE Electric Aircraft Technologies Symposium, Cincinnati, Ohio*, Jul. 2018.
- [25] Al Volo LLC, "Al Volo: Flight Systems." <http://www.alvolo.us>, Accessed Sep. 2018.

3.2 uavAP: A Modular Autopilot Framework for UAVs

Reference

M. Theile, O. Dantsker, R. Nai, M. Caccamo, and S. Yu, “uavAP: A modular autopilot framework for UAVs,” in *AIAA AVIATION 2020 FORUM*, p. 3268, 2020
DOI: <https://doi.org/10.2514/6.2020-3268>

Abstract

Being applied to many fields of research and industry, UAVs require reliable but modular autopilot software. An autopilot task can range from simple waypoint following to complex maneuvering or adaptive mission tracking. The developed and presented autopilot, uavAP, aims to be fully modular in a decentralized manner, embracing an object-oriented design in C++. It implements a typical control stack comprising of a mission planner, global planner, local planner, and controller. To facilitate its modularity, uavAP makes use of its core, cpsCore, for module management as well as core utilities. cpsCore administers the configuration, aggregation, and synchronization of all the modules in uavAP. With the emulation environment uavEE, uavAP forms an ecosystem for rapid prototyping and testing of modules for various research directions, ranging from scheduling and memory management, through planning and control system design, to flight profile and configuration optimization. The uavAP-uavEE ecosystem has facilitated the design of an accurate UAV power model based on the aircraft’s physical model, flight maneuver automation for aircraft system identification and dynamics parametrization, and an algorithm for geo-fencing of fixed-wing UAVs. This paper describes the control stack of uavAP, its core, cpsCore, as well as application examples highlighting the framework’s modularity and flexibility.

Contributions to this paper

- Conceptualization of the software frameworks
- Main contributor to the cpsCore and uavAP source code
- Autopilot configuration and monitoring during flight testing
- Majority of paper writing

Copyright

© 2020 by Mirco Theile, Or D. Dantsker, Richard Nai, Marco Caccamo, and Simon Yu. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission.

See Appendix A.2 for the reuse statement. The following shows the accepted version.

uavAP: A Modular Autopilot Framework for UAVs

Mirco Theile*, Or D. Dantsker[†], Richard Nai[‡] and Marco Caccamo[§]

Technical University of Munich, Garching, Germany

Simon Yu[¶]

University of Illinois at Urbana–Champaign, Urbana, IL, USA

Being applied to many fields of research and industry, UAVs require reliable but modular autopilot software. An autopilot task can range from simple waypoint following to complex maneuvering or adaptive mission tracking. The developed and presented autopilot, uavAP, aims to be fully modular in a decentralized manner, embracing an object-oriented design in C++. It implements a typical control stack comprising of a mission planner, global planner, local planner, and controller. To facilitate its modularity, uavAP makes use of its core, cpsCore, for module management as well as core utilities. cpsCore administers the configuration, aggregation, and synchronization of all the modules in uavAP. With the emulation environment uavEE, uavAP forms an ecosystem for rapid prototyping and testing of modules for various research directions, ranging from scheduling and memory management, through planning and control system design, to flight profile and configuration optimization. The uavAP-uavEE ecosystem has facilitated the design of an accurate UAV power model based on the aircraft's physical model, flight maneuver automation for aircraft system identification and dynamics parametrization, and an algorithm for geo-fencing of fixed-wing UAVs. This paper describes the control stack of uavAP, its core, cpsCore, as well as application examples highlighting the framework's modularity and flexibility.

I. Introduction

The popularity of UAVs in many fields of research and industry creates the need for reliable but modular autopilot software. An autopilot task can range from simple waypoint following to complex maneuvering or adaptive mission tracking. While there are existing autopilot systems with excellent community support, they are not sufficiently modular to enable rapid adaptability to varying research directions. The developed and presented autopilot, uavAP, aims to be fully modular in a decentralized manner, embracing object-oriented design in C++. Every functionality in uavAP is provided by a module, which can be adapted or replaced.

uavAP is an open-source autopilot framework.¹ The autopilot structure is designed for distributed functional executions that separate control, planning, and communication to increase safety and security at software level. It implements a typical control stack comprising of a mission planner, global planner, local planner, and controller. The high-level and abstract design of uavAP allows for seamless switching and transition between various planning and control algorithms. The customizability of the autopilot structure provides the flexibility that allows for rapid interfacing with various hardware systems such as the AI Volo FC+DAQ system,² which is used for high precision data collection necessary for applications such as power modeling.

To facilitate the modularity, uavAP makes use of cpsCore for module management as well as core utilities. cpsCore administers the configuration, aggregation, and synchronization of all the modules in uavAP. Through these steps,

*Ph.D. Student, Department of Mechanical Engineering. mirco.theile@tum.de

[†]Researcher, Department of Mechanical Engineering, or.dantsker@tum.de

[‡]M.S. Student, Department of Informatics, richard.nai@tum.de

[§]Professor, Department of Mechanical Engineering, mcaccamo@tum.de

[¶]Ph.D. Student, Department of Electrical and Computer Engineering. jundayu2@illinois.edu

each module can easily specify a set of parameters which are loaded and applied on program startup, communicate and interact with other modules, and start their task synchronously throughout threads and individual processes. Additionally, cpsCore offers core utilities for essential tasks such as scheduling, inter-process communication, and more. While initially only the core of uavAP, cpsCore has become an individual project because of its valuable support for modularization of any C++ software framework for cyber-physical systems (CPS).

The uavAP autopilot framework forms an ecosystem with uavEE, an open-source emulation environment for UAVs.^{3,4} uavAP interfaces with uavEE for the communications with flight simulations while uavEE enables rapid testing and debugging of the uavAP autopilot framework and planning and control designs and implementations. More importantly, the combination of the uavAP and uavEE framework has enabled projects on variable applications in a wide range of areas. The uavAP-uavEE ecosystem has facilitated the design of an accurate UAV power model based on the physical model of the aircraft. Additionally, a flight maneuver automation framework⁵ has been developed in uavAP and tested in uavEE. The framework automates flight testing maneuvers for aircraft system identification and dynamics parametrization,⁶ yielding more consistent and repeatable results than human operators. Finally, an accurate kinematic model and algorithm for fixed-wing aircraft geo-fencing have been developed using uavAP and uavEE.⁷

The paper is structured as follows: In Section II, the autopilot framework uavAP is introduced with a summary of its planning and control stack and distributed architecture. This is followed by a description of its underlying core, cpsCore, which manages the modules and provides core utilities. In Section IV, a flight maneuver automation integration into uavAP is shown as an example of uavAP's modularity and flexibility. Some applications of uavAP are shown in Section V, followed by a comparison with other open-source autopilots in Section VI. Section VII concludes the paper and presents an outlook into future work.

II. Modular Autopilot Framework – uavAP

This section describes the autopilot framework by introducing the implemented control stack and its individual modules. Furthermore, its distributed architecture is depicted and described.

A. Planning and Control Stack

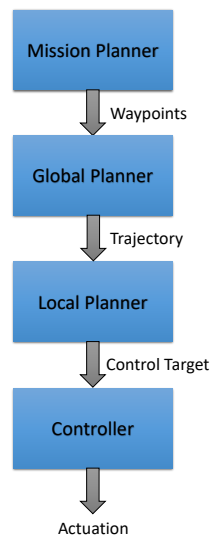


Figure 1. Planning and control stack implemented in uavAP.

The hierarchy of the control process can be represented by a stack, depicted in Figure 1. A mission planner generates waypoints according to the overall mission. The waypoints are passed to the global planner, which calculates the mission trajectory. Based on the trajectory and the current position of the aircraft, the local planner calculates the necessary angular rates and velocities to reach the trajectory, formulating a controller target. This controller target is passed to the controller, which calculates the actual actuator commands. The individual modules, as implemented in uavAP, are described in the following.

1. Mission Planner

The mission planner uses predefined missions, which can be selected by the user at run-time. The predefined mission consists of waypoints which should be passed in specified order with a specified velocity. Alternatively, the mission planner could generate waypoints to adapt its mission. However, this is out of scope for this work.

2. Global Planner

The global planner's task is to calculate a trajectory based on the set of waypoints received from the mission planner. The calculation of the trajectory can differ based on the overall goal of the mission. The simplest global planner is to connect the waypoints with lines, leading to a polygonal path, which is not the best solution since it leads to abrupt turns and consequently, high deviations from the planned trajectory. Alternatively, the waypoints can be connected with three-dimensional cubic splines, which are implemented in the *SplineGlobalPlanner* in uavAP.

A cubic spline is defined through a third degree polynomial

$$x(u) = f_x(u) = c_{0,x} + c_{1,x}u + c_{2,x}u^2 + c_{3,x}u^3. \quad (1)$$

The parametrization $u \in [0, 1]$ is defined such that for $u = 0$ the spline is at the start point and at $u = 1$ at the end point. Extending this expression to three dimensions yields

$$\vec{\mathbf{x}}(u) = \mathbf{f}(u) = \mathbf{c}_0 + \mathbf{c}_1u + \mathbf{c}_2u^2 + \mathbf{c}_3u^3, \quad \vec{\mathbf{x}} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad \mathbf{c}_k = \begin{pmatrix} c_{k,x} \\ c_{k,y} \\ c_{k,z} \end{pmatrix} \quad (2)$$

for one specific spline. The spline between the waypoints \mathbf{p}_i and \mathbf{p}_{i+1} is defined by

$$\vec{\mathbf{x}}(i, u) = \mathbf{f}_i(u) = \mathbf{c}_{0,i} + \mathbf{c}_{1,i}u + \mathbf{c}_{2,i}u^2 + \mathbf{c}_{3,i}u^3. \quad (3)$$

The *SplineGlobalPlanner* uses the Catmull-Rom formulation to calculate the parameters of the splines. The complete mathematical derivation can be found in.⁸ Catmull-Rom splines enforce a specified tangent at each waypoint. The tangent is based on the previous and next waypoint, making each spline dependent on only four waypoints. Defining $\mathbf{C}_i = [\mathbf{c}_{1,i}, \mathbf{c}_{2,i}, \mathbf{c}_{3,i}]^T$, the spline parameters can be calculated as follows:

$$\mathbf{C}_i = \begin{bmatrix} -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 3 & 3 - 2\tau & -\tau \\ -\tau & 2 - \tau & \tau - 2 & \tau \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_{i+2} \end{bmatrix} \quad (4)$$

and $\mathbf{c}_{0,i}$ is \mathbf{p}_i . The parameter τ indirectly defines how high the curvature is at the waypoints. A higher τ reduces the curvature at the waypoint but increases the curvature between the waypoints. Since each spline is dependent on a constant number of four waypoints the complexity of the Catmull-Rom spline generation is $\mathcal{O}(n)$, where n is the total number of waypoints in the mission. The *SplineGlobalPlanner* implements the calculation of Catmull-Rom splines

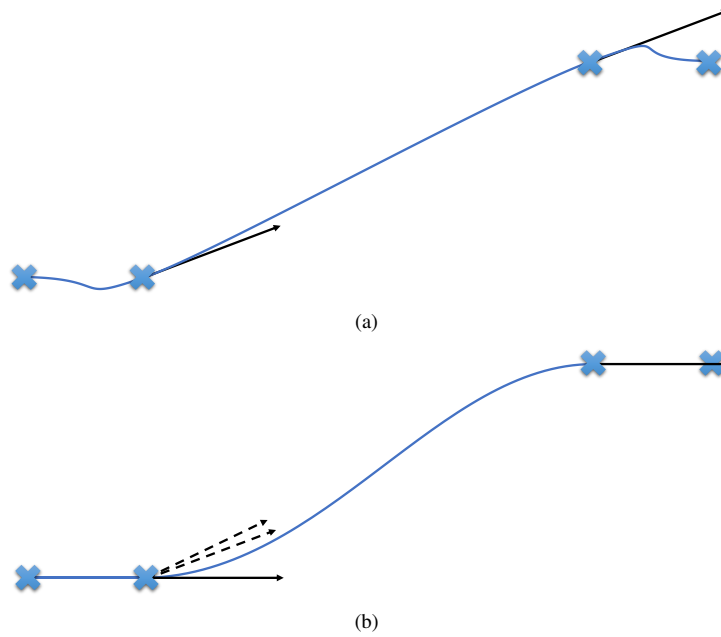


Figure 2. Catmull-Rom spline problem (a) and solution (b) for the z -coordinate spline.

because of their simplicity. The generated trajectory at its z -coordinate projection, however, shows problems when using the Catmull-Rom formulation. Figure 2(a) shows a side view of the trajectory. The black arrows show the tangents at the waypoints, which lead to unwanted altitude changes. The solution to this problem is to decouple the z -calculation from the calculation in (4). The tangent at the waypoints is set to the minimum absolute slope of three different slopes shown in Figure 2(b) on the left side. The three different slopes result from the vectors connecting the previous and the current waypoint, the current and the next waypoint, and the previous to the next waypoint. This modification is possible since the Catmull-Rom formulation allows for local control at each waypoint.

3. Local Planner

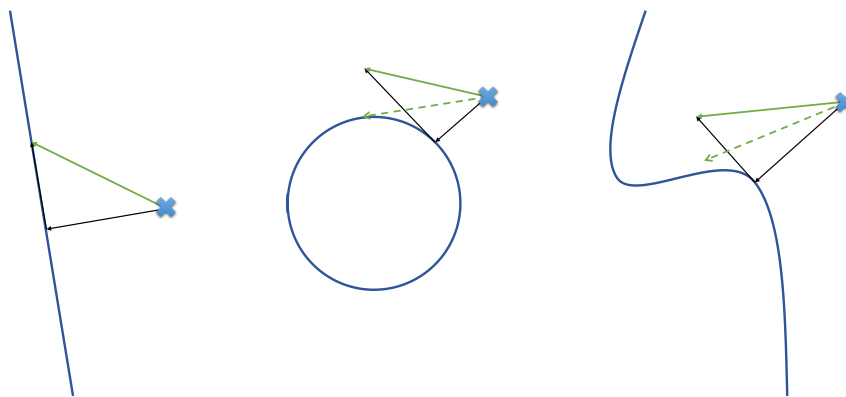


Figure 3. Three examples of the super-position in the Linear Local Planner; The X shows the position of the aircraft, and the green line is the super-position of the tangential and the orthogonal components in black. The dashed green arrow incorporates the curvature.

The trajectory that is calculated in the global planner is passed to a local planner. The local planner calculates the velocities and angular rates that are needed to converge to the trajectory. When the aircraft is moving on the trajectory,

the local planner calculates velocities and angular rates to stay on it. In the case of the *LinearLocalPlanner* in the uavAP framework, the functionality is a super-position of the movement on and towards the trajectory. A graphical representation of this super-position can be seen in Figure 3 represented by the solid green line.

Besides the super-position to calculate the target direction, the local planner can also incorporate the slope and curvature of the trajectory. For this, the planner first calculates the closest point on the trajectory to determine the local curvature and slope. For a line and orbit, the calculations of the closest point, the curvature, and the slope are straight-forward. The calculations for the *SplineGlobalPlanner* are based on the derivatives of (3). The results of adding the local curvature to the plan can be seen in Figure 3 represented by the dashed lines. A curvature target leads to an offset from the direction target of the super-position in the direction of the curvature, which allows the aircraft to converge faster.

4. Controller

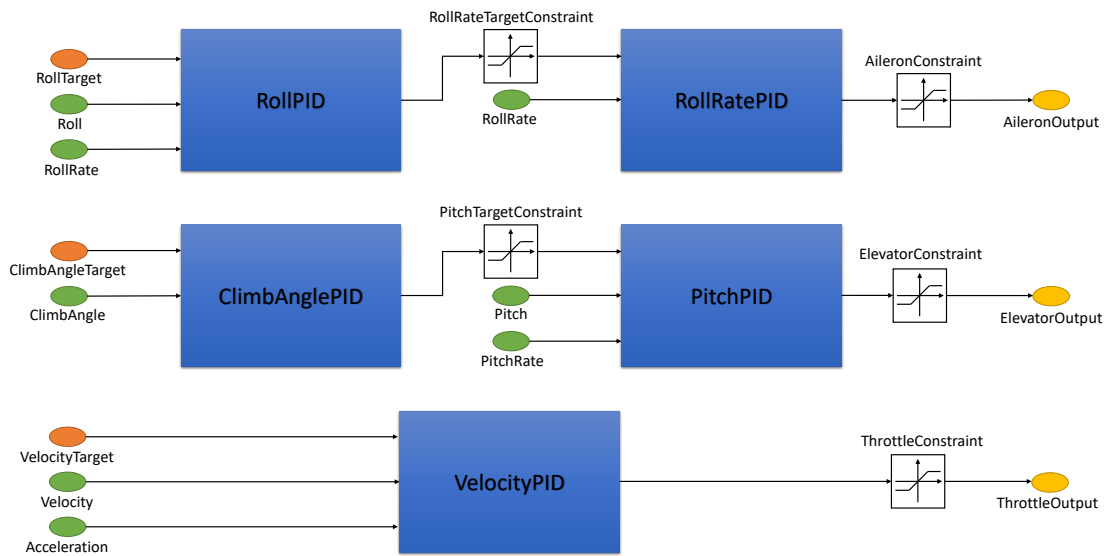


Figure 4. PID cascade: orange inputs represent the controller target from the local planner, green inputs are representing sensor data, and the yellow outputs are defining the actuation command.

Using the controller target, defining angular rate and velocity targets, the controller calculates the necessary actuation command. For its controller, uavAP uses cascaded PIDs. The schematics of the controller cascade is shown in Figure 4. The cascade consists of five PIDs that are connected in series or parallel to achieve the actuation calculation. The cascade can be separated into three different parts, yaw-rate control, climb-rate control, and velocity control. Additional PIDs can be used to actuate the rudder of the aircraft for β control.

The advantage of this PID cascade is that it is easy to set up and tune for different aircraft, using on-line tuning, as well as allowing intermediary constraints, such as the constraints on roll and pitch. Additional PIDs can be added if roll-rate or pitch-rate has to be constrained as well. On-line tuning is done using the ground station of uavAP, a part of uavEE.

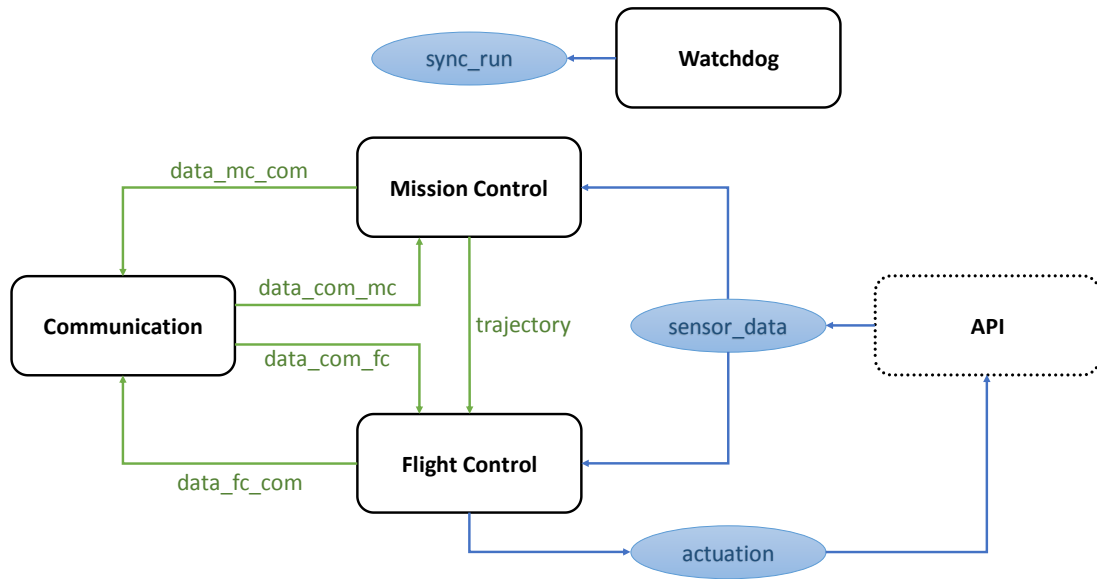


Figure 5. Control stack implementation showing processes and inter-process communication, green lines are message queues and blue lines and ellipses shared memory objects.

B. Distributed Architecture

The processes implementing the control stack and the periphery are visualized in Figure 5. Mission control is taking care of mission planning, and global planning and flight control is implementing the local planner and controller. The two are separate in order to protect the essential flight control from crashes or timing issues in mission control.

The watchdog process is the master of the synchronization in the start-up of the processes. Afterward, it monitors the processes, allowing them to restart them if they terminate or show a failure state. Additionally, the watchdog can perform strict scheduler monitoring conducting restarts of the processes if they are missing their deadlines.

The communication process is the interface of the autopilot to the outside, mainly the ground station. It is a separate process because it handles IO operations that can be slow or unpredictable. Additionally, since it is not mission-critical, if the communication process crashes, the other processes can continue unaffected. The main task of the communication process is to offer tuning, overriding, and selection interfaces to all the other processes. Additionally, it periodically sends the status information from flight control to the ground. If the user at the ground station requests the active mission and trajectory, the communication process forwards these requests to the appropriate process.

The API itself is not a process, but it is used by any background process that is responsible for the collection of sensor data and the actuation of the actuators using the actuation command. The implementation of the data collection can differ, but the API is defined in uavAP.

III. Autopilot Core – cpsCore

The core functionality of uavAP is grouped into cpsCore,⁹ a C++ framework designed to simplify the design and implementation of cyber-physical systems. The cpsCore can be used as a baseline for modular object-oriented C++ frameworks. It allows for configuration, aggregation, and synchronization of individual modules and provides utility modules for a variety of standard tasks such as scheduling. This section describes cpsCore and its role within uavAP.

A. Module Management

To facilitate uavAP's high module configurability, cpsCore contains the functionality to parse configuration files for configurable classes and create and arrange them on process start-up in a modular manner. The process uses a Helper module, a module with knowledge of all possible modules that can be created to parse the configuration, which then generates and configures the specified modules. These modules are passed to an *Aggregator*, which aggregates the modules. This aggregation is then synchronized in stages and, if successfully passing all the stages, allowed to start its schedule of tasks. This process is depicted in Figure 9. An example of the *SplineGlobalPlanner*'s usage of cpsCore is provided in Figure 6. Specific details of configuration, aggregation, and synchronization are presented in the following.

```
class SplineGlobalPlanner : public IGlobalPlanner,
    //uavAP: Its interface class
    public AggregatableObject<ILocalPlanner, IPC, DataPresentation>,
    //cpsCore: Dependencies to a local planner (uavAP), inter-process
    //      communication (cpsCore), and data presentation (cpsCore).
    //      Allows it to be aggregated.
    public ConfigurableObject<SplineGlobalPlannerParams>,
    //cpsCore: Struct with its parameters that should be configured
    public IRunnableObject
    //cpsCore: Indicating that it implements a run function for
    //      synchronization
{
    ...
};
```

Figure 6. Inheritance of the *SplineGlobalPlanner* using cpsCore's functionality.

1. Configuration

Configuration is used to make the module assembly and the modules themselves configurable. Typically a JSON file is used to define the configuration. However, support for other file types could be added easily. The configuration file indicates which objects are to be created and how their parameters are to be set. A parameter struct is used to specify the parameters of each object. An example is that of the *SplineGlobalPlanner*, as shown in Figure 7. In this struct, the default value, the corresponding string in the configuration file, and the mandatoriness, indicating if the parameter has to be specified in the configuration file, are defined. The templated `configure(Config& c)` function provides the C++ struct with reflection, a concept that allows, among others, the struct to be iterated over. A corresponding JSON type configuration file is shown in 8.

Additionally, the parameter structure is used to generate configuration files, showing all the possible parameters that can be modified. This is particularly helpful while adding new modules to help maintain configuration files. The parameter structure could further be used through other means of configuration, such as a graphical user interface, as the basic structure of configuration is templated and allows for the necessary modifications.

2. Aggregation

The concept of *Aggregation* is a decentralized solution for setting pointers to dependencies within a process. Instead of having one entity knowing all the dependencies of each module and setting all of them, an *Aggregation* of the modules is formed. Each module that is an *AggregatableObject* can browse through the *Aggregation* for its dependencies. If found, a weak pointer to the dependency is created and stored, which can be retrieved and upgraded to a shared pointer

```

struct SplineGlobalPlannerParams
{
// Parameter type      name      default      id      mandatory
Parameter<float> orbitRadius = {50.0,  "orbit_radius",  false};
Parameter<float> tau        = {0.5,   "tau",         false};
Parameter<bool>  smoothenZ  = {true,  "smoothen_z",  false};

template <typename Config>
inline void
configure(Config& c)
{
    c & orbitRadius;
    c & tau;
    c & smoothenZ;
}
};

```

Figure 7. Parameter structure of the *SplineGlobalPlanner*.

```

{
  "orbit_radius": 50.0,
  "tau": 0.5,
  "smoothen_z": true
}

```

Figure 8. Generated .json configuration file of structure in Figure 7.

using a templated `get<Dependency>()` function. The weak pointer is used to avoid circular ownership, which can lead to complications during tear down.

The *Aggregator* is the owner of the objects in a process and is therefore responsible for their destruction when the process is terminated. To do so in a predictable manner, the *Aggregator* first stops active subscriptions, to avoid triggers from other processes. Second, the scheduler is stopped, descheduling all of the periodic events. Finally, the *Aggregation* container of the *Aggregator* can be emptied, destroying the aggregated objects sequentially.

3. Synchronization

Synchronization in a distributed system is a crucial factor for clean and predictable behavior. Especially if there are dependencies between processes that need to be established first, synchronizing the start-up phase is crucial. In uavAP synchronization is conducted among the modules inside one process, and among the processes in a distributed multi-process setup.

As described before, the schematic, shown in Process 2 of Figure 9, illustrates the start-up steps of one single process. For synchronization, a Runner utility sequentially triggers the current run stage for each module before moving on to the next stage. It first triggers the INIT run stage. In this stage, each module should check if all its dependencies are met. If not, the Runner aborts and prints corresponding error messages. In run stage NORMAL, the objects schedule their tasks or communicate with other objects to set up the process. Run stage FINAL is reserved for tasks that need three steps to set up. After run stage FINAL, the scheduler is triggered to start its schedule.

For multi-process synchronization, as necessary in Figure 5, the single process case is extended. An entity, such as a Watchdog, starts all the desired processes, waiting until they all reach the beginning of run stage SYNC, which is an idle run stage that is used to wait for the other processes. When all the processes reached that point, the Watchdog

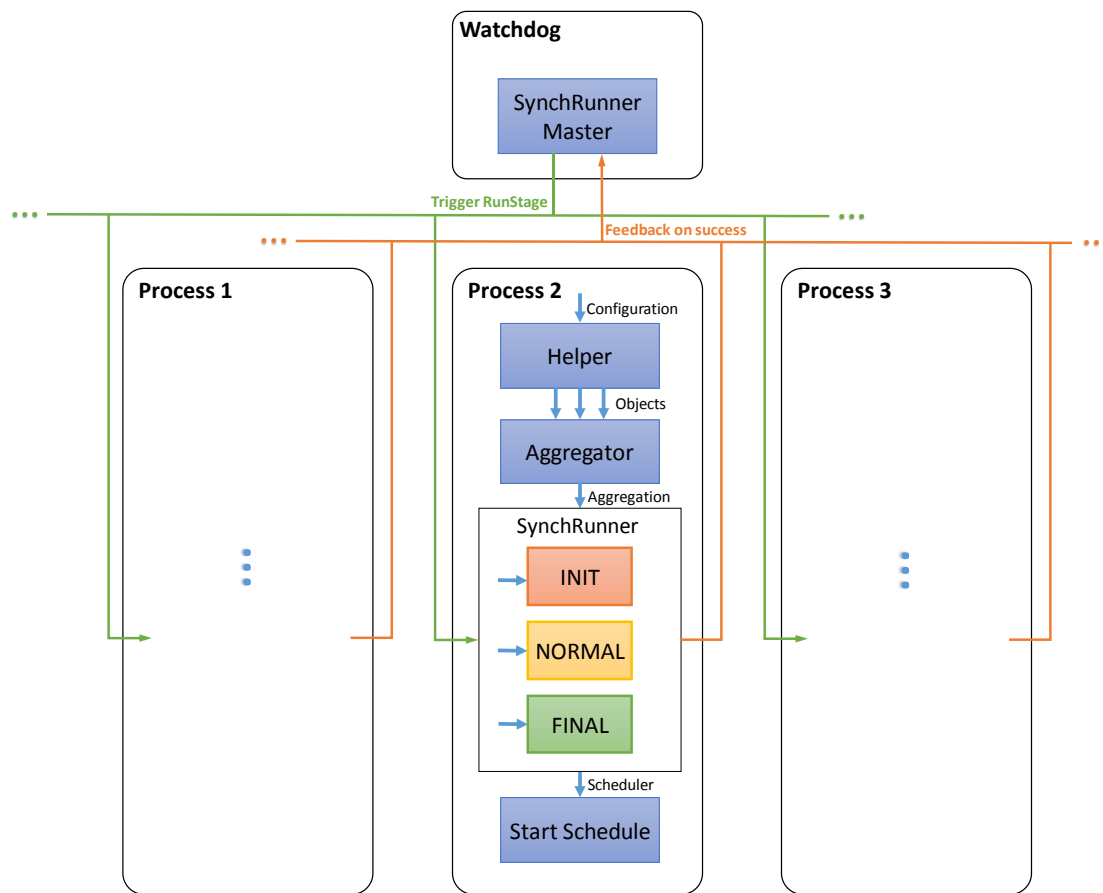


Figure 9. Multi process synchronization in uavAP

triggers the run stage INIT. All the processes now run their INIT run stage. If they succeed and do not discover a problem, they notify the Watchdog that they succeeded. This is handled by using thread barriers with a count of the number of processes. If they do not succeed, they do not notify the Watchdog leading to a time-out that lets the user know that one of the processes failed. After run stage INIT, the same synchronization procedure is executed for run stage NORMAL, followed by FINAL. After every process runs into the thread barrier of stage FINAL, they start their schedule simultaneously. In the multi-process case, the synchronization information is shared with a segment of shared memory, maintained by the synchronization master, e.g., the Watchdog.

B. Core Utilities

The core utilities of cpsCore comprise of functionality that is frequently used in CPS applications such as uavAP or also uavEE. They are implemented to be as generic as possible while allowing low-level optimization. These core utilities are used for scheduling, timing, inter-process communication, inter-device communication, and data presentation.

1. Scheduling and Timing

A scheduler handles every scheduling of periodic and non-periodic tasks in the autopilot. Any scheduler that is implemented should provide the ability to schedule periodic and non-periodic events as well as the possibility to start

and stop the schedule. The current main scheduler of uavAP is the *MultiThreadingScheduler*, which, as indicated by its name, uses multiple threads to execute tasks in parallel. When a task is scheduled, the scheduler creates an event object. Each event contains the function handle, scheduling information, a condition variable, and a thread for execution. The scheduler triggers the individual condition variables at the time of the task release. After completion of their tasks, the threads either end execution, if they were non-periodic or canceled, or wait until called again. The time provider handles the timing of the scheduler.

Any time provider in cpsCore has to provide the current time and the ability to sleep for a set amount of time or until a specific time point. The time provider used in uavAP is the *SystemTimeProvider*, which uses the standard chrono time library to provide time information as well as timing functionality, such as *wait for* or *wait until*. For manual time and scheduling control, essential for unit testing, the *MicroSimulator* can be used. It provides objects with manual time and scheduling information allowing full control during testing.

2. *Inter-Process Communication*

For the communication among processes, such as *Mission Control* and *Flight Control*, cpsCore offers inter-process communication (IPC) utilities. The IPC module allows message passing to one or multiple destinations. If communication to only one destination is requested, the IPC module creates a message queue to which the destination process can subscribe. For multiple destinations, the module creates a shared memory segment, allocating space for the data as well as synchronization fields. The destination processes can find the message queues or shared memory segments via string IDs. The general implementation is similar to message brokering, allowing publication and subscription.

3. *Inter-Device Communication*

To communicate with other devices, such as the ground station in uavAP, cpsCore provides utilities for inter-device communication (IDC). The IDC is split up into two layers, the transport layer, and the network layer. The transport layer is a current place holder if packet segmentation and acknowledgments are to be added. The network layer takes care of the dissemination of the packets to their destination. For that, it can use serial communication using the boost asio backbone, or ethernet communication using Redis,¹⁰ specifically its message broker service. For packet verification, crucial for radio communication, a cyclic redundancy checksum (CRC) is implemented. The checksum is appended to each packet and can be verified on the receiving end.

The network and transport layer functionality are hidden from the other modules by providing an IDC module, which routes the packets according to configured string IDs. This way, the communication method can be changed through configuration and does not require recompilation or rewriting of code.

4. *Data Presentation*

For IDC and IPC, the data structures in uavAP have to be represented as binary. For passive data structures or plain old data (POD), this binary representation is as simple as a memory copy, provided the sender and receiver device use the same endianness. For more complicated structures with optional fields or nested members, cpsCore provides a data presentation utility. Similar to the configuration, the data presentation adds functions for code reflection, which specify how to serialize and deserialize structures. Using these functions, data presentation creates string-based packets from the complex structures which can be sent via IPC or appended with headers and send via IDC.

IV. Flight Maneuver Automation Framework – uavAP Extension

The introduced uavAP software framework allows for a simple and configurable extension of a flight maneuver automation framework on top of the existing software stack. The flight maneuver automation framework utilizes the introduced cpsCore framework to extend the current mission planner module, enabling automatic maneuver executions and transitions. A new flight analysis process is also added for providing the extended mission planner module with various aircraft states analysis needed for automating the aircraft flight maneuvers.

The current mainstream approach for collecting aircraft aerodynamic parameters is to manually pilot UAVs through a series of flight testing maneuvers.^{11–15} Automating such flight testing maneuvers,^{5,6} on the other hand, allows for the process of aircraft parametrization and modeling to be performed systematically and repeatably with minimal trial-and-error, and, more importantly, reduces the required amount of flight time and power consumption. For instance, by automating maneuvers during the flight, aircraft states such as attitude angles and velocity vectors can be set and maintained by controllers with better accuracy, consistency, and repeatability than manual piloting.

As the industry of small UAVs becomes increasingly popular, the safety and regulation for these small aircraft are also becoming essential for their applications and deployment. One of the useful and practical methods of executing the safety regulations on those small aircraft is to require them to have mandatory and built-in geo-fencing systems that provide constraints to their behaviors and missions. The flight maneuver automation framework, together with a robust and precise kinematic model detailed in,^{7,16} forms an advanced geo-fencing system for UAVs to perform trajectory modeling, boundary checking, and automated evasive maneuvers.

A. Maneuver Planner

As discussed in Section II, the mission planner module in the uavAP software framework provides high-level mission planning and global plan generation. The existing global planner in the mission planner module takes mission waypoints as input parameters and generates a position-based trajectory as its output. The generated trajectory is then passed to the flight control module for local planning and controller target generation. In the above pipeline, flight maneuvers are generated by the local planner as controller targets for keeping the aircraft on the planned trajectories.

The existing global planner is useful for simple and fixed-path missions such as a race track flight path illustrated in Section V for surveying and power modeling. However, such missions limit the UAVs to fixed, position-based trajectories which prevent the aircraft from performing customized maneuvers aside from path-keeping. Therefore, in order to achieve more advanced autonomous UAV applications such as flight testing maneuver automation and geo-fencing, a more versatile mission planner is needed for planning and sequencing ad-hoc and customized flight maneuvers.

The new maneuver planner extends and augments the uavAP planning and control stack to achieve customized and flexible planning. Specifically, the maneuver planner generates user-defined flight maneuvers into override objects through the cpsCore configuration framework detailed in Section III. Such maneuvers, containing local plans, controller targets, controller outputs, and more, are published through cpsCore's IPC, as illustrated in Figure 10, to the respective modules, in which the regular mission is overridden until after executing the received flight maneuvers.

In this design, the maneuver planner provides versatile, trajectory-independent capabilities in terms of aircraft states. For instance, if a 45 degree right-rolling maneuver were needed for a particular application, the maneuver planner would simply generate a flight maneuver containing a roll angle controller target of 45 degrees and publishes such maneuver to the controller module for execution. When the maneuver was executed, the aircraft would override the regular trajectory and roll right at 45 degrees from its current state in a trajectory-free manner.

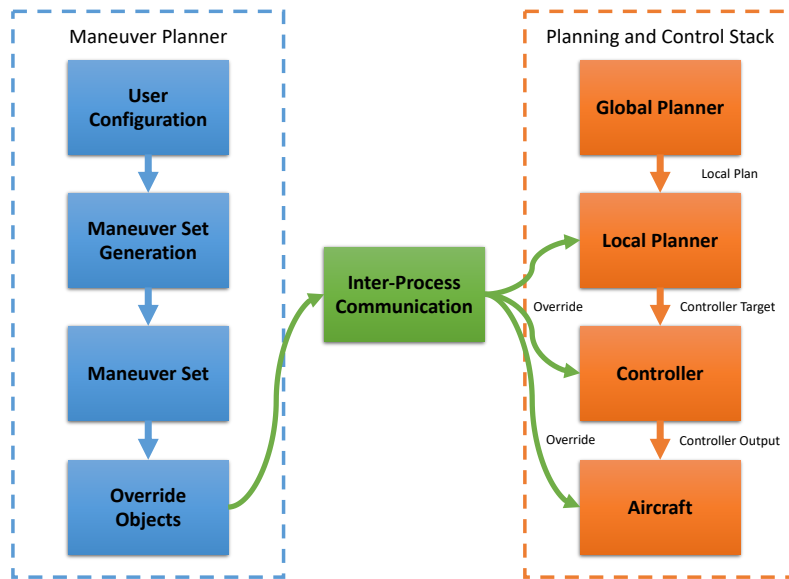


Figure 10. The maneuver planner pipeline and the uavAP planning and control stack (mission planner omitted). The override objects, representing the maneuvers, are generated by the maneuver planner, published through the cpsCore IPC framework discussed in Section III, and are subscribed by the uavAP stack for maneuver execution.

Furthermore, the maneuver planner is also capable of concatenating a series of individual flight maneuvers into a maneuver set, as shown in Figure 11, and executing through the set sequentially under some predefined transition conditions. Similar to a finite state machine (FSM), the maneuver planner stays at the current maneuver in a maneuver set and only continues to the next maneuver when the transition condition is met. When all the maneuvers in the maneuver set are exhausted, the maneuver planner halts, and the aircraft returns to its regular mission.

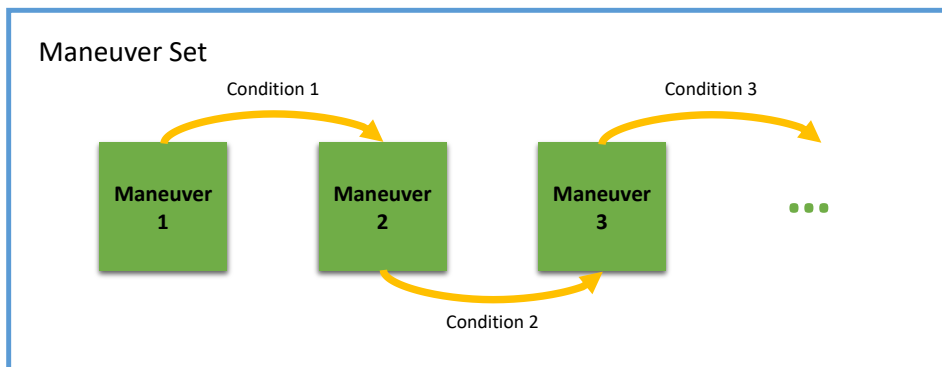


Figure 11. An example of a maneuver set, containing a series of individual flight maneuvers connected by their transition conditions.

B. Flight Analysis

For automating customized flight maneuvers with various traits, a new flight analysis process is needed in addition to the maneuver planner for analyzing various aspects of the aircraft during the flight. First of all, the analysis data provided by the flight analysis module can be used by either the uavAP modules during the flight or by post-processing programs through data collections after the flight. For example, during the execution of a particular maneuver set with

the maneuver planner, states of the aircraft are useful for post-processing, graphical visualization, scientific research and validation, and so on.

More importantly, automating the customized flight maneuvers often requires information about various aircraft states during the flight, such as whether the controllers have reached their steady states, aircraft control surface trims, how much time has elapsed since the start of the current maneuver, etc.⁵ Maneuvers in many applications should transition to the next maneuver only when the roll angle controller of the aircraft has reached its steady state, i.e., the controller has stabilized around its given target. The flight analysis module would provide this steady-state information and enable the enforcement of such transitions.

V. Applications

The uavAP autopilot has thus far been applied to 3 platforms: the prototyping ecosystem emulation environment, uavEE; a robust, fixed-wing, testbed unmanned aircraft, the UIUC Avistar UAV; and a long-endurance, computationally-intensive, solar-powered unmanned aircraft, the UIUC-TUM Solar Flyer. Between these 3 platforms, the uavAP autopilot has enabled: the rapid prototyping of flight modeling and control algorithms in emulation and real flight, the design of an accurate UAV power model based on the physical model of the aircraft, flight maneuver automation for aircraft system identifications and dynamics parametrization, an algorithm for geo-fencing of fixed-wing UAVs, and power-efficient flight through a turbulent and windy environment.

A. uavEE Emulation Environment

In order to decrease development time, emulation and modeling has become an important component of the UAV development process. Instead of prototyping, testing, and analyzing through the many stages of aircraft development in hardware, which is resource and time intensive, a virtual aircraft and its sub-systems were modeled and then implemented into the uavEE emulation environment.⁴ Specifically, the environment starts by creating a real-time connection between a high-fidelity flight simulator (e.g. X-Plane 11) and an autopilot software, i.e., uavAP on a desktop machine or embedded hardware, and then modeling layers are introduced (e.g. power, communication, fault, etc.), allowing for additional emulation complexity. Therefore, the physical aircraft design, the software, and the flight computation and possibly payloads can be tested in the lab. Within the scope of applying the uavAP autopilot to research tasks, uavEE was used to emulate each of the research efforts presented in the following subsections before they were tested on an actual aircraft. uavEE also provides the backbone for a ground control interface, shown in Figure 12, which is used to command and monitor the aircraft and autopilot in both emulation as well as in real flight.

B. Avistar UAV Testbed

The Avistar UAV is a highly-robust, fixed-wing unmanned aircraft, which has been used as the testbed platform for a variety of flight software and hardware development efforts.¹⁷⁻²¹ The aircraft was developed from the Great Planes Avistar Elite fixed-wing trainer-type radio control model and has wingspan of 1.59 m and a mass of 3.70 kg. The completed flight-ready aircraft is shown in Figure 13 and its physical and component specifications can be found in previous work.²² The uavAP autopilot was integrated into the Al Volo flight control and data acquisition system installed in the Avistar UAV in order to enable several avenues of research: a high-fidelity, low-order propulsion power model for fixed-wing electric unmanned aircraft, a flight testing automation tool for aircraft system identifications and dynamics parametrization, and an algorithm for geo-fencing of fixed-wing UAVs.

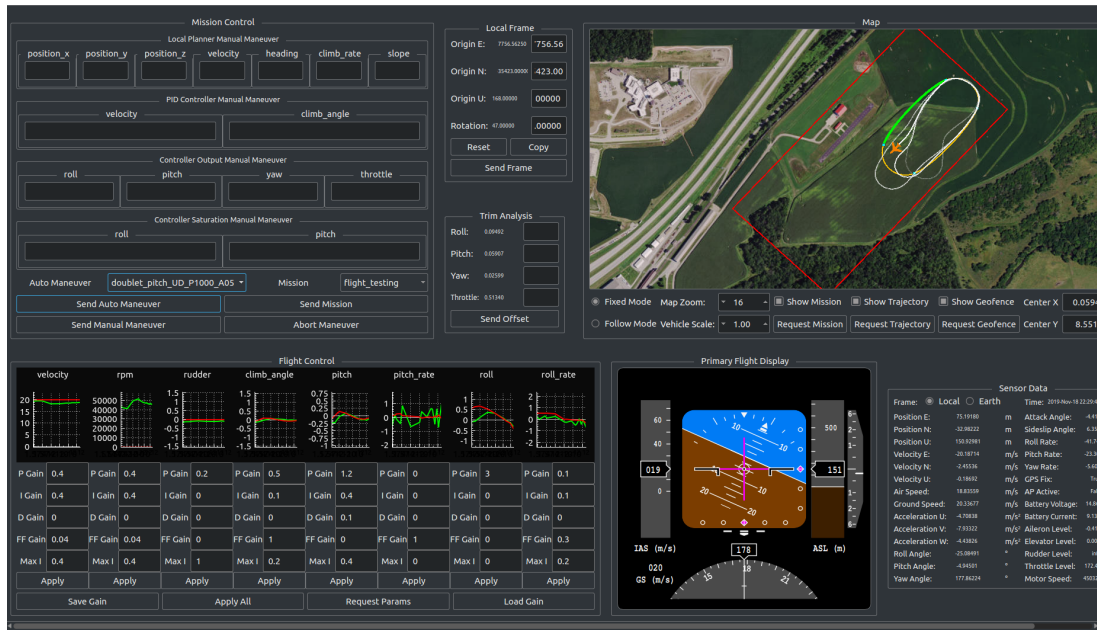


Figure 12. The uavEE ground station interface, which provides functionality in emulation and real flight; it is shown being used to automate a pitch doublet maneuver as part of the flight testing automation process.



Figure 13. Flight-ready Avistar UAV.

1. Propulsion Power Model for Fixed-Wing Electric Unmanned Aircraft

A high-fidelity, low-order power model for electric, fixed-wing unmanned aircraft¹⁹ was developed and integrated into uavEE. Previous works have separately looked at aircraft power modelling^{23–25} and propulsion system modelling^{26–28} with varying degrees of assumptions and verification. Compared to existing works, the propulsion power model developed provides a more holistic approach to UAV propulsion power modeling and has been tested under realistic flight conditions. The power model uses propulsion system modeling of the propeller and motor as well as aircraft power modeling using flight mechanics derivations. In order to enable online computation with limited resources, the resulting expression has been limited to using only measurable aircraft state variables, propulsion system parameters and curves, and (scalar) constants. The final expression for the developed power model is:

$$P_{propulsion} = K_p \frac{v^3}{\eta_p \eta_m} + K_i \frac{\cos^2 \gamma}{\eta_p \eta_m v \cos^2 \phi} + mg \frac{v \sin \gamma}{\eta_p \eta_m} + m \frac{\vec{a} \cdot \vec{v}}{\eta_p \eta_m} \quad (5)$$

where K_p and K_i are scalar constants that can be determined from aircraft specifications or can be learned through linear regression with non-linear kernel using a training data set. Note that complete derivation and validation can be found in related work.^{4, 19}

The resulting power model was evaluated by means of flight testing using uavAP. By flying a reference flight path, containing turns, climbs, descents, and straight line segments, the flight testing showed very close agreement between the power and energy estimates determined using the power model from aircraft state data and actual experimental power and energy measurements. Additionally, using the emulation environment, the reference flight path was also flown using the same autopilot and a simulated radio control model aircraft trainer, which was very similar to the one used in experimental flight testing. These flight paths are displayed in Figure 14. The flight path was nearly identical with the exception of 2 corners, where in experimental flight testing, light wind gusts deviated the aircraft slightly. The power and energy data generated was in close agreement with the experimental data as can be seen in Figure 15. The significance of this result is that the developed propulsion power model is able to accurately estimate the power consumption of an electric UAV based on flight path state, without needing precise aerodynamic measurements or estimation, e.g. angle-of-attack. Therefore, power estimation can be done with minimal computation.

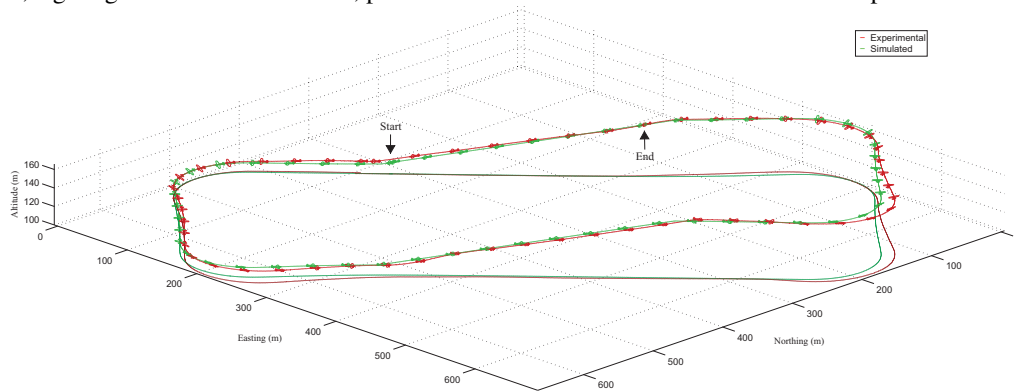


Figure 14. Comparison of aircraft path for experimental (red) and simulated flight (green) results; the airplane is plotted at 6x scale and every 2 seconds.

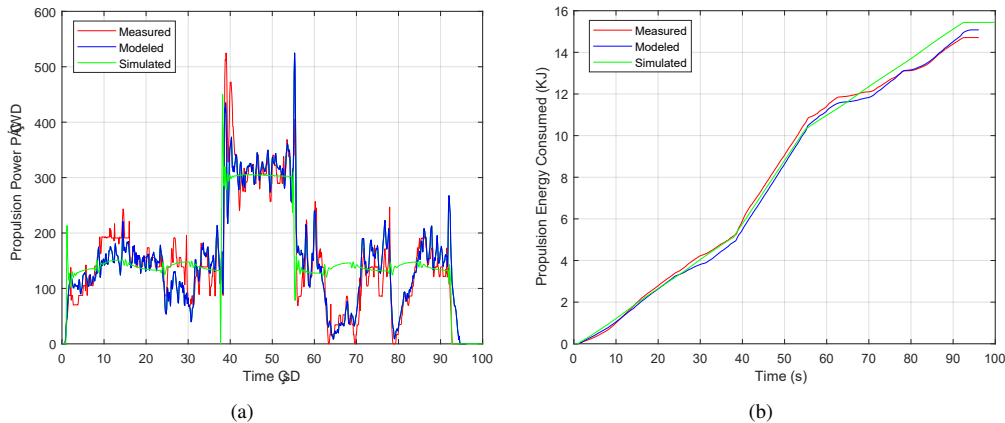


Figure 15. Comparison of (a) propulsion power and (b) energy consumed from experimental measured (red), experimental modeled (blue), and simulated (green) results using the propulsion power model.

2. Flight Testing Automation

The current state-of-the-art in performing flight testing maneuvers for aircraft parameterization has been manual piloting with instruction relayed through flight test cards. Performing manual flight maneuvers off of manually read test cards has shown to require thousands of hours of costly flight testing.²⁹ There have been ongoing efforts to parameterize aircraft dynamics on manned and unmanned aircraft using multi-sine and stick-shaker inputs. However, these can be error-prone, very computationally intensive, and require large datasets.^{30–32} Instead, the flight maneuver automation framework, described in Section IV, extends the existing uavAP stack to streamline the flight testing and flight dynamics

parameterization processes of an unmanned aircraft.⁶ The flight maneuver automation framework is able to command the aircraft through a user-defined, conditional set of motions and states to induce certain maneuver sets, which allow for dynamics to be more easily parameterized; these sets of maneuvers, motions, and states follow manned flight testing techniques.³³ Maneuvers of interest that have been implemented into the automation framework included: idle descent, stall, phugoid, doublets, and singlets, which provide the basis for determining the aircraft aerodynamics, longitudinal stability, and control effectiveness, respectively. Additionally, automating the data collection process using the new flight analysis module allows for reliable data selecting, eliminating work hours of parsing and matching data ranges to maneuvers.

The flight maneuver automation framework was initially demonstrated using software-in-the-loop simulation in the uavEE. A comparison between automated and manually-piloted flight was performed for testing stall using the full-scale Cessna 172 under ideal (still atmosphere) conditions in the X-Plane 11 Flight Simulator^a; this can be seen in Figure 16–19. In those time histories, the difficulty exhibited by the trained human pilot in simultaneously controlling the aircraft altitude and roll and heading angles can be seen. In comparison, the time history of autonomously controlled stall speed maneuver show smooth and accurate results. The flight maneuver automation framework was then demonstrated on the Avistar UAV testbed aircraft and subsequently used to collect an aircraft dataset. Due to limited calm weather day opportunities, only a subset of the maneuvers developed were flown, which include stall speed, stall polar, idle descent, singlets, and doublets. The complete resulting data set of flight testing maneuvers can be viewed in related work²¹ and can be downloaded from our UAV Database Site.³⁴

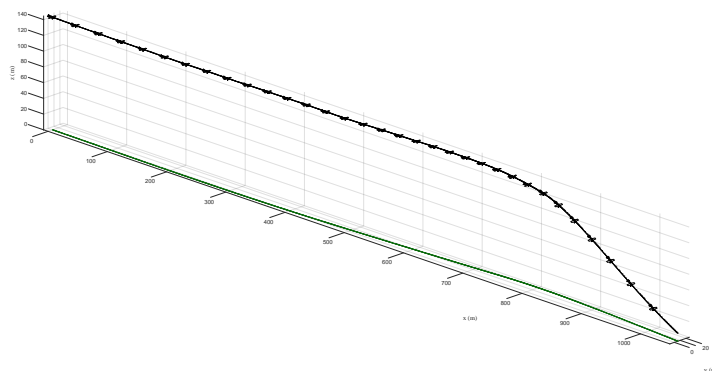


Figure 16. Trajectory plot of a stall speed maneuver performed by manual piloting (the aircraft is drawn once every 1.0 s).

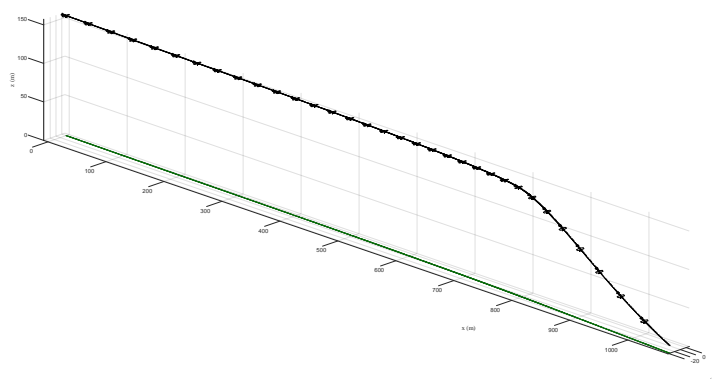


Figure 17. Trajectory plot of a stall speed maneuver performed by the flight maneuver automation framework (the aircraft is drawn once every 1.0 s).

^aThe manually-pilot aircraft was flown by a trained human pilot using a professional-grade simulator yoke system, throttle quadrant, and rudder pedals. Both maneuvers were set up the same, with the aircraft flying at 40 m/s and oriented at a yaw angle of 0 deg (East).

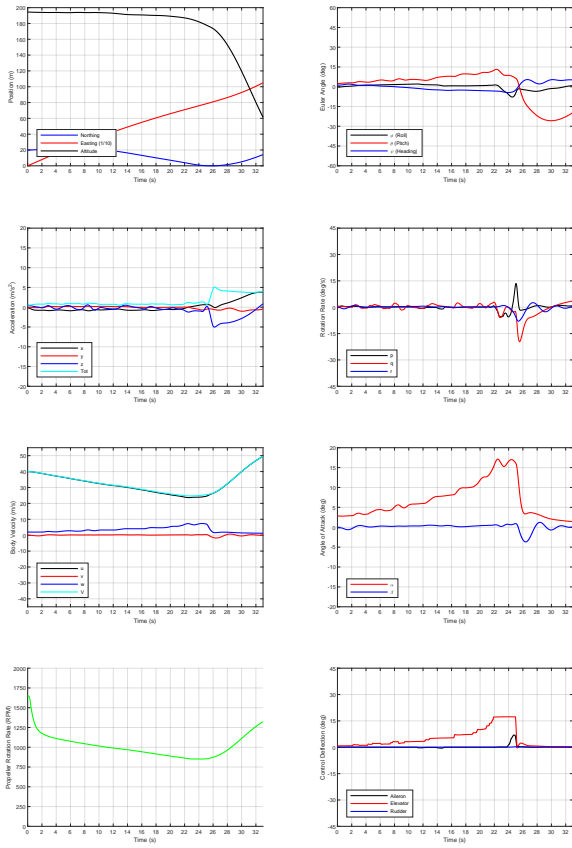


Figure 18. A time history of a stall speed maneuver performed by manual piloting.

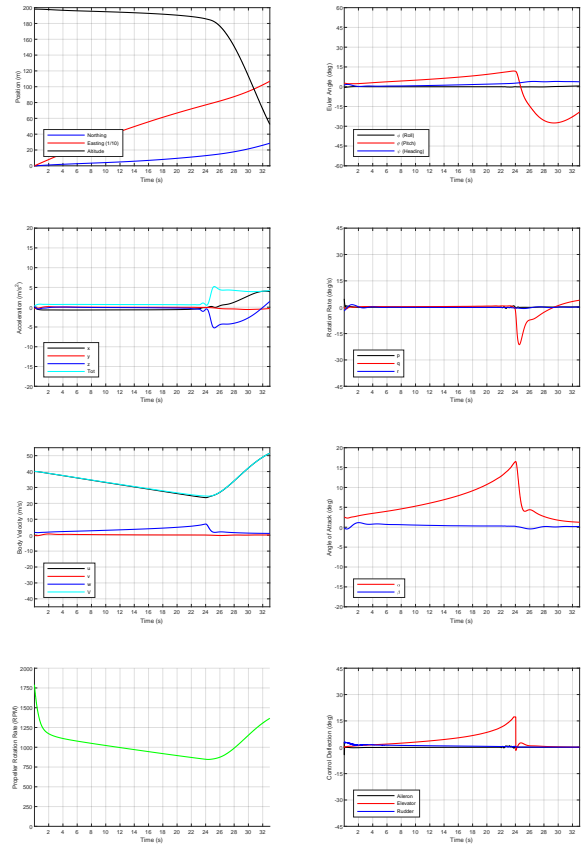


Figure 19. A time history of a stall speed maneuver performed by the maneuver automator.

3. Geo-Fencing Algorithm

To enable safe interactions among the surrounding humans, environments, and other aircraft, UAVs have to be constrained to designated areas or spaces. For rotary aircraft, such as quadcopters, the task of enforcing the geo-fence is relatively simple since those type of aircraft are capable of stopping in mid-air and turning around with zero translational velocity. However, for fixed-wing aircraft, such execution of maneuvers is impossible as they need to maintain a minimum velocity in order to stay airborne. Consequently, a precise kinematic model for fixed-wing aircraft is required to determine the feasibility of a trajectory as well as the exact time for the initiation of an evasion maneuver. Most analytical kinematic models only constrain the maximum curvature of a trajectory, namely a Dubin's Curve.^{35,36} These fixed-wing aircraft geo-fencing algorithms^{37,38} argue that the trajectories for a fixed-wing aircraft form a symmetric fan pattern around the velocity vector. This fan pattern, however, is based on the instantaneous change in roll, which was shown to not be applicable for fixed-wing aircraft.

Therefore, a precise kinematic model, the Beta-Trajectory, was developed for trajectory prediction and evasive maneuvering.⁷ The Beta-Trajectory implements a kinematic model for fixed-wing aircraft where roll is governed by constrained roll rates, yielding Beta-Curves. The algorithm then uses the results of the model to avoid boundaries and stay in a designated area. The full derivation of the Beta-Trajectory can be found in this technical report.¹⁶ In order to evaluate the proposed geo-fencing system, the Beta-Trajectory was implemented into uavAP using the flight maneuver automation framework, tested in uavEE, and then subsequently tested in real flights using the Avistar UAV. Figure 20 shows the performance of the Beta-Trajectory geo-fencing algorithm in real flights.

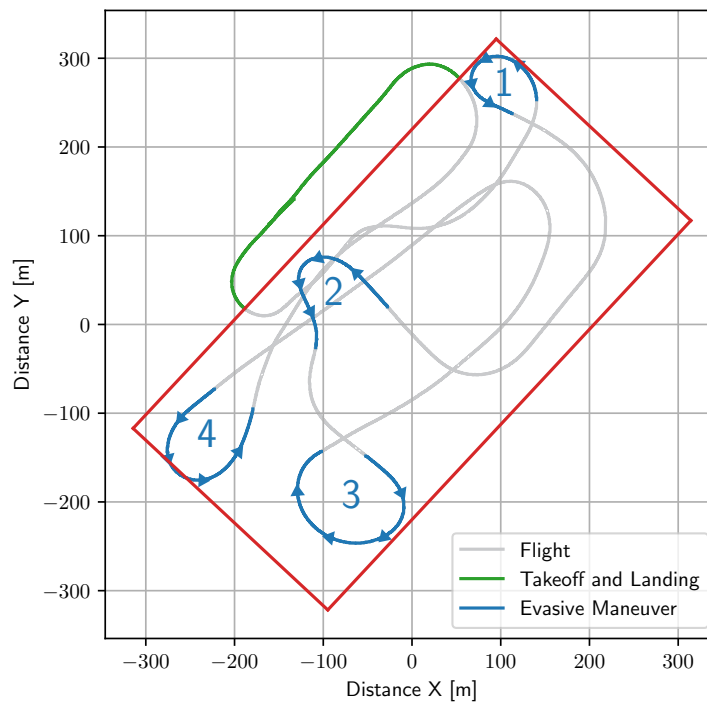


Figure 20. The real-life flight path of the Avistar UAV deployed with the uavAP autopilot with geo-fencing; red shows the geo-fence, blue shows the evasive maneuvers labeled from 1-4 (note that the boundary slack value is 5 meters and velocity is 20 m/s).

C. UIUC-TUM Solar Flyer

The UIUC-TUM Solar Flyer, which is shown in Figure 21, is a long-endurance, solar-powered unmanned aircraft currently in development to enable computationally-intensive flight that could support real-time data processing for a wide range of applications.³⁹ The aircraft is a highly-optimized, fixed-wing design that was assembled from only commercial-off-the-shelf (COTS) components and operates a narrow range of airspeeds in order to achieve highly-efficient flight. The aircraft has been instrumented with a custom AI Volo flight control and data acquisition system that integrates the uavAP autopilot, which has been adapted for the demanding requirements dictated by the aircraft and its flight profile.

Due to the low operating airspeed of the sailplane design, the aircraft is very susceptible to atmospheric turbulence and wind. Therefore it was crucial to integrate a responsive wind estimator and airspeed controller into uavAP. Among other flight testing recently performed, the UIUC-TUM Solar Flyer was autonomously flown using uavAP to measure the aircraft's power consumption,⁴⁰ which is crucial for modeling the aircraft. Additionally, in order to verify the aircraft's ability to maintain a precise flight path under varying flight conditions, the aircraft was flight tested using uavAP in various amounts of wind, up to the aircraft's typical cruise speed (note that the aircraft maximum speed is greater than cruise). Figure 22 shows the resulting trajectory, of the UIUC-TUM Solar Flyer attempting to maintain a repeated level race track maneuver under varying wind conditions, which would be sufficient to accomplish a typical mission profile, e.g. equivalent zig-zagged racetrack coverage profile over a field.



Figure 21. The UIUC-TUM Solar Flyer aircraft shown with solar arrays.

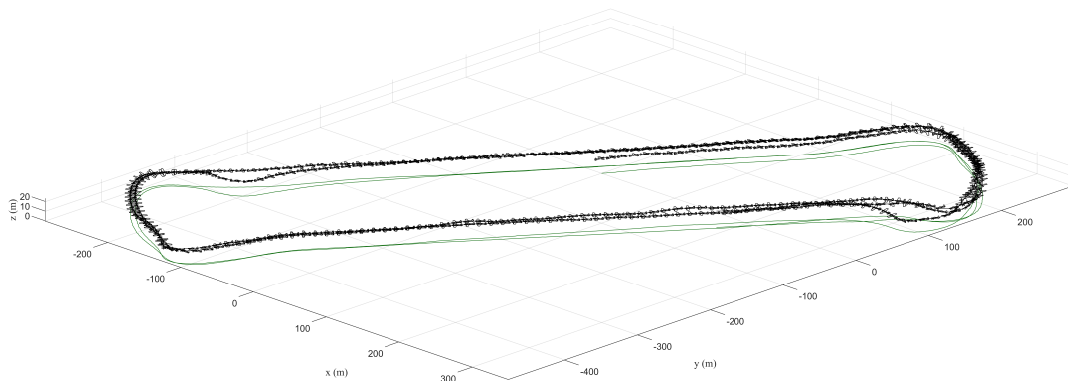


Figure 22. The trajectory of the UIUC-TUM Solar Flyer during a repeated level race track maneuver (note that the aircraft is plotted 6x scale every 0.5 sec).

VI. Related Autopilots

In this section, a comparison between uavAP and other openly available autopilot implementations is performed. The control stacks and software framework of Ardupilot, Paparazzi and PX4 are examined.

A. Ardupilot

Ardupilot is an open source software suite first established in 2009. It is not tied to a specific set of hardware but rather, it is firmware capable of running on various embedded platforms.⁴¹ Nevertheless, fully packaged autopilots with hardware and software, such as the The ArduPilot Mega (APM) based off the Arduino Mega, exist for convenience. APM has inspired many derivatives, such as FlyMaple.^{42,43} Ardupilot's runtime stack is organized hierarchically as follows. The lowest layer is the hardware level, consisting of external sensors, open hardware standards such as Pixhawk,⁴⁴ controller chips such as the Mateksys F405⁴⁵ or Navio2, and complete drones such as the Bebop2.⁴⁶ The next level is the OS layer, which consists of OSs like ChibiOS, BusyBox Linux and Linux.⁴¹ Ardupilot itself then runs above the OS layer. The flight code is further segmented into 3 layers, the hardware abstraction layer, shared libraries, and vehicle specific flight code. The hardware abstraction layer allows ArduPilot to be portable and platform agnostic. Shared libraries exist for the supported four vehicle types: Copter, Plane, Rover and AntennaTracker. The communication layer resides above the flight layer and communication is done via devices implementing the MAVLink protocol. In addition to these core autopilot modules, Ardupilot's codebase also has miscellaneous support tools. The highest layer of the runtime stack is the UI/API layer, which consists of the Ground Station and any DroneKit applications and their corresponding hosts.⁴¹ In terms of software design, Ardupilot is focused on reliably going from one waypoint to another. Ardupilot provides a reliable trajectory planner to travel between waypoints and assumes the average user is not interested in modifying the trajectory planner or other critical features. uavAP allows users to modify the trajectory planner, scheduler, inter device communication, and various other low level features if they so desire. This modularity allows users to implement more complex solutions ranging from low level to high level. Thus, uavAP is more geared towards being a testbed for various state-of-the-art research implementations.

uavAP's runtime stack is comparable to Ardupilot's. At the hardware layer, uavAP uses Al Volo libraries to communicate with sensors and data acquisition systems but can be extended to other platforms. One noteworthy difference is that below the hardware level, uavAP can also be configured to take actuation and flight data from real flights or emulated/replayed flight conditions in uavEE. uavEE is then capable of performing both SITL (Software in the Loop) and HITL (Hardware in the Loop) Simulation, with flight simulators such as X-Plane 11. Conversely, Ardupilot has its own SITL (Software In The Loop) simulation framework and simulator. It is a build of the autopilot using any C++ compiler, and thus allows the autopilot to be tested without hardware. Ardupilot's SITL Simulator can also be used with a wide variety of 3rd party simulators, such as Gazebo, XPlane-10, RealFlight, Morse, Replay, JSBSIM, AirSim, Silent Wings Soaring, Last Letter, CRRCsim, or SCRIMAGE. Hardware In The Loop Simulation is currently only supported for planes in X-Plane and FlightGear.⁴¹ While Ardupilot's SITL necessitates manual connections from the autopilot executable to the ground control station, physics and flight simulators, and proxy telemetry if multiple ground control stations are desired, uavAP uses the uavEE system to handle all inter-agent communication associated with HITL/SITL simulation. Ardupilot's fixed wing operating modes are comparable to uavAP. Ardupilot has various different flight modes, ranging from full manual control of aircraft control surfaces, to roll and pitch override, to circling a point, to following a mission. In AUTO mode, where the autopilot flies a mission, Ardupilot's framework allows the ground station to update the mission.⁴¹ Conversely in uavAP, manual flight is not supported by default. In the interest of autonomy, the mission is preconfigured ahead of the flight.

B. Paparazzi

Paparazzi UAV is another open-source project encompassing both the hardware and software aspects of UAV systems. As per their website, they support more target platforms than Ardupilot.⁴⁷ One example of a pre-built board running the autopilot software is the STM based Chimera board.^{42,43,47} Paparazzi's usage flow is as follows. The autopilot is configurable by an XML, where the flight mode state machine is defined, along with aircraft modules, additional header files, ground control settings, and exceptions. The specified firmware is then built and cross-compiled for that target aircraft hardware, and uploaded to the embedded board. Paparazzi has a wide array of modules for performing tasks such as reading external sensors or controlling cameras. The default features for fixed wing aircraft are the following: manual control via an RC transmitter, RC receiver, servo and motor control, control with augmented stability (AUTO1), autonomous navigation (AUTO2), and communication to and from the ground station. Autonomous navigation includes features like waypoint navigation, segment and circle navigation, takeoff and landing, and advanced fail-safe planning (e.g. geo-fencing).⁴⁷ Its configurable state machine nature also allows high flexibility and complexity in algorithm design for control, communication and other custom features. In comparison, uavAP provides a concrete control stack with defined roles, with the goal of minimizing complexity associated with changing autopilot functionality.

The system communication flow is as follows. When configured for real flight, the aircraft communicates over a wireless link to a ground network, which then sends the data to a server that logs, distributes, and pre-processes the messages for the ground control station and other ground agents. When configured for simulated flight, the hardware communication link is replaced with a SITL simulator that simulates actuation and radio communication. A Gaia agent then allows the user to set environmental variables such as windspeed and direction, sensor failure, and flight simulation speed. Paparazzi has two built in simulators, sim and nps (New Paparazzi Simulator). It also supports the Gazebo simulator.⁴⁷ In comparison to uavAP, uavAP sends sensor data and receives ground commands in real flight over a radio link to a uavEE environment with radio communication and ground station nodes. In simulated flight, sensor data from the flight simulator is sent over a simulated serial link to uavAP and processed with the same API. Environmental factors such as wind can be configured in the flight simulator (currently X-Plane 11). All simulated sensor data can be corrupted to simulate sensor fault, and all communication between simulation peripherals (i.e. flight simulator, power modeller, sensor fault modeller) is handled by the ROS environment uavEE is built on.

C. PX4

PX4 is another open source autopilot for drones and other unmanned aerial vehicles focused on support for a broad category of aerial vehicles, sensors and control hardware, and safe flight modes. It comes with a ground station called QGroundControl, supports PixHawk hardware, and uses the MAVSDK library for communication with companion devices using the MAVLink protocol.⁴⁸ Various embedded boards are designed to use the PX4 autopilot, such as PIXHAWK, Pixfalcon, and PixRacer.^{42,43} PX4 is split into a flight stack layer and middleware layer. The flight stack layer is responsible for all flight control tasks, such as guidance, navigation, control algorithms, and reading sensors. The workflow inside the flight stack is as follows. An estimator feeds a state estimate to a controller, which produces a command, and a mixer translates them to motor commands. This layer is vehicle specific and depends on factors such as the aircraft's motor arrangements and rotational inertia. PX4 uses a state machine in the flight controller to select a flight controller based on the level of flight autonomy desired.⁴⁸ The middleware layer handles communication with the external world and it includes device drivers for embedded sensors and communication with companion computers, ground control stations, etc. Similar to Ardupilot, PX4 provides broad community support and is focused on simple and reliable flight control. Thus, PX4 focuses on providing robust semi-autonomous flight (e.g. pitch and roll controlled by the autopilot but yaw manually controlled by RC stick) and autonomous waypoint following and assumes the average user is not interested in modifying the trajectory planner or other critical underlying features. In contrast, uavAP's

modular framework is designed to allow modifications to any underlying feature, making it more suitable to be a testbed for various state-of-the-art research implementations.

In addition, PX4 can be interfaced to run on a computer modeled vehicle in a simulated flight world. Currently for fixed wing aircraft, PX4 supports the Gazebo simulator for SITL and HITL and X-Plane for HITL only. When doing SITL simulation, PX4 communicates with offboard APIs, the ground control station and the simulator over the MAVLink protocol on UDP. Faster than real-time and lockstep simulation is also supported, as well as joystick integration, sensor failure, and camera simulation.⁴⁸ When uavAP communicates with uavEE and vice versa in SITL, HITL and real flight, point to point serial communication with CRC is used. Faster than real-time simulation playback is also supported in the uavEE environment via ROS-bags (saved flight data) and accelerated X-Plane simulation speed.

VII. Conclusion and Future Work

This work presented uavAP, a modular autopilot for UAVs, providing some details of its control stack implementation as well as applications. uavAP has been used in past research for the design of an accurate UAV power model, a flight maneuver automation framework, and an accurate kinematic model and algorithm for fixed-wing aircraft geo-fencing. Its core, cpsCore, is the C++ object-oriented backbone, used for module management such as configuration, aggregation, and synchronization. In essence, uavAP is a collection of modules merged together using cpsCore to form a flexible and distributed autopilot framework for UAVs.

In future work, uavAP will be applied to a broad range of research directions. The critical computation path of flight control provides a challenge for real-time system management, especially when parallelizing it with data-intensive vision computation. Providing real-time guarantees requires complex software isolation techniques, which can be implemented and tested in uavAP. Further work can be conducted with planning and control algorithms, branching out into trajectory optimization or power-optimal flight using techniques of artificial intelligence. While expanding the system to intelligent, unpredictable algorithms, a pairing of those algorithms with reliable, less complex algorithms might be essential. This architectural challenge can easily be addressed with uavAP, in which the intelligent algorithm can even be isolated as its own process. Another branch of research with the need for modularity and flexibility is the area of multi-agent systems, specifically multi-agent UAVs. In this field, uavAP can be used to facilitate the testing and development of various communication schemes, consensus algorithms, or even multi-agent reinforcement learning. As it is an open-source project, uavAP aims to expand into more research communities, with the goal to be a testbed of state-of-the-art research.

Acknowledgments

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant number CNS-1646383. Marco Caccamo was also supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF.

References

- ¹Mirco Theile, "uavAP: A Modular Autopilot for Unmanned Aerial Vehicles," <https://github.com/theilem/uavAP>, Accessed May 2020.
- ²Al Volo LLC, "Al Volo: Flight Data Acquisition Systems," <http://www.alvolo.us>.
- ³Mirco Theile, "uavEE: A Modular Emulation Environment for Rapid Development and Testing of Unmanned Aerial Vehicles," <https://github.com/theilem/uavEE>, Accessed May 2020.
- ⁴Theile, M., Dantsker, O. D., Nai, R., and Caccamo, M., "uavEE: A modular, power-aware emulation environment for rapid prototyping and testing of uavs," *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, IEEE, 2018, pp. 217–224.

- ⁵Yu, S., *Flight Maneuver Automation for System Analysis of Small Fixed-Wing UAVs*, Bachelor's thesis, University of Illinois at Urbana-Champaign, Department of Electrical and Computer Engineering, Urbana, IL, 2019.
- ⁶Dantsker, O. D., Yu, S., Vahora, M., and Caccamo, M., "Flight Testing Automation to Parameterize Unmanned Aircraft Dynamics," AIAA Paper 2019-3230, AIAA Aviation and Aeronautics Forum and Exposition, Dallas, Texas, June 2019.
- ⁷Theile, M., Yu, S., Dantsker, O. D., and Caccamo, M., "Trajectory Estimation for Geo-Fencing Applications on Small-Size Fixed-Wing UAVs," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 1971–1977.
- ⁸Twigg, C., "Catmull-rom splines," *Computer*, Vol. 41, No. 6, 2003, pp. 4–6.
- ⁹Mirco Theile, "Modular C++ Framework for Cyber-Physical Systems," <https://github.com/theilem/cpsCore>, Accessed May 2020.
- ¹⁰Redislabs, "Redis," <https://redis.io/>, Accessed May 2020.
- ¹¹Dantsker, O. D., Ananda, G. K., and Selig, M. S., "GA-USTAR Phase 1: Development and Flight Testing of the Baseline Upset and Stall Research Aircraft," AIAA Paper 2017-4078, AIAA Applied Aerodynamics Conference, Denver, Colorado, June 2017.
- ¹²Regan, C. D. and Taylor, B. R., "mAEWing1: Design, Build, Test - Invited," AIAA Paper 2016-1747, AIAA Atmospheric Flight Mechanics Conference, San Diego, California, Jun. 2016.
- ¹³Bunge, R. A., Alkurdi, A. E., Alfaris, E., and Kroo, I. M., "In-Flight Measurement of Wing Surface Pressures on a Small-Scale UAV During Stall/Spin Maneuvers," AIAA Paper 2016-3652, AIAA Flight Testing Conference, Washington, D.C., Jun. 2016.
- ¹⁴Bunge, R. A., Savino, F. M., and Kroo, I. M., "Approaches to Automatic Stall/Spin Detection Based on Small-Scale UAV Flight Testing," AIAA Paper 2015-2235, AIAA Atmospheric Flight Mechanics Conference, Dallas, Texas, Jun. 2015.
- ¹⁵Ragheb, A. M., Dantsker, O. D., and Selig, M. S., "Stall/Spin Flight Testing with a Subscale Aerobatic Aircraft," AIAA Paper 2013-2806, AIAA Applied Aerodynamics Conference, San Diego, CA, June 2013.
- ¹⁶M. Theile and S. Yu, "Kinematic Model for Fixed-Wing Aircraft with Constrained Roll-Rate," Tech. rep., University of Illinois at Urbana-Champaign, Department of Computer Science, Sep. 2018.
- ¹⁷Mancuso, R., Dantsker, O. D., Caccamo, M., and Selig, M. S., "A low-power architecture for high frequency sensor acquisition in many-DOF UAVs," *2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCP)*, IEEE, 2014, pp. 103–114.
- ¹⁸Dantsker, O. D., Loius, A. V., Mancuso, R., Caccamo, M., and Selig, M. S., "SDAC-UAS: A Sensor Data Acquisition Unmanned Aerial System for Flight Control and Aerodynamic Data Collection," *AIAA Infotech@Aerospace Conference, Kissimmee, Florida, Jan 2015*.
- ¹⁹Dantsker, O. D., Theile, M., and Caccamo, M., "A High-Fidelity, Low-Order Propulsion Power Model for Fixed-Wing Electric Unmanned Aircraft," AIAA Paper 2018-5009, AIAA/IEEE Electric Aircraft Technologies Symposium, Cincinnati, OH, July 2018.
- ²⁰Dantsker, O. D., Imtiaz, S., and Caccamo, M., "Electric Propulsion System Optimization for a Long-Endurance and Solar-Powered Unmanned Aircraft," AIAA Paper 2019-4486, AIAA/IEEE Electric Aircraft Technology Symposium, Indianapolis, Indiana, Aug. 2019.
- ²¹Dantsker, O. D., Caccamo, M., Theile, M., and Mancuso, R., "Flight & Ground Testing Data Set for an Unmanned Aircraft: Great Planes Avistar Elite," AIAA Paper 2020-0780, AIAA SciTech Forum, Orlando, Florida, Jan 2020.
- ²²Dantsker, O. D., Mancuso, R., Selig, M. S., and Caccamo, M., "High-Frequency Sensor Data Acquisition System (SDAC) for Flight Control and Aerodynamic Data Collection," *32nd AIAA Applied Aerodynamics Conference*, 2014, p. 2565.
- ²³Lee, J. S. and Yu, K. H., "Optimal Path Planning of Solar-Powered UAV Using Gravitational Potential Energy," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 53, No. 3, June 2017, pp. 1442–1451.
- ²⁴Grano-Romero, C., García-Juárez, M., Guerrero-Castellanos, J. F., Guerrero-Sánchez, W. F., Ambrosio-Lázaro, R. C., and Mino-Aguilar, G., "Modeling and control of a fixed-wing UAV powered by solar energy: An electric array reconfiguration approach," *2016 13th International Conference on Power Electronics (CIEP)*, June 2016, pp. 52–57.
- ²⁵Hosseini, S., Dai, R., and Mesbahi, M., "Optimal path planning and power allocation for a long endurance solar-powered UAV," *2013 American Control Conference*, June 2013, pp. 2588–2593.
- ²⁶Karabetsky, D., "Solar rechargeable airplane: Power system optimization," *2016 4th International Conference on Methods and Systems of Navigation and Motion Control (MSNMC)*, Oct 2016, pp. 218–220.
- ²⁷Lindahl, P., Moog, E., and Shaw, S. R., "Simulation, Design, and Validation of an UAV SOFC Propulsion System," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 48, No. 3, JULY 2012, pp. 2582–2593.
- ²⁸Bradt, J. B. and Selig, M. S., "Propeller Performance Data at Low Reynolds Numbers," *AIAA Paper 2011-1255, AIAA Aerospace Sciences Meeting, Orlando, Florida, Jan. 2011*.
- ²⁹Canin, D. G., McConnell, J. K., and James, P. W., "F-35 High Angle of Attack Flight Control Development and Flight Test Results," AIAA Paper 2019-3227, AIAA Aviation and Aeronautics Forum and Exposition, Dallas, Texas, June 2019.
- ³⁰Morelli, E., "Flight Test Validation of Optimal Input Design and Comparison to Conventional Inputs," AIAA Paper 1997-3711, AIAA Atmospheric Flight Mechanics Conference, New Orleans, Louisiana, Aug. 1997.
- ³¹Sobron, A., *On Subscale Flight Testing: Applications in Aircraft Conceptual Design*, Ph.D. thesis, Linköping University, Department of Management and Engineering, Linköping, Sweden, 2018.
- ³²Grauer, J. A. and Boucher, M., "Aircraft System Identification from Multisine Inputs and Frequency Responses," AIAA Paper 2020-0287, AIAA SciTech Forum, Orlando, Florida, Jan. 2020.
- ³³Kimberlin, R. D., *Flight Testing of Fixed-Wing Aircraft*, AIAA Education Series, AIAA, Reston, VA, 2003.
- ³⁴O. D. Dantsker and R. Mancuso and M. Vahora and M. Caccamo, "Unmanned Aerial Vehicle Database," <http://www.uavdb.org>.
- ³⁵Dubins, L. E., "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents," *American Journal of Mathematics*, Vol. 79, No. 3, 1957, pp. 497–516.
- ³⁶Lugo-Cárdenas, I., Flores, G., Salazar, S., and Lozano, R., "Dubins path generation for a fixed wing UAV," *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, May 2014, pp. 339–346.
- ³⁷Gurriet, T. and Ciarletta, L., "Towards a generic and modular geofencing strategy for civilian UAVs," *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2016, pp. 540–549.
- ³⁸Dill, E. T., Young, S. D., and Hayhurst, K. J., "SAFEGUARD: An assured safety net technology for UAS," *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, Sept 2016, pp. 1–10.

³⁹Dantsker, O. D., Theile, M., Caccamo, M., and Mancuso, R., “Design, Development, and Initial Testing of a Computationally-Intensive, Long-Endurance Solar-Powered Unmanned Aircraft,” AIAA Paper 2018-4217, AIAA Applied Aerodynamics Conference, Atlanta, Georgia, Jun. 2018.

⁴⁰Dantsker, O. D., Theile, M., Caccamo, M., Yu, S., Vahora, M., and Mancuso, R., “Continued Development and Flight Testing of a Long-Endurance Solar-Powered Unmanned Aircraft: UIUC-TUM Solar Flyer,” AIAA Paper 2020-0781, AIAA Scitech 2020 Forum, Orlando, Florida, Jan 2020.

⁴¹“ArduPilot Autopilot suite,” <http://ardupilot.org>, 2019.

⁴²Ebeid, E., Skriver, M., and Jin, J., “A survey on open-source flight control platforms of unmanned aerial vehicle,” *2017 Euromicro Conference on Digital System Design (DSD)*, IEEE, 2017, pp. 396–402.

⁴³Ebeid, E., Skriver, M., Terkildsen, K. H., Jensen, K., and Schultz, U. P., “A survey of open-source UAV flight controllers and flight simulators,” *Microprocessors and Microsystems*, Vol. 61, 2018, pp. 11–20.

⁴⁴“pixhawk — The Hardware Standard for Open Source Autopilots,” <https://pixhawk.org/>.

⁴⁵“Matek Systems,” <http://www.mateksys.com/>.

⁴⁶“Parrot Bepop 2 FPV Drone,” <https://www.parrot.com/global/drones/parrot-bebop-2-fpv>.

⁴⁷<http://wiki.paparazziuav.org/wiki/Overview>, 2018.

⁴⁸“PX4 Documentation,” <https://docs.px4.io/>, 2020.

Chapter 4

Reinforcement Learning for Map-based Path Planning

4.1 UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning

Reference

M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, “UAV coverage path planning under varying power constraints using deep reinforcement learning,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1444–1449, IEEE, 2020

DOI: <https://doi.org/10.1109/IROS45743.2020.9340934>

Abstract

Coverage path planning (CPP) is the task of designing a trajectory that enables a mobile agent to travel over every point of an area of interest. We propose a new method to control an unmanned aerial vehicle (UAV) carrying a camera on a CPP mission with random start positions and multiple options for landing positions in an environment containing no-fly zones. While numerous approaches have been proposed to solve similar CPP problems, we leverage end-to-end reinforcement learning (RL) to learn a control policy that generalizes over varying power constraints for the UAV. Despite recent improvements in battery technology, the maximum flying range of small UAVs is still a severe constraint, which is exacerbated by variations in the UAV’s power consumption that are hard to predict. By using map-like input channels to feed spatial information through convolutional network layers to the agent, we are able to train a double deep Q-network (DDQN) to make control decisions for the UAV, balancing limited power budget and coverage goal. The proposed method can be applied to a wide variety of environments and harmonizes complex goal structures with system constraints.

Contributions to this paper

- Shared conceptualization of the map-based methodology for coverage path planning
- Creation of the CPP code base in uavSim
- Conducting of training and evaluation simulations
- Majority of paper writing

Copyright

© 2020 IEEE. Reprinted, with permission, from Mirco Theile, Harald Bayerlein, Richard Nai, David Gesbert, and Marco Caccamo, “UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning”, 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), October 2020.

See Appendix A.3 for the reuse statement. The following shows the accepted version.

UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning

Mirco Theile¹, Harald Bayerlein², Richard Nai¹, David Gesbert², and Marco Caccamo¹

Abstract—Coverage path planning (CPP) is the task of designing a trajectory that enables a mobile agent to travel over every point of an area of interest. We propose a new method to control an unmanned aerial vehicle (UAV) carrying a camera on a CPP mission with random start positions and multiple options for landing positions in an environment containing no-fly zones. While numerous approaches have been proposed to solve similar CPP problems, we leverage end-to-end reinforcement learning (RL) to learn a control policy that generalizes over varying power constraints for the UAV. Despite recent improvements in battery technology, the maximum flying range of small UAVs is still a severe constraint, which is exacerbated by variations in the UAV’s power consumption that are hard to predict. By using map-like input channels to feed spatial information through convolutional network layers to the agent, we are able to train a double deep Q-network (DDQN) to make control decisions for the UAV, balancing limited power budget and coverage goal. The proposed method can be applied to a wide variety of environments and harmonizes complex goal structures with system constraints.

I. INTRODUCTION

Whereas the CPP problem for ground-based robotics has already found its way into our everyday life in the form of vacuum cleaning robots [1], autonomous coverage with UAVs, while not yet having attained the same level of prominence, is being considered for a wide range of applications, such as photogrammetry, smart farming and especially disaster management [2]. UAVs can be deployed rapidly to gather initial or continuous survey data of areas hit by natural disasters, or mitigate their consequences. In the aftermath of the 2019-20 Australian bushfire season, wildlife officers inventively used quadcopter drones with infrared sensors to conduct a search-and-rescue operation for koalas affected by the blaze [3].

As its name suggests, covering all points inside an area of interest with CPP is related to conventional path planning where the goal is to find a path between start and goal positions. In general, CPP aims to cover as much of the target area as possible within given energy or path-length constraints while avoiding obstacles or no-fly zones. Due to the limitations in battery energy density, available power limits mission duration for quadcopter UAVs severely. Finding a CPP control policy that generalizes over varying power constraints and setting a specific movement budget can be

seen as a way to model the variations in actual power consumption during the mission, e.g. caused by environmental factors that are hard to predict. Similar to conventional path planning, CPP can usually be reduced to some form of the travelling salesman problem, which is NP-hard [1]. Lawnmowing and milling [4] are other examples of closely related problems.

The most recent survey of UAV coverage path planning is given by Cabreira *et al.* [2]. Galceran and Carreras [1] provide a survey of general (ground robotics) approaches to CPP. Autonomous UAVs for applications in wireless communications have also sparked a lot of interest recently. Some scenarios, e.g. deep RL trajectory planning for UAVs providing wireless connectivity under battery power constraints, are related to CPP. An overview of UAV applications in wireless communications can be found in [5].

To guarantee complete coverage, most existing CPP approaches split the target area and surrounding free space into cells, by means of exact or approximate cellular decomposition. Choset and Pignon [6] proposed the classical boustrophedon (“the way of the ox”, back and forth motion) cellular decomposition, an exact decomposition method that guarantees full coverage but offers no bounds on path-length. This algorithm was extended by Mannadiar and Rekleitis [7] through encoding the cells of the boustrophedon decomposition as a Reeb graph and then constructing the Euler tour that covers every edge in the graph exactly once. Cases where the mobile agent does not have enough power to cover the whole area are not considered. The authors in [8] adapted this method for use in a non-holonomic, fixed-wing UAV and conducted extensive experimental validation. Two other approaches combining CPP and the travelling salesman problem to find near-optimal solutions for coverage of target regions enclosed by non-target areas are proposed by the authors in [9]: grid-based and dynamic programming-based, respectively. Both approaches suffer from exponential increase in time complexity with the number of target regions and do not consider obstacles or UAV power limitations.

Non-standard approaches have made use of neural networks (NNs) before. The authors in [10] design a network of neurons with only lateral connections that each represent one grid cell in a cleaning robot’s non-stationary 2D environment. The path planning is directly based on the neural network’s activity landscape, which is computationally simple and can support changing environments, but does not take path-length or power constraints into account.

Reinforcement learning with deep neural networks has only recently started to be considered for UAV path planning.

¹Mirco Theile, Richard Nai, and Marco Caccamo are with the TUM Department of Mechanical Engineering, Technical University of Munich, Germany {mirco.theile, richard.nai, mcaccamo}@tum.de

²Harald Bayerlein and David Gesbert are with the Communication Systems Department, EURECOM, Sophia Antipolis, France {harald.bayerlein, david.gesbert}@eurecom.fr

Maciel-Pearson *et al.* [11] proposed a method using an extended double deep Q-network (EDDQN) to explore and navigate from a start to a goal position in outdoor environments by combining map and camera information from the drone. Their approach is focused on obstacle avoidance under changing weather conditions. The authors in [12] investigate the CPP-related drone patrolling problem where a UAV patrols an area optimizing the relevance of its observations through the use of a single-channel relevance map fed into a convolutional layer of a DDQN agent. However, there is no consideration for power constraints and the relevance map is preprocessed showing only local information. To the best of our knowledge, deep RL has not been considered for UAV control in coverage path planning under power constraints before.

The main contributions of this paper are the following:

- Introduction of a novel UAV control method for coverage path planning based on double deep Q-learning;
- The usage of map-like channels to feed spatial information into convolutional network layers of the agent;
- Learning a control policy that generalizes over random start positions and varying power constraints and decides between multiple landing positions.

The remainder of this paper is organized as follows: Section II introduces the CPP problem formulation, Section III describes our DDQN learning approach and in Section IV follow simulation results and their discussion. We conclude the paper with a summary and outlook onto future work in Section V.

II. PROBLEM FORMULATION

A. Setup

The sensors of the UAV forming the input of the reinforcement learning agent are depicted in Figure 1: camera and GPS receiver. The camera gives a periodic frame of the current coverage view and the GPS yields the drone's position. Power constraints determined by external factors are modelled as a movement budget for the drone that is fixed at mission start. Two additional software components are running on the UAV. The first is the mission algorithm which is responsible for the analysis of the camera data. We assume that any mission algorithm can give feedback on the area that was already covered. The second component is a safety controller that evaluates the proposed action of the agent and accepts or rejects it based on the safety constraints (entering into no-fly zones or landing in unsuitable areas). Note that the safety controller does not assist the agent in finding the landing area. The last component is a map which is provided by the operator on the ground. While this map could be dynamic throughout the mission, we focus on static maps for the duration of one mission in this paper.

B. 3-Channel Map

The coverage problem to be solved can be represented by a two dimensional grid map with three channels. Each cell in the grid represents a square area of the coverage region. The three channels describe starting and landing zones, target

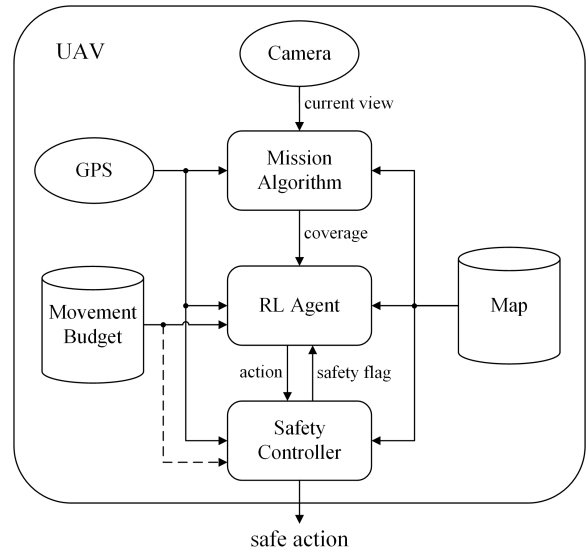


Fig. 1: System-level diagram depicting sensor and software components on the UAV during a coverage mission.

zones, and no-fly zones. The start and landing zones are areas the agent can start from and land on after finishing a coverage path. Target zones have to be covered at least once by the field of view (FoV) of the UAV's camera. No-fly zones represent areas which the drone is prohibited from entering. Note that it is possible that a cell is declared as none, or more than one of these zones, with the exception that starting and landing zones can not be no-fly zones at the same time.

C. Markov Decision Process

In order to solve the described coverage path planning problem with reinforcement learning, it is converted into a Markov decision process (MDP). An MDP is described by the tuple $(\mathcal{S}, \mathcal{A}, R, P)$, with the set of possible states \mathcal{S} , the set of possible actions \mathcal{A} , the reward function R , and the deterministic state transition function $P: \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$.

In a $N \times N$ grid, the state space \mathcal{S} has the following dimensions:

$$\mathcal{S} = \underbrace{\mathbb{B}^{N \times N \times 3}}_{\text{Map}} \times \underbrace{\mathbb{B}^{N \times N}}_{\text{Coverage}} \times \underbrace{\mathbb{R}^2}_{\text{Position}} \times \underbrace{\mathbb{N}}_{\text{Movement Budget}} \times \underbrace{\mathbb{B}}_{\text{Safety Flag}}$$

where \mathbb{B} is the Boolean domain $\{0, 1\}$. The action space \mathcal{A} contains the following five actions:

$$\mathcal{A} = \{\text{north, east, south, west, land}\}$$

The reward function $R: \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, mapping the current state $s \in \mathcal{S}$ and current action $a \in \mathcal{A}$ to a real-valued reward, consists of multiple components:

- r_{cov} (*positive*) coverage reward for each target cell that is covered by the UAV's field of view for the first time;
- r_{sc} (*negative*) safety penalty in case the safety controller (SC) rejects the agent's proposed action;

- r_{mov} (negative) constant movement penalty that is applied for every unit of the movement budget the UAV uses
- r_{crash} (negative) penalty in case the UAV runs out of movement budget without having safely landed in a landing zone.

III. METHODOLOGY

A. Q-Learning

Reinforcement learning, in general, proceeds in a cycle of interactions between an agent and its environment. At time t , the agent observes a state $s_t \in \mathcal{S}$, performs an action $a_t \in \mathcal{A}$ and subsequently receives a reward $r(s_t, a_t) \in \mathbb{R}$. The time index is then incremented and the environment propagates the agent to a new state s_{t+1} , from where the cycle restarts. The goal of the agent is to maximize the discounted cumulative return R_t from the current state up to a terminal state at time T . It is given as

$$R_t = \sum_{k=t}^T \gamma^{k-t} r(s_k, a_k). \quad (1)$$

with $\gamma \in [0, 1]$ being the discount factor, balancing the importance of immediate and future rewards. The return is maximized by adapting the agent's behavioral policy π . The policy can be deterministic with $\pi(s)$ such that $\pi: \mathcal{S} \mapsto \mathcal{A}$, or probabilistic with $\pi(a|s)$ such that $\pi: \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, yielding a probability distribution over the action space for each $s \in \mathcal{S}$.

To find a policy which maximizes the return, we utilize Q-learning, a model-free reinforcement learning approach. It is based on learning the state-action-value function, or Q-function $Q: \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, defined as

$$Q^\pi(s, a) = \mathbb{E}_\pi [R_t | s_t = s, a_t = a]. \quad (2)$$

Q-learning relies on iteratively updating the current knowledge of the Q-function. When the optimal Q-function is known, it is easy to construct an optimal policy by taking actions that maximize the Q-function. For convenience, s_t and a_t are abbreviated to s and a and s_{t+1} and a_{t+1} to s' and a' in the following.

B. Deep Q-Learning

The Q-function from (2) can be represented through a table of Q-values with the dimension $\mathcal{S} \times \mathcal{A}$. This is not feasible for large state or action spaces, but it is possible to approximate the Q-function by a neural network in those cases. A deep Q-network (DQN) parameterizing the Q-function with the parameter vector θ is trained to minimize the expected temporal difference (TD) error given by

$$L(\theta) = \mathbb{E}_\pi [(Q_\theta(s, a) - Y(s, a, s'))^2] \quad (3)$$

with the target value

$$Y(s, a, s') = r(s, a) + \gamma \max_{a'} Q_\theta(s', a'). \quad (4)$$

While a DQN is significantly more data efficient compared to a Q-table due to its ability to generalize, the

deadly triad [13] of function approximation, bootstrapping and off-policy training can make its training unstable and cause divergence. In 2015, Mnih *et al.* [14] presented a methodology to stabilize the DQN learning process. Their training approach makes use of an experience replay memory \mathcal{D} which stores experience tuples (s, a, r, s') collected by the agent during each interaction with the environment. Training the agent on uniformly sampled batches from the replay memory decorrelates the individual samples and rephrases the TD-error as

$$L^{\text{DQN}}(\theta) = \mathbb{E}_{s, a, s' \sim \mathcal{D}} [(Q_\theta(s, a) - Y^{\text{DQN}}(s, a, s'))^2]. \quad (5)$$

Additionally, Mnih *et al.* used a separate target network for the estimation of the next maximum Q-value changing the target value to

$$Y^{\text{DQN}}(s, a, s') = r(s, a) + \gamma \max_{a'} Q_{\bar{\theta}}(s', a') \quad (6)$$

with $\bar{\theta}$ representing the parameters of the target network. The parameters of the target network $\bar{\theta}$ can either be updated as a periodic hard copy of θ or as a soft update with

$$\bar{\theta} \leftarrow (1 - \tau)\bar{\theta} + \tau\theta \quad (7)$$

after each update of θ . $\tau \in [0, 1]$ is the update factor determining the adaptation pace. The combination of replay memory and target network separation to stabilize the training process laid the groundwork for the rise in popularity of DQN methods.

An additional improvement was proposed by Van Hasselt *et al.* [15], who showed that under certain conditions, action values in (6) get overestimated. To solve this issue, the double deep Q-network (DDQN) was introduced. The target value is then given by

$$Y^{\text{DDQN}}(s, a, s') = r(s, a) + \gamma Q_{\bar{\theta}}(s', \arg\max_{a'} Q_\theta(s', a')) \quad (8)$$

and the corresponding loss function

$$L^{\text{DDQN}}(\theta) = \mathbb{E}_{s, a, s' \sim \mathcal{D}} [(Q_\theta(s, a) - Y^{\text{DDQN}}(s, a, s'))^2], \quad (9)$$

in which the overestimation of action values is reduced by selecting the best action using θ but estimating the value of that action using $\bar{\theta}$. When calculating $\nabla_\theta L^{\text{DDQN}}(\theta)$ the target value is taken as is, hence, the back-propagating gradient is stopped before $Y^{\text{DDQN}}(s, a, s')$.

C. Neural Network Model and Training Procedure

The DQN solving the MDP from Section II consists of convolutional and fully-connected layers. It is visualized in Figure 2. The UAV's own position is converted to a 2D one-hot representation, i.e. the encoding of the occupied cell inside the whole grid. With the position encoded in this way, it can be stacked with the three-channel map and the coverage grid to form the five-channel input of the network's convolutional layers. The kernels of the convolutional layers are then able to form direct spatial connections between the current position and nearby cells. The remaining movement budget is fed into the network after the convolutional layers.

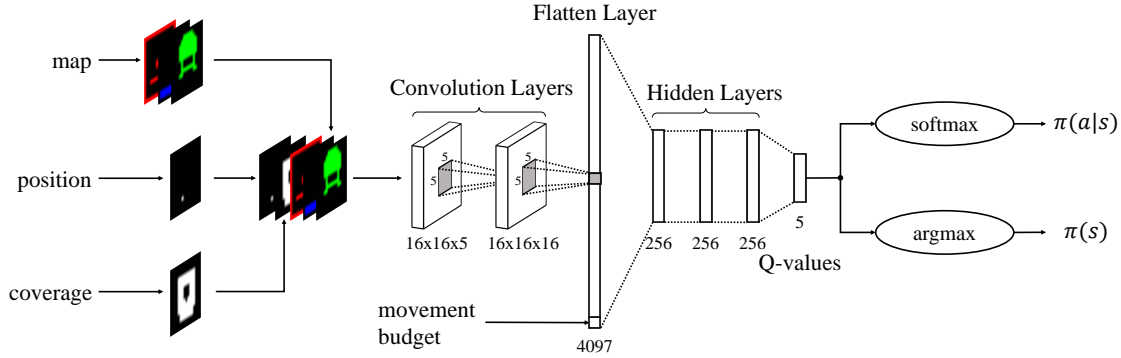


Fig. 2: Neural network structure for the reinforcement learning agent.

The convolutional layers are padded so that their output shape remains the same as their input shape. All layers are zero-padded for all channels, with the exception of the first layer's no-fly zone channel, which is one-padded. This is an explicit representation of the no-fly zone surrounding the mission grid. The rectified linear unit (ReLU) is chosen as activation function for the convolutional layers. The last layer of the convolutional network is flattened and concatenated with the movement budget input. Fully-connected layers with ReLU activation are attached to this flatten layer.

The last fully-connected layer is of size $|\mathcal{A}|$ and has no activation function. It directly represents the Q-values for each action given the input state. Choosing the argmax of the Q-values is called the greedy policy and exploits already learned knowledge. The greedy policy given by

$$\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q_{\theta}(s, a) \quad (10)$$

is deterministic and used when evaluating the agent's learning progress. During training, the sampled soft-max policy for exploration of the state and action space is used instead. It is given by

$$\pi(a_i|s) = \frac{e^{Q_{\theta}(s, a_i)/\beta}}{\sum_{\forall a_j \in \mathcal{A}} e^{Q_{\theta}(s, a_j)/\beta}} \quad (11)$$

with the temperature parameter $\beta \in \mathbb{R}$ scaling the balance of exploration versus exploitation. When β is increased so does exploration. The limit $\beta \rightarrow 0$ of the soft-max policy (11) is the greedy policy (10). The soft-max policy was chosen over the ϵ -greedy policy because it offers variable exploration based on the relative difference of Q-values and does not depend on the number of training steps or episodes. This appeared to be beneficial for this particular problem.

Algorithm 1 describes the training procedure for the double deep Q-network in more detail. After replay memory and network parameters are initialized, a new training episode begins with resetting the state, choosing a random UAV starting position and random movement budget $b_0 \in \mathcal{B}$. The episode continues as long as the movement budget is greater than zero and the UAV has not landed. A new action $a \in \mathcal{A}$

Algorithm 1 DDQN training for coverage path planning

Initialize \mathcal{D} , initialize θ randomly, $\bar{\theta} \leftarrow \theta$

- 1: **for** $n = 0$ to N_{max} **do**
- 2: Initialize state s_0 with random starting position and sample initial movement budget b_0 uniformly from \mathcal{B}
- 3: **while** $b > 0$ and not landed **do**
- 4: Sample a according to (11)
- 5: Observe r, s'
- 6: Store (s, a, r, s') in \mathcal{D}
- 7: **for** $i = 1$ to m **do**
- 8: Sample (s_i, a_i, r_i, s'_i) uniformly from \mathcal{D}
- 9: $Y_i = \begin{cases} r_i, & \text{if } s'_i \text{ terminal} \\ \text{according to (8),} & \text{otherwise} \end{cases}$
- 10: Compute loss $L_i(\theta)$ according to (9)
- 11: **end for**
- 12: Update θ with gradient loss $\frac{1}{m} \sum_{i=1}^m L_i(\theta)$
- 13: Soft update of $\bar{\theta}$ according to (7)
- 14: $b = b - 1$
- 15: **end while**
- 16: **end for**

is chosen according to (11) and the subsequent experience stored in the replay memory buffer \mathcal{D} .

Sampling a minibatch of size m from the replay memory, the primary network parameters θ are updated by performing a gradient step using the Adam optimizer. Subsequently, the target network parameters $\bar{\theta}$ are updated using the soft update (7) and the movement budget is decremented. The episode ends when either the drone lands or the movement budget decreases to zero. Then, a new episode starts unless the maximum number of episodes N_{max} is reached. The hyperparameters that were used during training are listed in Table I.

IV. EXPERIMENTS

A. Simulation Setup

The agent can move in a two dimensional grid through action commands in \mathcal{A} if accepted by the safety controller.

Parameter	Value	Description
$ \theta $	1,190,389	number of trainable parameters
$ \mathcal{D} $	50,000	replay memory buffer size
N_{max}	10,000	maximum number of training episodes
β	0.1	temperature parameter (11)
m	128	minibatch size
γ	0.95	discount factor for target value in (8)
τ	0.005	target network update factor (7)

TABLE I: Hyperparameters for DDQN training.

Each action, no matter if accepted or rejected, consumes one unit of movement budget since energy is spent during hovering as well. The agent’s initial state $s_0 \in \mathcal{S}$ consists of a fixed map, a zero-initialized coverage grid and a position, which is uniformly sampled from the starting and landing zone of the map. Additionally, the initial movement budget is uniformly sampled from a movement budget range \mathcal{B} , which is set to 25-75 for the purpose of this evaluation. The value of the safety flag in s_0 is initialized to zero and the UAV’s camera field of view (FoV) is set to a fixed 3-by-3-cell area centered underneath the agent. After each step the mission algorithm marks the FoV as seen in the coverage grid map.

Three evaluation scenarios were chosen, each with a unique problem for the agent to solve. Map A depicted in Figure 3 (a)-(c) has a large starting and landing zone, which yields high variation during training. Additionally, the shape of the target area is challenging to cover. The difficulty of map B in Figure 3 (d)-(f) lies in the yellow area that is marked as target zone, but also marked as a no-fly zone, and therefore must be covered by flying adjacent to it. Map C, in Figure 3 (g)-(i) with a narrow passage between no-fly zones, while easy to cover is very difficult for training as discussed later.

B. Evaluation

After being trained on their respective scenario with varying movement budgets and varying starting positions under the exploration policy $\pi(a|s)$ from (11), the agents are evaluated under their exploitation policy $\pi(s)$ from (10). Their performance during coverage for the full movement budget range and all possible starting positions is evaluated. The performance is described through Figures 3 and 4 and Table II.

The agents’ ability to plan a trajectory that ends with a safe landing inside the landing zone over the full movement budget range and starting at each possible position is evaluated through Table II, showing the ratio of landing for all scenario variations. Despite the agent’s good landing performance, the safety controller’s capabilities on a real-world UAV would likely be extended to force navigation to the closest landing zone in the rare cases when the RL agent misses the right moment to return.

To evaluate the impact of movement budget on the achieved coverage ratio, the starting position was fixed. For the selected starting positions the agents successfully landed after completing a trajectory for each movement budget. Figure 4 shows the coverage ratio of each agent with respect

to initial movement budget. Selected trajectories for each map under three different movement budgets are depicted in Figure 3. Whereas the movement budget is increasing from left to right, the agent does not necessarily utilize the whole allocated budget if it determines that there is a risk of not returning to the landing area in time or the coverage goal is already fulfilled. It can be seen that the agent finds a trajectory balancing the goals of safe landing and maximum coverage ratio.

Figure 5 shows the training process of an agent on map C. The curve describes the cumulative reward of the exploitation strategy when evaluated during the training process. Three major phases appear during the training process, which are highlighted in the graph. In phase one the agent learns to land safely, but does not venture far enough from the landing zone to find the target zone. When transitioning to phase two, the agent discovers the target zone, yielding high immediate reward. Due to the focus on mid-term reward through the choice of discount factor γ , this strategy represents a local optimum. In phase three the agent discovers the path back to the landing zone, avoiding the crashing penalty r_{crash} . After refining the trajectory, the agent finds the optimal path at the end of phase three. The phase transitions are highly dependent on the exploration strategy. Soft-max exploration appeared to be more effective than the ϵ -greedy policy to guide these transitions. The basic pattern of this incremental learning process is also visible when applied to other maps, albeit with less pronounced transitions due to bigger variations in coverage ratios.

V. CONCLUSION

We have introduced a new deep reinforcement learning approach for the longstanding problem of coverage path planning. While existing approaches might offer guarantees on the (near)-optimality of their solutions, the case where available power constrains the path planning is usually not considered. By feeding spatial information through map-like input channels to the agent, we train a double deep Q-network to learn a UAV control policy that generalizes over varying starting positions and varying power constraints modelled as movement budgets. Using this method, we observed an incremental learning process that successfully balances safe landing and coverage of the target area on three different environments, each with unique challenges.

In the future we will investigate the possibilities of transfer learning for this set of problems. At first we will train the agents on a curriculum of problems based on individual map channels to further accelerate the training process described in this work. From there we will examine approaches to transfer the two dimensional grid agent to higher dimensions and dynamics, e.g. adding altitude and heading. To this effect, it might be beneficial to investigate other RL techniques such as actor-critic methods and policy optimization. The proposed approach can also be seen as an initial step for handling variable power consumption in a real-world scenario. Through these steps an application on physical hardware will likely be within reach.

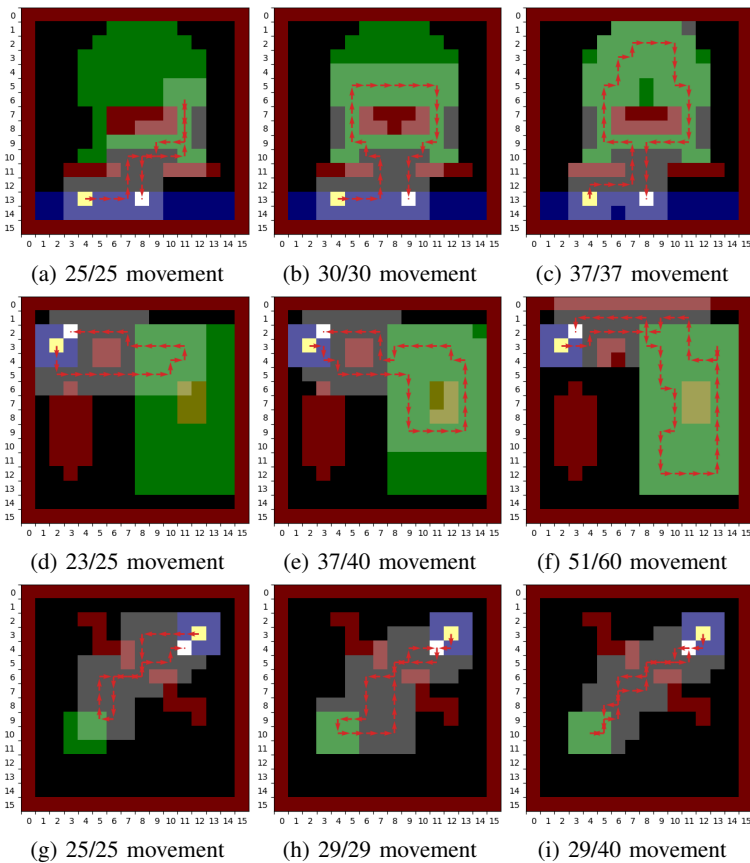


Fig. 3: Coverage plots for three different maps (map A: (a)-(c), map B: (d)-(f), map C: (g)-(i)) with three different movement budgets each; red, blue, and green are no-fly zones, starting/landing zones, and target zones, respectively; the red arrows describe the trajectory and the yellow and white cell describe start and landing position, respectively; lighter cells were covered by the agent's FoV.

ACKNOWLEDGMENTS

Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research. Harald Bayerlein and David Gesbert are supported by the PERFUME project funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement no. 670896).

REFERENCES

- [1] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258–1276, 2013.
- [2] T. Cabreira, L. Brisolará, and P. R. Ferreira, "Survey on coverage path planning with unmanned aerial vehicles," *Drones*, vol. 3, no. 1, 2019.
- [3] D. Gimesy, "Drones and thermal imaging: saving koalas injured in the bushfires - [news]," *The Guardian*, 10 Feb 2020.
- [4] E. M. Arkin, S. P. Fekete, and J. S. Mitchell, "Approximation algorithms for lawn mowing and milling," *Computational Geometry*, vol. 17, no. 1-2, pp. 25–50, 2000.
- [5] Y. Zeng, Q. Wu, and R. Zhang, "Accessing from the sky: A tutorial on UAV communications for 5G and beyond," *Proceedings of the IEEE*, vol. 107, no. 12, pp. 2327–2375, 2019.
- [6] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon cellular decomposition," in *Field and service robotics*, pp. 203–209, Springer, 1998.
- [7] R. Mannadiar and I. Rekleitis, "Optimal coverage of a known arbitrary environment," in *2010 IEEE International conference on robotics and automation*, pp. 5525–5530, 2010.
- [8] A. Xu, C. Viriyasuthee, and I. Rekleitis, "Optimal complete terrain coverage using an unmanned aerial vehicle," in *2011 IEEE International conference on robotics and automation*, pp. 2513–2519, 2011.
- [9] J. Xie, L. R. G. Carrillo, and L. Jin, "An integrated traveling salesman and coverage path planning problem for unmanned aircraft systems," *IEEE control systems letters*, vol. 3, no. 1, pp. 67–72, 2018.
- [10] S. X. Yang and C. Luo, "A neural network approach to complete coverage path planning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 34, no. 1, pp. 718–724, 2004.
- [11] B. G. Maciel-Pearson, L. Marchegiani, S. Akcay, A. Atapour-Abarghouei, J. Garforth, and T. P. Breckon, "Online deep reinforcement learning for autonomous UAV navigation and exploration of outdoor environments," *arXiv preprint arXiv:1912.05684*, 2019.
- [12] C. Piciarelli and G. L. Foresti, "Drone patrolling with reinforcement learning," in *Proceedings of the 13th International Conference on Distributed Smart Cameras*, ACM, 2019.
- [13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: an introduction*. MIT Press, second ed., 2018.
- [14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [15] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Thirtieth AAAI conference on artificial intelligence*, pp. 2094–2100, 2016.

	Map A	Map B	Map C
Landing ratio	99.37%	99.78%	98.26%

TABLE II: Landing ratio for each map evaluated on the full range of movement budgets and possible starting positions.

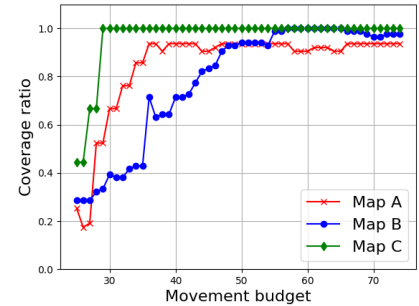


Fig. 4: Coverage ratio with varying movement budget for the three maps.

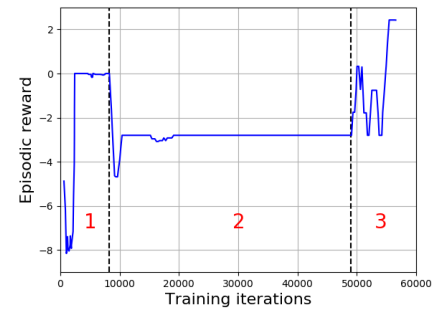


Fig. 5: Training process of an agent trained on map C with dashed lines indicating training phase transitions.

4.2 UAV Path Planning for Wireless Data Harvesting: A Deep Reinforcement Learning Approach

Reference

H. Bayerlein, M. Theile, M. Caccamo, and D. Gesbert, “UAV path planning for wireless data harvesting: A deep reinforcement learning approach,” in *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2020
DOI: <https://doi.org/10.1109/GLOBECOM42002.2020.9322234>

Abstract

Autonomous deployment of unmanned aerial vehicles (UAVs) supporting next-generation communication networks requires efficient trajectory planning methods. We propose a new end-to-end reinforcement learning (RL) approach to UAV-enabled data collection from Internet of Things (IoT) devices in an urban environment. An autonomous drone is tasked with gathering data from distributed sensor nodes subject to limited flying time and obstacle avoidance. While previous approaches, learning and non-learning based, must perform expensive recomputations or relearn a behavior when important scenario parameters such as the number of sensors, sensor positions, or maximum flying time, change, we train a double deep Q-network (DDQN) with combined experience replay to learn a UAV control policy that generalizes over changing scenario parameters. By exploiting a multi-layer map of the environment fed through convolutional network layers to the agent, we show that our proposed network architecture enables the agent to make movement decisions for a variety of scenario parameters that balance the data collection goal with flight time efficiency and safety constraints. Considerable advantages in learning efficiency from using a map centered on the UAV’s position over a non-centered map are also illustrated.

Contributions to this paper

- Shared conceptualization of the map-based planning methodology for data harvesting using map-centering
- Shared adaptation of the uavSim code for data harvesting
- Share of paper writing

Copyright

© 2020 IEEE. Reprinted, with permission, from Harald Bayerlein, Mirco Theile, Marco Caccamo, and David Gesbert, “UAV Path Planning for Wireless Data Harvesting: A Deep Reinforcement Learning Approach”, *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, December 2020.

See Appendix A.4 for the reuse statement. The following shows the accepted version.

UAV Path Planning for Wireless Data Harvesting: A Deep Reinforcement Learning Approach

Harald Bayerlein¹, Mirco Theile², Marco Caccamo², and David Gesbert¹

¹Communication Systems Department, EURECOM, Sophia Antipolis, France

²TUM Department of Mechanical Engineering, Technical University of Munich, Germany

{harald.bayerlein, david.gesbert}@eurecom.fr, {mirco.theile, mcaccamo}@tum.de

Abstract—Autonomous deployment of unmanned aerial vehicles (UAVs) supporting next-generation communication networks requires efficient trajectory planning methods. We propose a new end-to-end reinforcement learning (RL) approach to UAV-enabled data collection from Internet of Things (IoT) devices in an urban environment. An autonomous drone is tasked with gathering data from distributed sensor nodes subject to limited flying time and obstacle avoidance. While previous approaches, learning and non-learning based, must perform expensive re-computations or relearn a behavior when important scenario parameters such as the number of sensors, sensor positions, or maximum flying time, change, we train a double deep Q-network (DDQN) with combined experience replay to learn a UAV control policy that generalizes over changing scenario parameters. By exploiting a multi-layer map of the environment fed through convolutional network layers to the agent, we show that our proposed network architecture enables the agent to make movement decisions for a variety of scenario parameters that balance the data collection goal with flight time efficiency and safety constraints. Considerable advantages in learning efficiency from using a map centered on the UAV’s position over a non-centered map are also illustrated.

I. INTRODUCTION

While unmanned aerial vehicles (UAVs) are envisioned for a multitude of applications, their prospective roles in telecommunications can be classified into two categories: cellular-connected UAVs attached to mobile network links or UAVs providing communication services themselves, e.g. collecting data from distributed Internet of Things (IoT) devices [1]. As an example in the context of infrastructure maintenance and preserving structural integrity, Hitachi is already commercially deploying partially autonomous UAVs that collect data from IoT sensors embedded in large infrastructure structures [2].

Collecting data from sensor devices in an urban environment imposes challenging constraints on the trajectory design for autonomous UAVs. Battery energy density restricts mission duration for quadcopter drones severely, while the complex urban environment poses challenges in obstacle avoidance and the adherence to regulatory no-fly zones (NFZs). Additionally, the wireless communication channel is characterized

by frequent fluctuations in attenuation through alternating line-of-sight (LoS) and non-line-of-sight (NLoS) links. Deep reinforcement learning (DRL) offers the opportunity to balance challenges and data collection goal for complex environments in a straightforward way by combining them in the reward function. This advantage also holds for other instances of UAV path planning, such as coverage path planning, a classical robotics problem where the UAV’s goal is to cover all points inside an area of interest. By basing our proposed method on an approach to UAV coverage path planning [3], we would like to highlight the connection between these research areas.

A recent tutorial covering the paradigms of cellular-connected UAVs as well as UAV-assisted communications, including trajectory planning for IoT data collection, is given in [1]. Bithas *et al.* [4] provide a survey on machine learning techniques, including but not limited to reinforcement learning (RL), for various UAV communications scenarios.

Most existing approaches to UAV data collection are not based on RL and only find a solution for one set of scenario parameters at a time. Esrafilian *et al.* [5] proposed a two-step algorithm to optimize a UAV’s trajectory and its scheduling decisions in an urban data collection mission using a combination of dynamic and sequential convex programming. While set in a similar environment, the scenario does not account for NFZs or obstacle avoidance as the drone is assumed to always fly above the highest building. This also holds for the hybrid offline-online optimization approach presented in [6], where a preliminary trajectory is computed before the UAV’s start based on a probabilistic LoS channel model and then optimized while the UAV is on its mission in an online fashion.

(Deep) reinforcement learning has been explored in other related UAV communication scenarios. The approach in the simple scenario of [7], where a UAV base station serves two ground users, is focused on showing the advantages of neural network (NN) over table-based Q-learning, while not making any explicit assumptions about the environment at the price of long training time. Deep deterministic policy gradient (DDPG), an actor-critic RL method, was proposed by Qi *et al.* [8] to learn a continuous control policy for a UAV providing persistent communications coverage to a group of users in an environment without obstacles. If a critical scenario parameter like the number of users changes, the agent has to undergo computationally expensive retraining.

Some works under the paradigm of mobile crowdsensing,

H. Bayerlein and D. Gesbert are supported by the PERFUME project funded by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (grant agreement no. 670896). M. Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research. The code for this work is available under https://github.com/hbayerlein/uav_data_harvesting.

where mobile devices are leveraged to collect data of common interest, have also suggested the use of UAVs for data collection. Liu *et al.* [9] proposed an RL multi-agent DDPG algorithm collecting data simultaneously with ground and aerial vehicles in an environment with obstacles and charging stations. While their approach also makes use of convolutional processing to exploit a map of the environment, they do not center the map on the agent's position, which we show to be highly beneficial. Furthermore, in contrast to our method, control policies have to be relearned entirely when scenario and environmental parameters change.

If deep RL methods are to be applied in real-world missions, the prohibitively high training data demand poses one of the most severe challenges [10]. This is exacerbated by the fact that even small changes in the scenario, such as the number of sensor devices typically require complete retraining. By taking varying parameters in the design and training of the neural network model into account, we take a step towards the mitigation of this challenge.

The main contributions of this paper are the following:

- Introducing a novel DDQN-based method to control a UAV on an IoT data harvesting mission, maximizing collected data under flying time and navigation constraints without prior information about the wireless channel characteristics;
- Showing the considerable increase in learning efficiency for the RL agent when exploiting a centered multi-layer map of the environment;
- Learning to effectively adapt to variations in environmental and scenario parameters as the first step to more realistic RL methods in the context of UAV IoT data collection.

II. SYSTEM MODEL AND PROBLEM FORMULATION

A. System Model

We consider a square grid world of size $M \times M \in \mathbb{N}^2$ with the UAV collecting data from K static IoT devices. The k -th device is located on ground level at $\mathbf{u}_k = [x_k, y_k, 0]^T \in \mathbb{R}^3$ with $k \in [1, K]$. The UAV's data collection mission is over at time $T \in \mathbb{N}$, where the time horizon is discretized into equal mission time slots $t \in [0, T]$. The UAV's position is given by $[x(t), y(t), h]^T \in \mathbb{R}^3$ with constant altitude h . Its 2D projection on the ground is given by $\mathbf{p}(t) = [x(t), y(t)]^T$. Mission time slots are chosen sufficiently small so that the UAV's velocity $v(t)$ can be considered to remain constant in one time slot. The UAV is limited to moving with constant velocity V or hovering, i.e. $v(t) \in \{0, V\}$ for all $t \in [0, T]$.

As it is expected that the communication channel is subject to faster changes than the UAV's movement, we partition each mission time slot $t \in [0, T]$ into a number of $\delta \in \mathbb{N}$ communication time slots. The communication time index is then $n \in [0, N]$ with $N = \delta T$. The number of communication time slots per mission time slot δ is chosen sufficiently large so that the UAV's position, which is interpolated linearly between $\mathbf{p}(t)$ and $\mathbf{p}(t+1)$, and the channel gain can be considered constant within one communication time slot.

Similar to the channel model in [5], the communication links between UAV and the K IoT devices are modeled as LoS/NLoS point-to-point channels with log-distance path loss and shadow fading. The information rate at time n for the k -th device is given by

$$R_k(n) = \log_2(1 + \text{SNR}_k(n)), \quad (1)$$

where the signal-to-noise ratio (SNR) with transmit power P_k , white Gaussian noise power at the receiver σ^2 , UAV-device distance $d_k(n)$, path loss exponent α_l and $\eta_l \sim \mathcal{N}(0, \sigma_l^2)$ modeled as a Gaussian random variable, is defined as

$$\text{SNR}_k(n) = \frac{P_k}{\sigma^2} \cdot d_k(n)^{-\alpha_l} \cdot 10^{\eta_l/10}. \quad (2)$$

Note that the urban environment causes a strong dependence of the propagation parameters on the $l \in \{\text{LoS}, \text{NLoS}\}$ condition and that (2) is the SNR averaged over small scale fading.

The sensor nodes are served by the UAV in a simple time-division multiple access (TDMA) manner where, in each communication time slot $n \in [0, N]$, the sensor node $k \in [1, K]$ with the highest $\text{SNR}_k(n)$ with remaining data to be uploaded is picked by the scheduling algorithm. The TDMA constraint for the scheduling variable $q_k(n) \in \{0, 1\}$ is given by

$$\sum_{k=1}^K q_k(n) \leq 1, \quad n \in [0, N]. \quad (3)$$

The achievable throughput for one mission time slot t is then the sum of the achieved rates of the corresponding communication time slots $n \in [\delta t, \delta(t+1) - 1]$ over K sensor nodes and given by

$$C(t) = \sum_{n=\delta t}^{\delta(t+1)-1} \sum_{k=1}^K q_k(n) R_k(n). \quad (4)$$

The central goal of the trajectory optimization problem is the maximization of throughput over the whole data collection mission while minimizing flight duration, subject to the constraints of maximum flight time, adherence to NFZs, obstacle avoidance, and safe landing in designated landing areas. We translate this optimization problem into a reward function as part of a Markov decision process, which we solve using deep reinforcement learning.

B. Markov Decision Process

A Markov decision process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, R, P)$ with state-space \mathcal{S} , action space \mathcal{A} and reward function R . We consider a finite horizon MDP with a probabilistic state transition function $P: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$. In line with standard MDP convention, the time index t is written in subscript in the following.

The state at mission time t in the grid world of size $M \times M$ is given by $s_t = (\mathbf{D}_t, \mathbf{p}_t, b_t, \mathbf{M}, \mathbf{U})$ and consists of five components:

- $\mathbf{D}_t \in \mathbb{R}^{K \times 2}$ represents the initially available and the already collected data for each device;
- $\mathbf{p}_t \in \mathbb{R}^2$ is the UAV position projected on the ground;

- $b_t \in \mathbb{N}$ is the UAV's remaining flying time;
- $\mathbf{M} \in \mathbb{B}^{M \times M \times 3}$ is the map of the physical environment in the Boolean domain $\{0, 1\}$ encoded with three map layers for start/landing positions, NFZs and buildings;
- $\mathbf{U} \in \mathbb{R}^{K \times 2}$ are the 2D coordinates of the K IoT devices.

Note that the state is transformed before being fed into the agent as detailed in III-C. Considering the five described components, the total size of the state space is

$$\mathcal{S} = \underbrace{\mathbb{R}^2}_{\text{Position}} \times \underbrace{\mathbb{B}^{M \times M \times 3}}_{\text{Environment Map}} \times \underbrace{\mathbb{R}^{K \times 2}}_{\text{Device Positions}} \times \underbrace{\mathbb{R}^{K \times 2}}_{\text{Device Data}} \times \underbrace{\mathbb{N}}_{\text{Flying Time}},$$

while the UAV is limited to six actions contained in the action space

$$\mathcal{A} = \{\text{north, east, south, west, hover, land}\}.$$

The reward function maps state-action pairs to a real-valued reward, i.e. $R: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Representing the mission goals, the reward function consists of the following components:

- r_{data} (*positive*) the data collection reward given by the achieved throughput (4) in the current time slot;
- r_{sc} (*negative*) safety controller (SC) penalty in case the drone has to be prevented from colliding with a building or entering an NFZ;
- r_{mov} (*negative*) constant movement penalty that is applied for every action the UAV takes without completing the mission;
- r_{crash} (*negative*) penalty in case the drone's remaining flying time reaches zero without having landed safely in a landing zone.

III. METHODOLOGY

A. Q-Learning

Q-learning is a model-free RL method [11] where a cycle of interaction between an agent and the environment enables the agent to learn and optimize a behavior, i.e. the agent observes state $s_t \in \mathcal{S}$ and performs an action $a_t \in \mathcal{A}$ at time t and the environment subsequently assigns a reward $r(s_t, a_t) \in \mathbb{R}$ to the agent. The cycle restarts with the propagation of the agent to the next state s_{t+1} . The agent's goal is to learn a behavior rule, referred to as a policy that maximizes the reward it receives. A probabilistic policy $\pi(a|s)$ is a distribution over actions given the state such that $\pi: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. In the deterministic case, it reduces to $\pi(s)$ such that $\pi: \mathcal{S} \rightarrow \mathcal{A}$.

Q-learning is based on iteratively improving the state-action value function or Q-function to guide and evaluate the process of learning a policy π . It is given as

$$Q^\pi(s, a) = \mathbb{E}_\pi [R_t | s_t = s, a_t = a] \quad (5)$$

and represents an expectation of the discounted cumulative return R_t from the current state s_t up to a terminal state at time T given by

$$R_t = \sum_{k=t}^T \gamma^{k-t} r(s_k, a_k) \quad (6)$$

with $\gamma \in [0, 1]$ being the discount factor, balancing the importance of immediate and future rewards. For the ease of exposition, s_t and a_t are abbreviated to s and a and s_{t+1} and a_{t+1} to s' and a' in the following.

B. Double Deep Q-learning and Combined Experience Replay

As demonstrated in [7], representing the Q-function (5) as a table of values is not efficient in the large state and action spaces of UAV trajectory planning. Through the work of Mnih *et al.* [12] on the application of techniques such as experience replay, it became possible to stably train large neural networks with parameters θ , referred to as deep Q-networks (DQNs), to approximate the Q-function instead.

Experience replay is a technique to reduce correlations in the sequence of training data where new experiences made by the agent, represented by quadruples of (s, a, r, s') , are stored in the replay memory \mathcal{D} . During training, minibatches of size m are sampled uniformly from \mathcal{D} , where the buffer size $|\mathcal{D}|$ was shown to be an important hyperparameter for the agent's performance and must be carefully tuned for different scenarios. Zhang and Sutton [13] proposed combined experience replay as a remedy for this sensitivity with very low computational complexity $\mathcal{O}(1)$. Then, only $m-1$ samples of the minibatch are sampled from memory, while the agent's latest experience is always added. Therefore, all new transitions influence the agent immediately, making the agent less sensitive to the selection of the replay buffer size.

Further improvements to the training process were suggested in [14], resulting in the inception of double deep Q-networks (DDQNs). We train our network with parameters θ accordingly to minimize the loss function given by

$$L(\theta) = \mathbb{E}_{s, a, s' \sim \mathcal{D}} [(Q_\theta(s, a) - Y(s, a, s'))^2] \quad (7)$$

where the target value, computed using a separate target network with parameters $\bar{\theta}$, is given by

$$Y(s, a, s') = r(s, a) + \gamma Q_{\bar{\theta}}(s', \underset{a'}{\operatorname{argmax}} Q_\theta(s', a')). \quad (8)$$

C. Centered Global Map

The global map is composed of the static environmental map and a dynamic device data map, which is formatted as two real-valued map layers. The first layer represents the data available for collection from each device at its respective position and the second layer records the data that has already been collected throughout the mission.

With this encoding, it would be possible to feed the map data directly into the agent as it was done in [3], with an input space defined through

$$\mathcal{I} = \underbrace{\mathbb{R}^2}_{\text{Position}} \times \underbrace{\mathbb{B}^{M \times M \times 3}}_{\text{Environment Map}} \times \underbrace{\mathbb{R}^{M \times M \times 2}}_{\text{Device Data Map}} \times \underbrace{\mathbb{N}}_{\text{Flying Time}}.$$

In this work, we show that centering the map layers on the UAV's position greatly benefits its ability to generalize over varying scenario parameters. While centering an input map was already applied to local maps that only show the area immediately surrounding the agent, such as in the related field

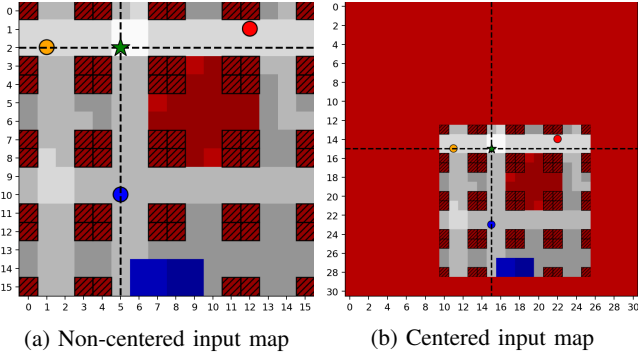


Fig. 1: Comparison of non-centered and centered input maps, with UAV's position represented by the green star and the intersection of the dashed lines.

	Symbol	Description
DQN Input	■	Start and landing zone
	■	Regulatory no-fly zone (NFZ)
	▨	Buildings blocking wireless links
	●	IoT device
Visualization	■	Summation of building shadows
	◇	Starting and landing positions during an episode
	→	UAV movement while comm. with green device
	+	Hovering while comm. with green device
	→ ✗	Actions without comm. (all data collected)

TABLE I: Legend for scenario plots.

of UAV navigation [15], we apply it for the first time to global maps in a UAV data collection scenario.

The map centering process inside the computational graph is illustrated in Fig. 1 with a legend provided in Table I. For centering, the maps are expanded to $(2M - 1) \times (2M - 1)$ in order to enable the agent to observe the entire map independent of its position in it. Translation of the original map centers the expanded map on the UAV's position. The resulting input space is defined through

$$\mathcal{I}_c = \underbrace{\mathbb{B}^{(2M-1) \times (2M-1) \times 3}}_{\text{Centered Environment Map}} \times \underbrace{\mathbb{R}^{(2M-1) \times (2M-1) \times 2}}_{\text{Centered Device Data Map}} \times \underbrace{\mathbb{N}}_{\text{Flying Time}}.$$

The benefit of using a centered map is the result of a change in position to which a neuron of the "Flatten" layer (see Fig. 2) corresponds. If the map is not centered, the neurons in that layer correspond to features at *absolute* positions. If the map is centered, they correspond to features at positions *relative* to the agent. Since the agent's actions are solely based on its relative position to features, e.g. its distance to devices, learning efficiency increases considerably.

D. Neural Network Model

Fig. 2 shows the DQN structure and the map centering pre-processing. The centered map is fed through convolutional layers with ReLU activation and then flattened and concatenated with the scalar input indicating remaining flight time. After passing through fully connected layers with ReLU activation,

the data reaches the last fully-connected layer of size $|\mathcal{A}|$ and without activation function, directly representing the Q-values for each action given the input state. The argmax of the Q-values, the greedy policy is given by

$$\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q_\theta(s, a). \quad (9)$$

It is deterministic and used when evaluating the agent. During training, the soft-max policy

$$\pi(a_i | s) = \frac{e^{Q_\theta(s, a_i) / \beta}}{\sum_{\forall a_j \in \mathcal{A}} e^{Q_\theta(s, a_j) / \beta}} \quad (10)$$

is used. The temperature parameter $\beta \in \mathbb{R}$ scales the balance of exploration versus exploitation.

IV. SIMULATIONS

A. Simulation Setup

The UAV starts each new mission in a world discretized into 16×16 cells where each grid cell is of size $10\text{m} \times 10\text{m}$. It starts with a remaining flying time of T steps, which is decremented by one after every action the agent takes, no matter if moving or hovering. The UAV flies at a constant altitude of $h = 10\text{m}$ inside 'urban canyons' through a city environment or open fields and is, for regulatory reasons, not allowed to fly over buildings, enter NFZs, or leave the 16×16 grid.

Each mission time slot contains $\delta = 4$ scheduled communication time slots. Propagation parameters (see II-A) are chosen in-line with [5] according to the urban micro scenario with $\alpha_{\text{LoS}} = 2.27$, $\alpha_{\text{NLoS}} = 3.64$, $\sigma_{\text{LoS}}^2 = 2$ and $\sigma_{\text{NLoS}}^2 = 5$. The shadowing maps to simulate the environment were computed using ray tracing from and to the center points of cells. Transmission and noise powers are normalized through the definition of a cell-edge SNR of -15dB, which describes the SNR between the drone on ground level at the very center of the map and an unobstructed device at one of the grid corners. The agent has absolutely no prior knowledge of the shadowing maps or wireless channel characteristics.

We use the following metrics to evaluate the agent's performance in different scenarios and to compare training instances:

- *Cumulative reward*: the sum of all rewards received throughout an episode;
- *Has landed*: records whether the agent landed in time at the end of an episode;
- *Collection ratio*: the ratio of collected data to total initially available device data at the end of a mission;
- *Collection ratio and landed*: the product of *has landed* and *collection ratio* per episode.

Evaluation is challenging as we train a single agent to generalize over a large scenario parameter space. During training, we evaluate the agent's training progress in a randomly selected scenario every ten episodes and form an average over multiple evaluations. As it is computationally infeasible to evaluate the trained agent on all possible scenario variations, we perform Monte Carlo analysis on a large number of randomly selected scenario parameter combinations.

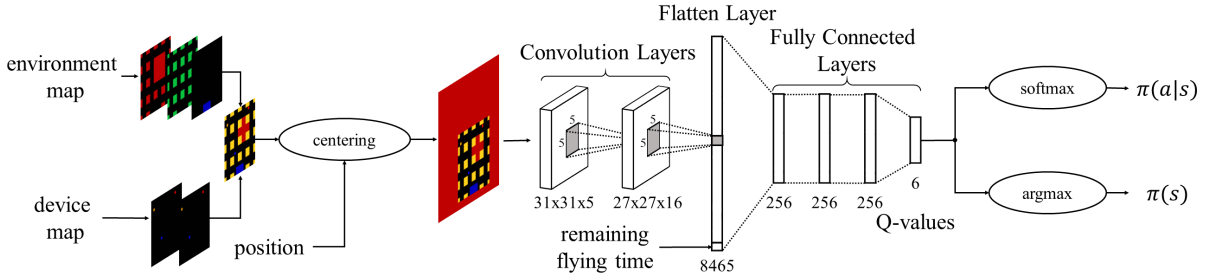
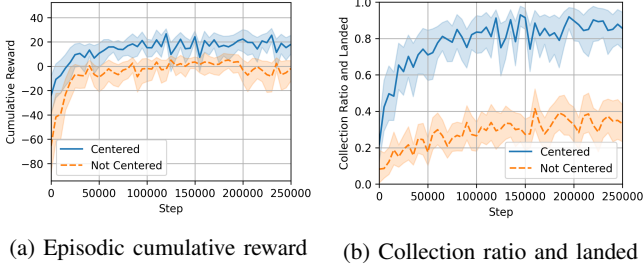


Fig. 2: DQN architecture with map centering, with the device map encoded in separate layers but visualized in RGB channels.



(a) Episodic cumulative reward (b) Collection ratio and landed

Fig. 3: Training process comparison between centered and non-centered map input showing the average and 99% quantiles of three training processes each, with episodic metrics grouped in bins of 5000 step width.

B. Centered vs. Non-Centered Map

Centering the map information on the UAV’s position as described in III-C proved to be highly beneficial to the learning performance and the generalization ability of the DDQN agent. Fig. 3 shows comparisons of two performance metrics, cumulative reward per episode, and achieved data collection ratio in missions with in-time landing over training time, for centered and non-centered map inputs in identical scenarios. To compare the two approaches, the input of the non-centered agent is padded with NFZ cells to have the same shape as the centered agent. The only difference between the agents is that the non-centered agent receives the position as a 2D-one-hot encoded map layer similar to [3]. Each graph is averaged over three training runs to account for possible random variations in the training process. A clear performance advantage for the agent using the centered map input can be seen throughout the whole learning process.

C. Collectible Data and Device Accessibility

The scenario map in Fig. 4 is divided into an open field and an adjacent city. To show the agent’s responsiveness to differences in collectible data at the same devices, we fixed the number of IoT devices to $K = 2$, while allowing for fully randomized device positions in unoccupied map space, for each device randomized collectible data ($D_0 \in [1.0, 25.0]$ data units), randomized flying time limits ($b_0 \in [35, 70]$ steps) and eight possible start positions.

Fig. 4 shows the agent adapting to a change in collectible data at the two devices. The agent only enters the hard to navigate courtyard if the amount of data at the orange

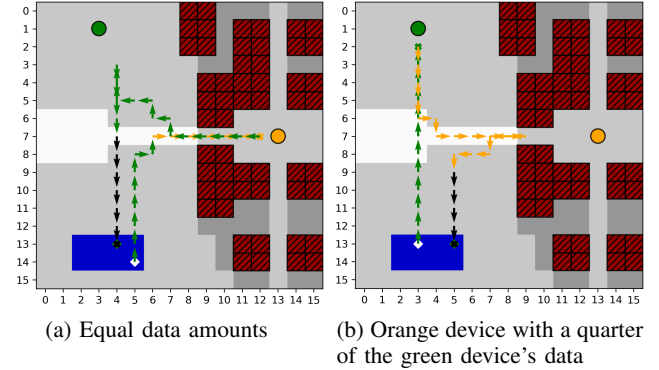


Fig. 4: Illustration of the same agent adapting to differences in collectible data with all other mission parameters fixed.

device requires it. While starting to communicate with the unobstructed green device in Fig. 4a, the agent proceeds to collect data from the harder-to-access orange device first, then picking up the rest from the green device before returning straight to the landing area. For the case in Fig. 4b, the UAV changes its strategy. While immediately reducing its distance to the green node after starting and collecting all its data, it collects the data from the orange device on the way back with a detour only as long as required, minimizing the overall mission duration. The UAV is also clearly able to identify unobstructed positions to communicate with the orange device.

D. Manhattan Scenario

The main scenario we investigate is defined by a Manhattan-like city structure (see Fig. 5) containing regularly distributed city blocks with streets in between, as well as an NFZ district. In this challenging setting, we want to demonstrate the agent’s ability to generalize over significant variations in scenario parameters with randomly changing device count ($K \in [2, 5]$), device data ($D_0 \in [5.0, 20.0]$ data units), maximum flying time ($b_0 \in [35, 70]$ steps), and eight possible starting positions. Similar to the previous scenario, device positions are randomized throughout the unoccupied map space.

This and the previous scenario are evaluated using Monte Carlo simulations on their full range of scenario parameters with average performance metrics shown in Table II. Both agents show a similarly high successful landing performance. It is expected that the collection ratio must be less than

100% in some scenario instances depending on the randomly assigned maximum flying time and IoT device parameters.

In Fig. 5, four scenario instances chosen from the random Monte Carlo evaluation for device counts of $K \in \{2, 3, 4, 5\}$ for 5a through 5d illustrate the agent’s adaptability. With $K = 2$ devices in Fig. 5a, finding a trajectory is complicated by the location of the blue device inside the NFZ and the resulting shadowing effects, which have to be deduced by the agent from building and device positions. In Fig. 5b, the considerable distance to the red device requires the agent to exhaust its entire flight time. For the scenario in Fig. 5c the available flying time $T = 35$ is not sufficient to collect all data. Therefore, the agent ignores the isolated blue device and lands early after collecting all data within reach. In Fig. 5d the agent successfully collects all data in an efficient order while minimizing its flying time, e.g. by turning away from the green device before transmitting all its data. We observed that rerunning the same scenario configuration leads to a variation in trajectories which adapt to effects of the random communication channel fading.

Metric	Manhattan	Open Field and City
Has Landed	99.5%	99.9%
Collection Ratio	94.8%	90.0%
Collection Ratio and Landed	94.6%	89.9%

TABLE II: Performance metrics averaged over 1000 random scenario Monte Carlo iterations.

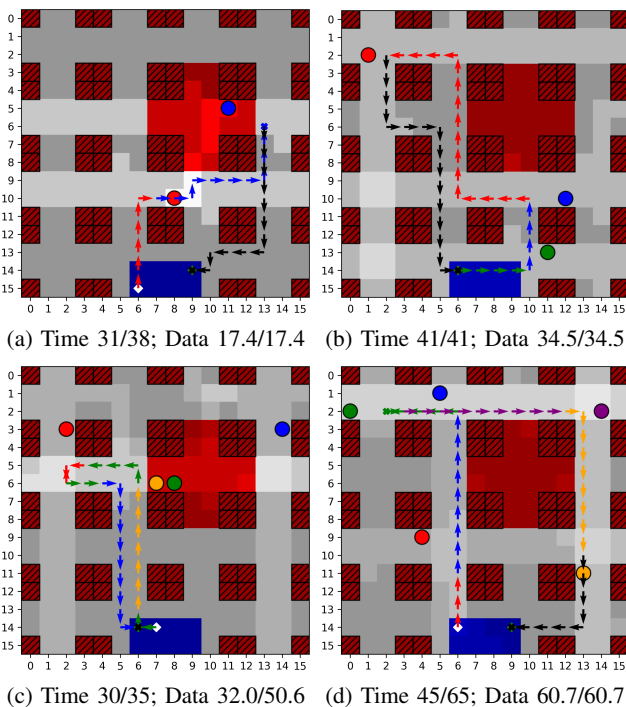


Fig. 5: Illustration of the same agent adapting to differences in device count and device placement as well as flight time limits, showing used and available flying time and collected and available total data in the Manhattan scenario.

V. CONCLUSION

We have introduced a new DDQN method with combined experience replay for UAV trajectory planning in an IoT data harvesting scenario. By leveraging a neural network model that exploits information about the environment from centered map layers through convolutional processing, we show that the UAV agent learns to effectively adapt to significant variations in the scenario such as number and position of IoT devices, amount of collectible data or maximum flying time, without the need for expensive retraining or recollection of training data. Using this method, we have shown that the UAV balances the goals of data collection, obstacle avoidance, and minimizing mission time effectively, while not requiring any prior information about the challenging wireless channel characteristics in an urban environment. In future work, we will tackle the issue of scalability to larger maps, namely the linear increase of trainable parameters in the flatten layer with map area. We also envision to combine our approach with multi-task reinforcement learning or transfer learning [10], as well as extending the UAV’s action space to altitude control.

REFERENCES

- [1] Y. Zeng, Q. Wu, and R. Zhang, “Accessing from the sky: A tutorial on UAV communications for 5G and beyond,” *Proceedings of the IEEE*, vol. 107, no. 12, pp. 2327–2375, 2019.
- [2] M. Minevich, “How Japan is tackling the national & global infrastructure crisis & pioneering social impact - [news],” *Forbes*, 21 Apr 2020.
- [3] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, “UAV coverage path planning under varying power constraints using deep reinforcement learning,” *arXiv:2003.02609 [cs.RO]*, accepted at *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2020.
- [4] P. S. Bithas, E. T. Michailidis, N. Nomikos, D. Vouyioukas, and A. G. Kanatas, “A survey on machine-learning techniques for UAV-based communications,” *Sensors*, vol. 19, no. 23, p. 5170, 2019.
- [5] O. Esrafilian, R. Gangula, and D. Gesbert, “Learning to communicate in UAV-aided wireless networks: Map-based approaches,” *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1791–1802, 2018.
- [6] C. You and R. Zhang, “Hybrid offline-online design for UAV-enabled data harvesting in probabilistic los channel,” *IEEE Transactions on Wireless Communications*, 2020.
- [7] H. Bayerlein, P. De Kerret, and D. Gesbert, “Trajectory optimization for autonomous flying base station via reinforcement learning,” in *IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2018.
- [8] H. Qi, Z. Hu, H. Huang, X. Wen, and Z. Lu, “Energy efficient 3-D UAV control for persistent communication service and fairness: A deep reinforcement learning approach,” *IEEE Access*, vol. 8, 2020.
- [9] C. H. Liu, Z. Dai, Y. Zhao, J. Crowcroft, D. O. Wu, and K. Leung, “Distributed and energy-efficient mobile crowdsensing with charging stations by deep reinforcement learning,” *IEEE Transactions on Mobile Computing*, 2019.
- [10] G. Dulac-Arnold, D. Mankowitz, and T. Hester, “Challenges of real-world reinforcement learning,” *arXiv:1904.12901 [cs.LG]*, 2019.
- [11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: an introduction*. MIT Press, second ed., 2018.
- [12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, et al., “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [13] S. Zhang and R. S. Sutton, “A deeper look at experience replay,” *arXiv:1712.01275 [cs.LG]*, 2017.
- [14] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” in *Thirtieth AAAI conference on artificial intelligence*, pp. 2094–2100, 2016.
- [15] B. G. Maciel-Pearson, L. Marchegiani, S. Akcay, A. Atapour-Abarghouei, J. Garforth, and T. P. Breckon, “Online deep reinforcement learning for autonomous UAV navigation and exploration of outdoor environments,” *arXiv:1912.05684 [cs.CV]*, 2019.

4.3 UAV Path Planning using Global and Local Map Information with Deep Reinforcement Learning

Reference

M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, “UAV path planning using global and local map information with deep reinforcement learning,” in *2021 20th International Conference on Advanced Robotics (ICAR)*, pp. 539–546, IEEE, 2021
DOI: <https://doi.org/10.1109/ICAR53236.2021.9659413>

Abstract

Path planning methods for autonomous unmanned aerial vehicles (UAVs) are typically designed for one specific type of mission. This work presents a method for autonomous UAV path planning based on deep reinforcement learning (DRL) that can be applied to a wide range of mission scenarios. Specifically, we compare coverage path planning (CPP), where the UAV’s goal is to survey an area of interest to data harvesting (DH), where the UAV collects data from distributed Internet of Things (IoT) sensor devices. By exploiting structured map information of the environment, we train double deep Q-networks (DDQNs) with identical architectures on both distinctly different mission scenarios to make movement decisions that balance the respective mission goal with navigation constraints. By introducing a novel approach exploiting a compressed global map of the environment combined with a cropped but uncompressed local map showing the vicinity of the UAV agent, we demonstrate that the proposed method can efficiently scale to large environments. We also extend previous results for generalizing control policies that require no retraining when scenario parameters change and offer a detailed analysis of crucial map processing parameters’ effects on path planning performance.

Contributions to this paper

- Conceptualization of global-local map processing for UAV path planning
- Adaptation of the CPP and DH code for global-local processing and target generalization
- Conducting training and evaluation experiments
- Majority of paper writing

Copyright

© 2021 IEEE. Reprinted, with permission, from Mirco Theile, Harald Bayerlein, Richard Nai, David Gesbert, and Marco Caccamo, “UAV Path Planning using Global and Local Map Information with Deep Reinforcement Learning”, 2021 20th International Conference on Advanced Robotics (ICAR), December 2021.

See Appendix A.5 for the reuse statement. The following shows the accepted version.

UAV Path Planning using Global and Local Map Information with Deep Reinforcement Learning

Mirco Theile¹, Harald Bayerlein², Richard Nai¹, David Gesbert², and Marco Caccamo¹

Abstract— Path planning methods for autonomous unmanned aerial vehicles (UAVs) are typically designed for one specific type of mission. This work presents a method for autonomous UAV path planning based on deep reinforcement learning (DRL) that can be applied to a wide range of mission scenarios. Specifically, we compare coverage path planning (CPP), where the UAV’s goal is to survey an area of interest to data harvesting (DH), where the UAV collects data from distributed Internet of Things (IoT) sensor devices. By exploiting structured map information of the environment, we train double deep Q-networks (DDQNs) with identical architectures on both distinctly different mission scenarios to make movement decisions that balance the respective mission goal with navigation constraints. By introducing a novel approach exploiting a compressed global map of the environment combined with a cropped but uncompressed local map showing the vicinity of the UAV agent, we demonstrate that the proposed method can efficiently scale to large environments. We also extend previous results for generalizing control policies that require no retraining when scenario parameters change and offer a detailed analysis of crucial map processing parameters’ effects on path planning performance.

I. INTRODUCTION

Autonomous unmanned aerial vehicles (UAVs) are envisioned for a multitude of applications that all require efficient and safe path planning methods, which necessitate the combination of a mission goal with navigation constraints, e.g., flying time and obstacle avoidance. Examples for these applications are area coverage path planning (CPP) [1], and data harvesting (DH) from Internet of Things (IoT) sensor nodes [2]. As its name suggests, covering all points inside an area of interest with CPP is related to conventional path planning, where the goal is to find a path between start and goal positions. In general, CPP aims to cover as much of the target area as possible within given energy or path-length constraints while avoiding obstacles or no-fly zones.

In the DH scenario, the UAV’s goal is to collect data from IoT devices distributed in an urban environment, which implies challenging radio channel conditions through alternating line-of-sight (LoS) and non-line-of-sight (NLoS) links between UAV and IoT devices through building obstruction. DH and CPP are very similar when described as an RL problem since the path planning problem’s constraints are mostly identical, and only the goal function changes. In

previous work, we have looked at CPP [1] and DH [2] separately. We show that both problems can be solved using the same deep reinforcement learning (DRL) approach based on feeding spatial map information directly to the DRL agent via convolutional network layers. This work’s focus lies in proposing improvements to existing DRL approaches to generalized, large-scale UAV path planning problems with CPP and DH as examples.

Using maps as a direct input becomes problematic for large map sizes, as the network’s size, trainable parameters, and training time increase equivalently. We introduce a global-local map scheme to address the scalability problems of the standard map-based input. In path planning, the intuition is that distant features lead to general direction decisions, while close features lead to immediate actions such as collision avoidance. Thus, the level of detail passed to the agent for distant objects can be less than for close objects. With the global map, a compressed version of the complete environment map centered on the agent’s position, general information of all objects on the map is provided to the agent. In contrast, the local map, uncompressed but cropped to show only the UAV agent’s immediate surroundings, provides detailed local information.

While numerous path planning algorithms for both problems exist, DRL offers the possibility to solve both distinctly different problems with the same approach. For each problem, DRL agents can learn control policies that generalize over a large scenario parameter space requiring no expensive retraining or recomputation when the scenario changes. However, previous work usually only focuses on finding optimal paths for one single scenario at a time. The DRL paradigm is popular in this context because of its flexibility regarding prior knowledge and assumptions about the environment, the computational efficiency of DRL inference, as well as the complexity of autonomous UAV control tasks, which are usually non-convex optimization problems and proven to be NP-hard in many instances [3], [4]. A general summary of issues in using UAVs as part of communication networks, including IoT data harvesting, can be found in [3]. A survey of various applications for UAV systems from a cyber-physical perspective is offered in [4]. Cabreira *et al.* [5] provide a survey of UAV coverage path planning.

Previous works in UAV path planning have already made use of convolutional map processing for DRL agents. In the drone patrolling problem presented in [6], a local relevance map of the patrolling area showing the agent’s vicinity cropped to a fixed size is fed into a DDQN agent. No information of the physical environment is included, and there

¹Mirco Theile, Richard Nai, and Marco Caccamo are with the TUM School of Engineering and Design, Technical University of Munich, Germany {mirco.theile, richard.nai, mcaccamo}@tum.de

²Harald Bayerlein and David Gesbert are with the Communication Systems Department, EURECOM, Sophia Antipolis, France {harald.bayerlein, david.gesbert}@eurecom.fr

is no consideration for navigation constraints like obstacle avoidance or flying time. In [7], fixed-wing UAVs are tasked with monitoring a wildfire propagating stochastically over time. Control decisions are based on either direct observations or belief maps fed into the DRL agents. The focus here is the inherent uncertainty of the problem, not balancing a mission goal and navigation constraints in large complex environments. Wildfire surveillance is also the mission of the quadcopter UAVs in [8], which is set in a similar scenario without navigation constraints and makes use of uncertainty maps to guide path planning. Their approach is based on an extended Kalman filter and not the reinforcement learning (RL) paradigm. To monitor another natural disaster situation, Baldazo *et al.* [9] present a multi-agent DRL method for flood surveillance using the UAVs' local observations of the inundation map to make control decisions. All mentioned approaches focus on solving a single class of UAV missions in simple physical environments and do not consider combining local and global map information.

Missions, where UAVs provide communication services to ground users or devices, include the work in [10] set in a complex urban environment where the UAV path planning is based on exploiting map information with a method combining dynamic and sequential convex programming. In [11], data is collected simultaneously with ground and aerial vehicles on a small map with obstacles. Due to the small map size, the full global map information can be fed into the DRL agents. Another scenario is investigated in [12], where a cellular-connected UAV has to navigate from a start to an end position maintaining connectivity with a ground network exploiting a radio map. The approach includes global radio map compression to reduce computational complexity but is not based on RL and includes no higher precision local map or hard navigation constraints. To the best of our knowledge, no dual global-local map processing method applicable to multiple mission types for autonomous UAVs has been suggested previously.

The main contributions of this paper are the following:

- Establishing the presented DRL approach as a general method for UAV path planning by demonstrating its applicability to two distinctly different mission scenarios: coverage path planning and path planning for wireless data harvesting;
- Introducing a novel approach¹ to exploit global-local map information that allows DRL for path planning to scale to large, realistic scenario environments efficiently, with an order of magnitude more grid cells compared to earlier works [1], [2];
- Overcoming the limitation of fixed target zones in previous DRL CPP approaches [1] by extending control policy generalization over scenario parameters to randomly generated target zones;
- Analyzing and discussing the effects of key map processing parameters on the path learning performance.

¹<https://www.github.com/theilem/uavSim.git>

II. PROBLEM FORMULATION

In the following, we show that a universal problem description of coverage path planning and path planning for data harvesting can be established through separation into two parts: the environment and the target.

A. Environment and UAV Model

We consider a square grid world of size $M \times M \in \mathbb{N}^2$ with cell size c , where \mathbb{N} is the set of natural numbers. The environment contains designated start/landing positions, regulatory no-fly zones (NFZs), and obstacles. The map can be described through a tensor $\mathbf{M} \in \mathbb{B}^{M \times M \times 3}$, where $\mathbb{B} = \{0, 1\}$ and with the start/landing zones in map-layer 1, the union of NFZs and obstacles in map-layer 2, and the obstacles alone in map-layer 3.

The UAV moves through this environment at a constant altitude h occupying one cell of the environment. Its position can thus be defined through $\mathbf{p}(t) \in \mathbb{N}^2$. The movement of the UAV is constrained through collision avoidance with obstacles and not entering NFZs. Additionally, the UAV must start and end its mission in any cell belonging to the start and landing zones while staying within its maximum flying time determined by its initial battery level. The battery level of the UAV $b(t)$ is set to $b_0 \in \mathbb{N}$ at time $t = 0$ and is decremented by 1 per action step.

B. Target and Mission Definitions

1) *Coverage Path Planning*: In coverage path planning, the mission is to cover a designated target area by flying above or near it, such that it is in the field of view of a camera-like sensor mounted underneath the UAV. The target area can be described through $\mathbf{T}(t) \in \mathbb{B}^{M \times M}$, in which each element describes whether a cell has to be covered or not. The current field of view of the camera can be described with $\mathbf{V}(t) \in \mathbb{B}^{M \times M}$ indicating for each cell whether it is in the current field of view or not. In this work, the field of view is a square of 5×5 surrounding the current UAV position. Additionally, buildings can block line-of-sight, which is also incorporated in calculating $\mathbf{V}(t)$. This prohibits the UAV from *seeing* around the corner.

Consequently, the target area evolves according to

$$\mathbf{T}(t+1) = \mathbf{T}(t) \wedge \neg \mathbf{V}(t), \quad (1)$$

in which \wedge and \neg are the cell-wise logical *and* and *negation* operators, respectively. In our mission definition, obstacle cells in the environment cannot be a coverage target, while start and landing zones and no-fly-zones can be. The goal is to cover as much of the target area as possible within the maximum flying time constraint.

2) *Data Harvesting*: Conversely, the mission in path planning for wireless data harvesting is to collect data from $K \in \mathbb{N}$ stationary IoT devices spread throughout the environment at ground-level, with the position of device $k \in [1, K]$ given through $\mathbf{u}_k \in \mathbb{N}^2$. Each device has an amount of data $D_k(t) \in \mathbb{R}$ to be collected by the UAV. The data throughput $C_k(t)$ between the selected device k and the UAV is based on the standard log-distance path loss model with Gaussian

shadow fading and whether they can establish a line-of-sight connection or are obstructed by obstacles. The UAV is communicating with one device at a time and selects the device with remaining data and the highest possible data rate. A detailed description of the link performance and multiple access protocol can be found in [2]. The data at each device evolves according to

$$D_k(t+1) = D_k(t) - C_k(t) \quad (2)$$

Devices can be located in every cell except for the starting and landing zones or inside obstacles. The goal of the data harvesting problem is to collect as much of the devices' data as possible within the maximum flying time.

3) *Unifying Map-Layer Description*: Both problems can be described through a single target map-layer $\mathbf{D}(t) \in \mathbb{R}^{M \times M}$. In CPP, the target map-layer is given through $\mathbf{T}(t)$ evolving according to (1). In DH, the target map-layer shows the amount of available data in each cell that one of the devices is occupying, i.e. the cell at position \mathbf{u}_k has value $D_k(t)$ and is evolving according to (2). If a cell does not contain a device or the device data has been collected fully, the cell's value is 0. Since the two problems can be described with similar state representations, both can be solved through deep reinforcement learning with a neural network having the same structure.

III. METHODOLOGY

While a variety of methods exist to solve the CPP and DH problems separately, the approach presented in the following can be directly applied to both distinct path planning problems. In most classical CPP approaches, individual target zones are extracted through segmentation and then connected with distance costs into a graph, while each segment is covered with a boustrophedon path. This reduces the CPP problem to an instance of the travelling salesman problem (TSP), which is NP-hard and can be solved by classical methods, e.g. as demonstrated in [13] at the price of an exponential increase in time complexity with the number of target zones.

In principle, the DH problem can also be converted into a TSP with the IoT devices as nodes in the graph and the distances between the devices as edge costs. However, the conversion neglects that communication with the device happens while traveling to and from it. In general, the optimal behavior in DH problems is not a sequential visit of all devices, as data can already be efficiently collected by establishing a LoS link from farther away, or a large amount of data waiting to be collected might require the drone to hover for an extended period of time near the device. These constraints in conjunction with stochastic communication channel models and the various possibilities for the choice of multiple access protocol are non-trivial to model and solve with classical approaches. For both problems, the UAV battery constraint adds another complication for classical approaches, as full coverage or full collection are not always feasible. The following DRL methodology allows us to combine all goals and constraints of the respective path

planning problems directly without the need for additional approximations.

A. Partially Observable Markov Decision Process

To address the aforementioned problems we formulate them as a partially observable Markov decision process (POMDP) [14] which is defined through the tuple $(\mathcal{S}, \mathcal{A}, P, R, \Omega, \mathcal{O}, \gamma)$. In the POMDP, \mathcal{S} describes the state space, \mathcal{A} the action space, and $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ the transition probability function. $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function mapping state, action, and next state to a real valued reward. The observation space is defined through Ω and $\mathcal{O} : \mathcal{S} \mapsto \Omega$ is the observation function. The discount factor $\gamma \in [0, 1]$ varies the importance of long and short term rewards.

We unify the UAV path planning problems by describing their state space with

$$\mathcal{S} = \underbrace{\mathbb{B}^{M \times M \times 3}}_{\text{Environment Map}} \times \underbrace{\mathbb{R}^{M \times M}}_{\text{Target Map}} \times \underbrace{\mathbb{N}^2}_{\text{Position}} \times \underbrace{\mathbb{N}}_{\text{Flying Time}}, \quad (3)$$

in which the elements $s(t) \in \mathcal{S}$ are

$$s(t) = (\mathbf{M}, \mathbf{D}(t), \mathbf{p}(t), b(t)). \quad (4)$$

The four components of the tuple are

- \mathbf{M} the environment map containing start and landing zones, no-fly zones, and obstacles;
- $\mathbf{D}(t)$ the target map indicating remaining data at device locations or remaining cells to be uncovered at time t ;
- $\mathbf{p}(t)$ the UAV's position at time t ;
- $b(t)$ the UAV's remaining movement budget at time t ;

Action $a(t) \in \mathcal{A}$ of the UAV at time t is given as one of the possible actions

$$\mathcal{A} = \{\text{north, east, south, west, hover, land}\}.$$

The generalized reward function $R(s(t), a(t), s(t+1))$ consists of the following elements:

- r_c (*positive*) the data collection or cell covering reward given by the collected data or the amount of newly covered target cells, comparing $s(t+1)$ and $s(t)$;
- r_{sc} (*negative*) safety controller (SC) penalty in case the drone has to be prevented from colliding with a building or entering an NFZ;
- r_{mov} (*negative*) constant movement penalty that is applied for every action the UAV takes without completing the mission;
- r_{crash} (*negative*) penalty in case the drone's remaining flying time reaches zero without having landed safely in a landing zone.

B. Map Processing

To aid an agent in interpreting the large state space given in (3), two map processing steps are used. The first is centering the map around the agent's position, shown in [2] to improve the agent's performance significantly. The downside of this approach is that it increases the representation size of the state space even further. Thus, the second map processing

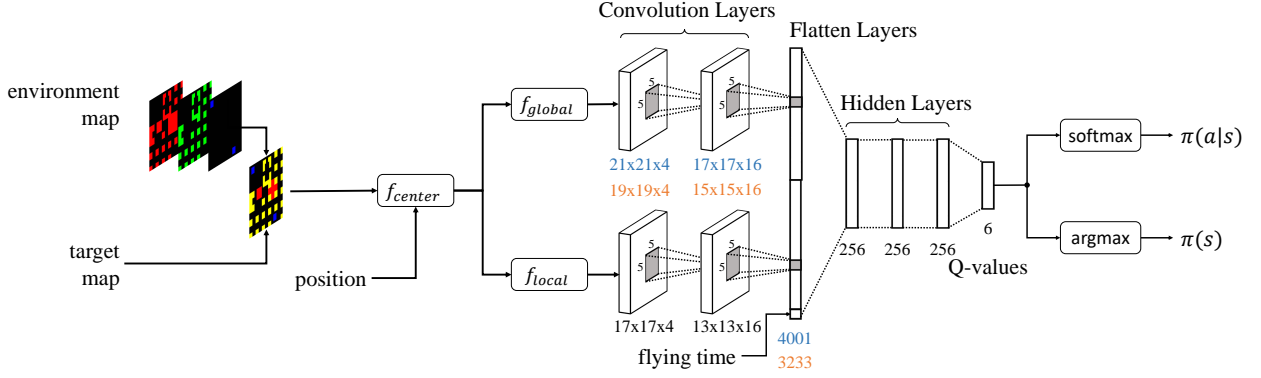


Fig. 1: DQN architecture with map centering and global and local mapping, showing differences of layer size in blue for the 'Manhattan32' and orange for the 'Urban50' scenario.

step, the main contribution of this work, is to present the centered map as two inputs: a full-detail local map showing the agent's immediate surroundings and a compressed global map showing the entire environment with less detail. The mathematical description of the three functions is presented in the following. Fig. 1 indicates where the functions are used within the data pipeline.

1) *Map Centering*: Given a tensor $\mathbf{A} \in \mathbb{R}^{M \times M \times n}$ describing the map layers of the environment, a centered tensor $\mathbf{B} \in \mathbb{R}^{M_c \times M_c \times n}$ with $M_c = 2M - 1$ is defined through

$$\mathbf{B} = f_{\text{center}}(\mathbf{A}, \mathbf{p}, \mathbf{x}_{\text{pad}}), \quad (5)$$

with the centering function defined as

$$f_{\text{center}} : \mathbb{R}^{M \times M \times n} \times \mathbb{N}^2 \times \mathbb{R}^n \mapsto \mathbb{R}^{M_c \times M_c \times n}. \quad (6)$$

The elements of \mathbf{B} with respect to the elements of \mathbf{A} are defined as

$$\mathbf{b}_{i,j} = \begin{cases} \mathbf{a}_{i+p_0-M+1, j+p_1-M+1}, & M \leq i+p_0+1 < 2M \\ & \wedge M \leq j+p_1+1 < 2M \\ \mathbf{x}_{\text{pad}}, & \text{otherwise,} \end{cases} \quad (7)$$

effectively padding the map layers of \mathbf{A} with the padding value \mathbf{x}_{pad} . Note that $\mathbf{a}_{i,j}$, $\mathbf{b}_{i,j}$, and \mathbf{x}_{pad} are vector valued of dimension \mathbb{R}^n . For both problems, the map layers are padded with $[0, 1, 1, 0]^T$, i.e. NFZs and obstacles. A qualitative description of centering with an example can be found in [2].

2) *Global-Local Mapping*: The tensor $\mathbf{B} \in \mathbb{R}^{M_c \times M_c \times n}$ resulting from the map centering function is processed in two ways. The first is creating a local map according to

$$\mathbf{X} = f_{\text{local}}(\mathbf{B}, l) \quad (8)$$

with the local map function defined by

$$f_{\text{local}} : \mathbb{R}^{M_c \times M_c \times n} \times \mathbb{N} \mapsto \mathbb{R}^{l \times l \times n}. \quad (9)$$

The elements of \mathbf{X} with respect to the elements of \mathbf{B} are defined as

$$\mathbf{x}_{i,j} = \mathbf{b}_{i+M-\lceil \frac{l}{2} \rceil, j+M-\lceil \frac{l}{2} \rceil}. \quad (10)$$

This operation is effectively a central crop of size $l \times l$.

The global map is created according to

$$\mathbf{Y} = f_{\text{global}}(\mathbf{B}, g) \quad (11)$$

with the global map function defined by

$$f_{\text{global}} : \mathbb{R}^{M_c \times M_c \times n} \times \mathbb{N} \mapsto \mathbb{R}^{\lfloor \frac{M_c}{g} \rfloor \times \lfloor \frac{M_c}{g} \rfloor \times n}. \quad (12)$$

The elements of \mathbf{Y} with respect to the elements of \mathbf{B} are defined as

$$\mathbf{y}_{i,j} = \frac{1}{g^2} \sum_{u=0}^{g-1} \sum_{v=0}^{g-1} \mathbf{b}_{gi+u, gj+v}, \quad (13)$$

which is an operation equal to average pooling.

The functions f_{local} and f_{global} are parameterized through l and g , respectively. Increasing l increases the size of the local map, whereas increasing g increases the size of the average pooling cells, decreasing the size of the global map.

C. Observation Space

The observation space Ω , which is the input to the agent, is given as

$$\Omega = \Omega_l \times \Omega_g \times \mathbb{N}$$

containing the local map $\Omega_l = \mathbb{B}^{l \times l \times 3} \times \mathbb{R}^{l \times l}$ and the global map $\Omega_g = \mathbb{R}^{\lfloor \frac{M_c}{g} \rfloor \times \lfloor \frac{M_c}{g} \rfloor \times 3} \times \mathbb{R}^{\lfloor \frac{M_c}{g} \rfloor \times \lfloor \frac{M_c}{g} \rfloor}$. Note that the compression of the map layers through average pooling transforms the environment layers from boolean to real. The observations $o(t) \in \Omega$ are defined through the tuple

$$o(t) = (\mathbf{M}_l(t), \mathbf{D}_l(t), \mathbf{M}_g(t), \mathbf{D}_g(t), b(t)). \quad (14)$$

In the observation, $\mathbf{M}_l(t)$ and $\mathbf{M}_g(t)$ are the local and global observations of the environment, and $\mathbf{D}_l(t)$ and $\mathbf{D}_g(t)$ are the local and global observations of the target, respectively. $b(t)$ is the remaining flying time of the UAV and is equal to the one in the state space. Note that the local and global observations of the environment are time-dependent, as they are centered around the time-dependent position of the UAV.

The mapping from state to observation space is given by $\mathcal{O} : \mathcal{S} \mapsto \Omega$, with the elements $o(t) \in \mathcal{O}$ defined as:

$$\mathbf{M}_l(t) = f_{\text{local}}(f_{\text{center}}(\mathbf{M}, \mathbf{p}(t), [0, 1, 1]^T), l) \quad (15a)$$

$$\mathbf{D}_l(t) = f_{\text{local}}(f_{\text{center}}(\mathbf{D}(t), \mathbf{p}(t), 0), l) \quad (15b)$$

$$\mathbf{M}_g(t) = f_{\text{global}}(f_{\text{center}}(\mathbf{M}, \mathbf{p}(t), [0, 1, 1]^T), g) \quad (15c)$$

$$\mathbf{D}_g(t) = f_{\text{global}}(f_{\text{center}}(\mathbf{D}(t), \mathbf{p}(t), 0), g) \quad (15d)$$

By feeding the observation space Ω into the agent instead of the state space \mathcal{S} , the problem is artificially converted into a partially observable MDP. The partial observability results from the restricted size of the local map and the averaging in the global map. With the following results, we show that partial observability does not make the problem infeasible for memory-less agents and that the compression greatly reduces the size of the neural network, yielding significantly less training time.

D. Double Deep Reinforcement Learning - Neural Network

To solve the aforementioned POMDP, we use reinforcement learning, specifically double deep Q-networks (DDQN) proposed by Van Hasselt *et al.* [15]. DDQNs approximate the Q-value of each state-action pair given as

$$Q^\pi(s(t), a(t)) = \mathbb{E}_\pi \left[\sum_{k=t}^T \gamma^{k-t} R(s(k), a(k), s(k+1)) \right], \quad (16)$$

describing the discounted cumulative reward of an agent following policy π . To converge to the optimal Q-value the agent explores the environment, collecting experiences $(s(t), a(t), r(t), s(t+1))$ and storing them as (s, a, r, s') in a replay memory \mathcal{D} , omitting temporal information. Two Q-networks parameterized through θ and $\bar{\theta}$ are used, in which the first Q-network is updated by minimizing the loss

$$L(\theta) = \mathbb{E}_{s,a,s' \sim \mathcal{D}} [(Q_\theta(s, a) - Y(s, a, s'))^2] \quad (17)$$

given by experiences in the replay memory. The target value is given by

$$Y(s, a, s') = r(s, a) + \gamma Q_{\bar{\theta}}(s', \arg\max_{a'} Q_\theta(s', a')). \quad (18)$$

The parameters of the second Q-network are updated as $\bar{\theta} \leftarrow (1 - \tau)\bar{\theta} + \tau\theta$ with the soft update parameter $\tau \in (0, 1]$. To address training sensitivity to the size of the replay memory we make use of combined experience replay proposed by Zhang and Sutton [16].

The neural network architecture used for both Q-networks is shown in Fig. 1. The environment map and target map are stacked and centered around the UAV position and then converted into global and local observation components. After being fed through two convolutional layers each, the resulting tensors are flattened and concatenated with the remaining flying time input and passed through three hidden layers with ReLU activation functions. The output layer with no activation function represents the Q-values directly, passed through a softmax function to create an action distribution for exploration or an argmax function for exploitation.

Parameter	32×32	50×50	Description
$ \theta $	1,175,302	978,694	trainable parameters
l	17	17	local map size
g	3	5	global map scaling
n_c		2	number of conv. layers
n_k		16	number of kernels
s_k		5	conv. kernel width

TABLE I: Hyperparameters for 32×32 and 50×50 maps.

The relevant parameter for scalability is the size of the flatten layer. It can be calculated through

$$N = n_k \left(\left(l - n_c \lfloor \frac{s_k}{2} \rfloor \right)^2 + \left(\lfloor \frac{M_c}{g} \rfloor - n_c \lfloor \frac{s_k}{2} \rfloor \right)^2 \right) + 1 \quad (19)$$

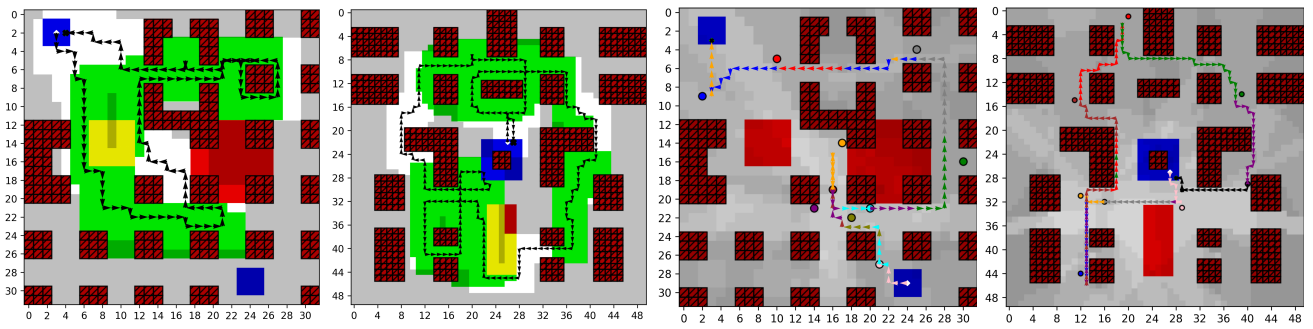
with n_k being the number of kernels, n_c the number of convolutional layers, and s_k being the kernel size. Setting the global map scaling parameter to $g = 1$ and the local map size to $l = 0$ deactivates global-local map processing, i.e., no downsampling and no extra local map. The parameters used in evaluation are listed in Table I.

IV. SIMULATIONS

A. Simulation Setup

The UAV is flying in two different grid worlds. The 'Manhattan32' scenario (Fig. 2a and 2c) with 32×32 cells with two starting and landing zones in the top left and bottom right corners. Besides regular building patterns, some irregularly shaped buildings and additional NFZs are present. The 'Urban50' scenario (Fig. 2b and 2d) contains 50×50 cells and one starting and landing area around the center building. Buildings are generally larger and spaced out, and an additional large NFZ is present on the bottom of the map. Note that the number of cells in the 'Urban50' map is roughly one magnitude larger than in the previous works [1] and [2]. The cell size for the scenarios is $10\text{m} \times 10\text{m}$ with Table II providing a legend for the plots.

1) *Coverage Path Planning*: For the CPP problem, the UAV is flying at a constant altitude of 25m with a camera mounted underneath that has a field of view angle of 90° . Consequently, the UAV can cover an area of 5×5 cells simultaneously, as long as obstacles do not block line of sight. The target areas are generated by randomly sampling geometric shapes of different sizes and types and overlaying them, creating partially connected target zones. For evaluation, a traditional metric for the CPP problem is the path length. However, this metric only offers meaningful comparison when full coverage is possible. In this work, we investigate flight time constrained CPP, in which full coverage is often impossible. Therefore, the evaluation metrics used are the coverage ratio (CR), i.e. the ratio of covered target cells to the initial target cells at the end of the episode, and coverage ratio and landed (CRAL), which is zero if the UAV did not land successfully and equal to CR if it did. The benefits of the CRAL metric are that it combines the two goals, achieving high coverage and returning to the landing zone



(a) Movement 124/140, CR=0.94 (b) Movement 234/250, CR=0.94 (c) Movement 92/150, CR=0.99 (d) Movement 191/200, CR=1.0
 Fig. 2: Example trajectories from the Monte Carlo simulations for CPP (a)+(b) and DH (c)+(d) on 32×32 Manhattan map (a)+(c) and 50×50 Urban map (b)+(d).

	Symbol	Description
DQN Input		Start and landing zone
		Regulatory no-fly zone (NFZ)
		Buildings blocking wireless links and FoV
		DH: IoT device
		CPP: Remaining target zone (yellow also NFZ)
Visualization		DH: Summation of building shadows
		DH: Movement while comm. with green device
		DH: Hovering while comm. with green device
		CPP: Not covered and covered
		Starting and landing positions during an episode
		Actions without comm.

TABLE II: Legend for scenario plots, **DH** and **CPP** are only applicable in data harvesting and coverage path planning scenarios, respectively.

within the flight time constraint. By normalizing performance to a value in $[0, 1]$, it enables performance comparisons over the changing scenarios with randomly generated target zones.

2) *Data Harvesting*: In the DH problem, the UAV is flying at a constant altitude of 10m communicating with devices on ground level. The achievable data rate is calculated based on distance, random shadow fading, and line-of-sight condition with the same communication channel parameters used in [2]. As in CPP, the path length is not an applicable metric. It is impossible to collect all data in all scenarios depending on the randomly changing locations of IoT devices, data amount, and maximum flying time. Therefore, the evaluation metric used is the collection ratio (CR), describing the ratio of collected data from all devices to the initially available data summed over all devices. Like in CPP, we also use collection ratio and landed (CRAL) in this context, showing the full data collection and landing performance in one normalized metric.

B. General Evaluation

The CPP agents were trained on target zones containing 3-8 shapes covering 20-50% of the available area. The movement range was set to 50-150 steps for the 'Manhattan32' scenario and 150-250 for the 'Urban50' scenario. For the DH scenarios, 3-10 devices are placed randomly in free cells and

Metric	Manhattan32 CPP	Manhattan32 DH	Urban50 CPP	Urban50 DH
Landed	98.5%	98.2%	98.1%	99.5%
CR	71.0%	83.6%	81.5%	74.5%
CRAL	70.3%	82.5%	80.1%	74.2%

TABLE III: Performance metrics averaged over 1000 random scenario Monte Carlo iterations.

contain 5.0-20.0 data units. The movement range was set to 50-150 steps for the 'Manhattan32' scenario and 100-200 for the 'Urban50' scenario. Four scenarios are evaluated in detail.

In the CPP scenarios, the agents in Fig. 2a and 2b show that they can find trajectories to cover most of the target area. Even the area in the NFZs is mostly covered. It can be seen that small areas that would require a detour are ignored, leading to incomplete coverage. However, most of the target area is covered efficiently.

The agents in the DH scenarios in Fig. 2c and 2d perform very well. In the 'Manhattan32' scenario, the agent leaves small amounts of data at the orange and purple devices totaling a collection ratio of 99.1%. However, the agent finds a concise path, using only 92 of the allowed 150 movement steps. In the 'Urban50' scenario, the agent manages to collect all the data and return with some movement steps in spare.

All four agents were trained for 2 million steps. When analyzing their performance in all four missions using 1000 Monte Carlo generated scenarios (see Table III), it can be seen that all agents' landing performances are good, with the 'Urban50' DH agent being slightly better.

C. Global-Local Parameter Evaluation

To establish the performance sensitivity to the new hyper-parameters, global map scaling g , and local map size l , we trained multiple agents with different parameters on the CPP and DH problems. We chose four values for l and four for g and trained three agents for each possible combination. Additionally, we trained three agents without the usage of global and local map processing, which is equivalent to setting $g = 1$ and $l = 0$. The resulting 51 agents for the

Global map scaling g	Local map scaling l			
	9	17	25	33
2	8,481	9,761	13,089	18,465
3	2,721	4,001	7,329	12,705
5	273	1,553	4,881	10,257
7	33	1,313	4,641	10,017

TABLE IV: Flatten layer size for 'Manhattan32' with different global map scaling and local map sizes; Without global-local map processing the size is 48,401 neurons.

Global map scaling g	Local map scaling l							
	9		17		25		33	
	CPP	DH	CPP	DH	CPP	DH	CPP	DH
2	2.7	2.2	2.3	2.0	1.8	1.6	1.3	1.1
3	3.5	3.0	3.0	2.5	2.2	1.9	1.6	1.4
5	4.2	3.6	3.4	3.0	2.5	2.2	1.9	1.6
7	4.7	3.8	3.6	3.0	2.5	2.2	2.5	2.1

TABLE V: Training time speedup for the CPP and DH problem relative to without global-local map processing.

CPP and DH problems were trained for 500k steps each and evaluated on 200 Monte Carlo generated scenarios. The difference to the previous evaluation is that the movement budget range was set to 150 – 300.

Table IV shows the selected parameters and the resulting flatten layer size according to (19). A significant speedup of the training process compared to agents without global and local map processing can be observed in Table V.

The resulting CRAL values from the Monte Carlo simulations for each agent with respect to the agent's flatten layer size are shown in Fig. 3a and 3b for the CPP and DH problem, respectively. It can be seen that the DH problem is more sensitive to the parameters than the CPP problem. Generally, a larger flatten layer yields better performance up to a point. For both problems, it can be seen that a large flatten layer can cause the learning to get unstable, resulting in a CRAL of zero for some runs. This is caused by the agent's failure to learn how to land. The DH agents, which are not using the global-local map approach, never learn how to land reliably and thus have a CRAL score near zero.

In both cases, the agents with $l = 17$ and $g = 3$ or $g = 5$ show the best performance with respect to their flatten layer size, justifying the selection in Table I. Besides these two parameter combinations, it is noteworthy that the agents with $l = 9$ and $g = 7$ also perform well in both scenarios, despite their small flatten layer size of only 33 neurons.

V. CONCLUSION

We have presented a method for generalizing autonomous UAV path planning over two distinctly different mission types, coverage path planning and data harvesting. Through the flexibility afforded by combining specific mission goals and navigation constraints in the reward function, we trained DDQNs with identical structures in both scenarios to make

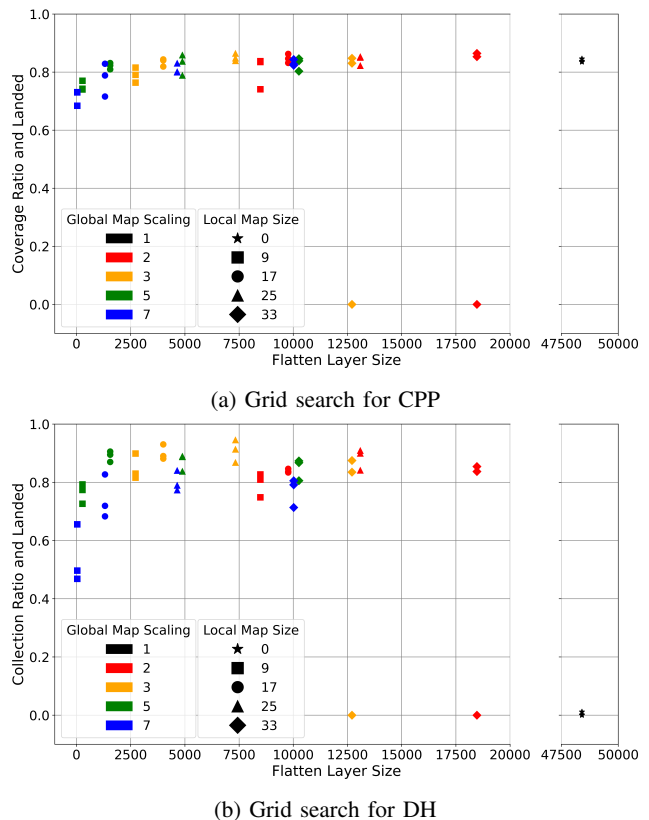


Fig. 3: Parameter grid search for CPP and DH with parameters from Table IV; the black stars correspond to agents without global-local map processing.

efficient movement decisions. We have introduced a novel global-local map processing scheme that allows to feed large maps directly into convolutional layers of the DRL agent and analyzed the effects of map processing parameters on learning performance. In future work, we will investigate still existing hindrances for applying our method to even larger maps, namely avoiding small-scale decision alternation through the use of macro-actions or options [17]. Combining the presented high-level path planning approach with a low-level flight dynamics controller will also make it possible to conduct experiments with realistic open-source UAV simulators in the future. Additionally, we will investigate the effect of irregularly shaped, non-convex obstacles on the path planning performance.

ACKNOWLEDGMENTS

Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research. Harald Bayerlein and David Gesbert were partially supported by the French government, through the 3IA Côte d'Azur project number ANR-19-P3IA-0002, as well as by the TSN CARNOT Institute under project Robots4IoT.

REFERENCES

- [1] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, "UAV coverage path planning under varying power constraints using deep reinforcement learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.
- [2] H. Bayerlein, M. Theile, M. Caccamo, and D. Gesbert, "UAV path planning for wireless data harvesting: A deep reinforcement learning approach," in *IEEE Global Communications Conference (GLOBECOM)*, 2020.
- [3] Y. Zeng, Q. Wu, and R. Zhang, "Accessing from the sky: A tutorial on UAV communications for 5G and beyond," *Proceedings of the IEEE*, vol. 107, no. 12, pp. 2327–2375, 2019.
- [4] R. Shakeri, M. A. Al-Garadi, A. Badawy, A. Mohamed, T. Khattab, A. K. Al-Ali, K. A. Harras, and M. Guizani, "Design challenges of multi-UAV systems in cyber-physical applications: A comprehensive survey and future directions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3340–3385, 2019.
- [5] T. Cabreira, L. Brisolaro, and P. R. Ferreira, "Survey on coverage path planning with unmanned aerial vehicles," *Drones*, vol. 3, no. 1, 2019.
- [6] C. Piciarelli and G. L. Foresti, "Drone patrolling with reinforcement learning," in *Proceedings of the 13th International Conference on Distributed Smart Cameras*, ACM, 2019.
- [7] K. D. Julian and M. J. Kochenderfer, "Distributed wildfire surveillance with autonomous aircraft using deep reinforcement learning," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 8, pp. 1768–1778, 2019.
- [8] E. Seraj and M. Gombolay, "Coordinated control of UAVs for human-centered active sensing of wildfires," in *American Control Conference (ACC)*, pp. 1845–1852, IEEE, 2020.
- [9] D. Baldazo, J. Parras, and S. Zazo, "Decentralized multi-agent deep reinforcement learning in swarms of drones for flood monitoring," in *27th European Signal Processing Conference (EUSIPCO)*, IEEE, 2019.
- [10] O. Esrafilian, R. Gangula, and D. Gesbert, "Learning to communicate in UAV-aided wireless networks: Map-based approaches," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1791–1802, 2018.
- [11] C. H. Liu, Z. Dai, Y. Zhao, J. Crowcroft, D. O. Wu, and K. Leung, "Distributed and energy-efficient mobile crowdsensing with charging stations by deep reinforcement learning," *IEEE Transactions on Mobile Computing*, vol. 20, no. 1, pp. 130–146, 2021.
- [12] S. Zhang and R. Zhang, "Radio map based path planning for cellular-connected UAV," in *IEEE Global Communications Conference (GLOBECOM)*, 2019.
- [13] J. Xie, L. R. G. Carrillo, and L. Jin, "An integrated traveling salesman and coverage path planning problem for unmanned aircraft systems," *IEEE Control Systems Letters*, vol. 3, no. 1, pp. 67–72, 2019.
- [14] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.
- [15] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Thirtieth AAAI conference on artificial intelligence*, pp. 2094–2100, 2016.
- [16] S. Zhang and R. S. Sutton, "A deeper look at experience replay," *arXiv:1712.01275 [cs.LG]*, 2017.
- [17] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

4.4 Multi-UAV Path Planning for Wireless Data Harvesting With Deep Reinforcement Learning

Reference

H. Bayerlein, M. Theile, M. Caccamo, and D. Gesbert, “Multi-UAV path planning for wireless data harvesting with deep reinforcement learning,” *IEEE Open Journal of the Communications Society*, vol. 2, pp. 1171–1187, 2021

DOI: <https://doi.org/10.1109/OJCOMS.2021.3081996>

Abstract

Harvesting data from distributed Internet of Things (IoT) devices with multiple autonomous unmanned aerial vehicles (UAVs) is a challenging problem requiring flexible path planning methods. We propose a multi-agent reinforcement learning (MARL) approach that, in contrast to previous work, can adapt to profound changes in the scenario parameters defining the data harvesting mission, such as the number of deployed UAVs, number, position and data amount of IoT devices, or the maximum flying time, without the need to perform expensive recomputations or relearn control policies. We formulate the path planning problem for a cooperative, non-communicating, and homogeneous team of UAVs tasked with maximizing collected data from distributed IoT sensor nodes subject to flying time and collision avoidance constraints. The path planning problem is translated into a decentralized partially observable Markov decision process (Dec-POMDP), which we solve through a deep reinforcement learning (DRL) approach, approximating the optimal UAV control policy without prior knowledge of the challenging wireless channel characteristics in dense urban environments. By exploiting a combination of centered global and local map representations of the environment that are fed into convolutional layers of the agents, we show that our proposed network architecture enables the agents to cooperate effectively by carefully dividing the data collection task among themselves, adapt to large complex environments and state spaces, and make movement decisions that balance data collection goals, flight-time efficiency, and navigation constraints. Finally, learning a control policy that generalizes over the scenario parameter space enables us to analyze the influence of individual parameters on collection performance and provide some intuition about system-level benefits.

Contributions to this paper

- Shared conceptualization of the multi-UAV data harvesting problem
- Shared adaptation of the code base for multi-agent
- Share of paper writing

Copyright

Creative Commons License – CC BY 4.0 DEED

<https://creativecommons.org/licenses/by/4.0/>

See Appendix A.6 for the reuse statement. The following shows the accepted version.

Multi-UAV Path Planning for Wireless Data Harvesting with Deep Reinforcement Learning

Harald Bayerlein, *Student Member, IEEE*, Mirco Theile, *Student Member, IEEE*,
Marco Caccamo, *Fellow, IEEE*, and David Gesbert, *Fellow, IEEE*

Harvesting data from distributed Internet of Things (IoT) devices with multiple autonomous unmanned aerial vehicles (UAVs) is a challenging problem requiring flexible path planning methods. We propose a multi-agent reinforcement learning (MARL) approach that, in contrast to previous work, can adapt to profound changes in the scenario parameters defining the data harvesting mission, such as the number of deployed UAVs, number, position and data amount of IoT devices, or the maximum flying time, without the need to perform expensive recomputations or relearn control policies. We formulate the path planning problem for a cooperative, non-communicating, and homogeneous team of UAVs tasked with maximizing collected data from distributed IoT sensor nodes subject to flying time and collision avoidance constraints. The path planning problem is translated into a decentralized partially observable Markov decision process (Dec-POMDP), which we solve through a deep reinforcement learning (DRL) approach, approximating the optimal UAV control policy without prior knowledge of the challenging wireless channel characteristics in dense urban environments. By exploiting a combination of centered global and local map representations of the environment that are fed into convolutional layers of the agents, we show that our proposed network architecture enables the agents to cooperate effectively by carefully dividing the data collection task among themselves, adapt to large complex environments and state spaces, and make movement decisions that balance data collection goals, flight-time efficiency, and navigation constraints. Finally, learning a control policy that generalizes over the scenario parameter space enables us to analyze the influence of individual parameters on collection performance and provide some intuition about system-level benefits.

Index Terms—Internet of Things (IoT), map-based planning, multi-agent reinforcement learning (MARL), trajectory planning, unmanned aerial vehicle (UAV).

I. INTRODUCTION

Autonomous unmanned aerial vehicles (UAVs) are not only envisioned as passive cellular-connected users of telecommunication networks but also as active connectivity enablers [2]. Their fast and flexible deployment makes them especially useful in situations where terrestrial infrastructure is overwhelmed or destroyed, e.g. in disaster and search-and-rescue situations [3], or where fixed coverage is in any way lacking. UAVs have shown particular promise in collecting data from distributed Internet of Things (IoT) sensor nodes. For instance, IoT operators can deploy UAV data harvesters in the absence of otherwise expensive cellular infrastructure nearby. Another reason is the throughput efficiency benefits related to having UAVs that describe a flight pattern that brings them close to the IoT devices. As an example in the context of infrastructure maintenance and preserving structural integrity, Hitachi is already commercially deploying partially autonomous UAVs that collect data from IoT sensors embedded in large structures, such as the San Juanico and Agas-Agas Bridges in the Philippines [4]. Research into UAV-aided data collection from IoT devices or wireless sensors include the

works [5]–[9], with [10]–[13] concentrating on minimizing the age of information of the collected data. Additional coverage of past related work is offered in the next section.

In this work, we focus on controlling a team of UAVs, consisting of a variable number of identical drones tasked with collecting varying amounts of data from a variable number of stationary IoT sensor devices at variable locations in an urban environment. This imposes challenging constraints on the trajectory design for autonomous UAVs. In addition, the limited on-board battery energy density restricts mission duration for quadcopter drones severely. At the same time, the complex urban environment poses challenges in obstacle avoidance and adherence to regulatory no-fly zones (NFZs). Additionally, the wireless communication channel is characterized by random signal blocking events due to alternating between line-of-sight (LoS) and non-line-of-sight (NLoS) links. We believe this work is the first to address multi-UAV path planning where learned control policies are generalized over a wide scenario parameter space and can be directly applied when scenario parameters change without the need for retraining.

While some challenges to real-world deep reinforcement learning (DRL) such as limited training samples, safety and lack of explainable actions remain, DRL offers the opportunity to balance challenges and data collection goals for complex environments in a straightforward way by combining them in the reward function. Another reason for the popularity of the DRL paradigm in this context is the computational efficiency of DRL inference. DRL is also one of the few methods that allows us to tackle the complex task directly, given that UAV control and deployment in communication scenarios are generally non-convex optimization problems [2], [14]–[18], and proven to be NP-hard in many instances [2], [16], [17]. These advantages of DRL also hold for other UAV

H. Bayerlein and D. Gesbert were partially supported by the French government, through the 3IA Côte d’Azur project number ANR-19-P3IA-0002, as well as by the TSN CARNOT Institute under project Robots4IoT. M. Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research. This article was presented in part at IEEE GLOBECOM 2020 [1]. The code for this work is available under https://github.com/hbayerlein/uav_data_harvesting. (*Corresponding author: Harald Bayerlein*)

H. Bayerlein and D. Gesbert are with the Communication Systems Department, EURECOM, Sophia Antipolis, France, {harald.bayerlein, david.gesbert}@eurecom.fr.

M. Theile and M. Caccamo are with the TUM Department of Mechanical Engineering, Technical University of Munich, Germany, {mirco.theile, mcaccamo}@tum.de.

path planning instances, such as coverage path planning [19], a classical robotics problem where the UAV's goal is to cover all points inside an area of interest. The equivalence of these path planning problems and the connection between the often disjoint research areas is highlighted in [20].

A. Related Work

A survey that spans the various application areas for multi-UAV systems from a cyber-physical perspective is provided in [17]. The general challenges and opportunities of UAV communications are summarized in publications by Zeng *et al.* [2] and Saad *et al.* [18], which both include data collection from IoT devices. This specific scenario is also included in [21] and [22], surveys that comprise information on the classification of UAV communication applications with a focus on DRL methods.

Path planning for UAVs providing some form of communication services or collecting data has been studied extensively, including numerous approaches based on reinforcement learning (RL). However, it is crucial to note that the majority of previous works concentrates on only finding the optimal trajectory solution for one set of scenario parameters at a time, requiring full or partial retraining if the scenario changes. In contrast, our approach aims to train and generalize over a large scenario parameter space directly, finding efficient solutions without the need for lengthy retraining, but also increasing the complexity of the path planning problem significantly.

Many existing RL approaches also only focus on single-UAV scenarios. An early proposal given in [23] to use (deep) RL in a related scenario where a single UAV base station serves ground users shows the advantages of using a deep Q-network (DQN) over table-based Q-learning, while not making any explicit assumptions about the environment at the price of long training time. The authors in [5] only investigate table-based Q-learning for UAV data collection. A particular variety of IoT data collection is the one tackled in [10], where the authors propose a DQN-based solution to minimize the age of information of data collected from sensors. In contrast to our approach, the mentioned approaches are set in much simpler environments and agents have to undergo computationally expensive retraining when scenario parameters change.

Multi-UAV path planning for serving ground users employing table-based Q-learning is investigated in [16], based on a relatively complex 3-step algorithm consisting of grouping the users with a genetic algorithm, then deployment and movement design in two separated instances of Q-learning. The investigated optimization problem is proven to be NP-hard, with Q-learning being confirmed as a useful tool to solve it. Pan *et al.* [6] investigate an instance of multi-UAV data collection from sensor nodes formulated as a classical traveling salesman problem without modeling the communication phase between UAV and node. The UAVs' trajectories are designed with a genetic algorithm that uses some aspects of DRL, namely training a deep neural network and experience replay. In contrast to the multi-stage optimization algorithms in [16] and [6], our approach consists of a more straightforward end-to-end DRL approach that scales to large and complex environments, generalizing over varying scenario parameters.

The combination of DRL and multi-UAV control has been studied previously in various scenarios. The authors in [11] focus on trajectory design for minimizing the age of information of sensing data generated by multiple UAVs themselves where the data can be either transmitted to terrestrial base stations or mobile cellular devices. Their focus lies on balancing the UAV sensing and transmission protocol in an unobstructed environment for one set of scenario parameters at a time. Other MARL path planning approaches to minimize the age of information of collected data include [12] and [13]. In [24], a swarm of UAVs on a target detection and tracking mission in an unknown environment is controlled through a distributed DQN approach. While the authors also use convolutional processing to feed map information to the agents, the map is initially unknown and has to be explored to detect the targets. The agents' goal is to learn transferable knowledge that enables adaptation to new scenarios with fast relearning, compared to our approach to learn a control policy that generalizes over scenario parameters and requires no relearning.

Hu *et al.* [14] proposed a distributed multi-UAV meta-learning approach to control a group of drone base stations serving ground users with random uplink access demands. While meta-learning allows them to reduce the number of training episodes needed to adapt to a new unseen uplink demand scenario, several hundred are still required. Our approach focuses on training directly on random but observable scenario parameters within a given value range, therefore not requiring retraining to adapt. Due to the small and obstructionless environment, no maps are required in [14] and navigation constraints are omitted by keeping the UAVs at dedicated altitudes. In [7], multi-agent deep Q-learning is used to optimize trajectories and resource assignment of UAVs that collect data from pre-defined clusters of IoT devices and provide power wirelessly to them. The focus here is on maximizing minimum throughput in a wirelessly powered network without a complex environment and navigation constraints, only for a single scenario at a time. Similarly, in [8] there is also a strong focus on the energy supply of IoT devices through backscatter communications when a team of UAVs collects their data. The authors propose a multi-agent approach that relies on the definition of ambiguous boundaries between clusters of sensors. The scenario is set in a simple, unobstructed environment, not requiring maps or adherence to multiple navigation constraints, but requiring retraining when scenario parameters change.

In [25], a group of interconnected UAVs is tasked with providing long-term communication coverage to ground users cooperatively. While the authors also formulate a POMDP that they solve by a DRL variant, there is no need for map information or processing. The scenario is set in a simple environment without obstacles or other navigation constraints. This work was extended under the paradigm of mobile crowdsensing, where mobile devices are leveraged to collect data of common interest in [9]. The authors proposed a heterogeneous multi-agent DRL algorithm collecting data simultaneously with ground and aerial vehicles in an environment with obstacles and charging stations. While in this work, the authors also suggest a convolutional neural network to exploit a map of the environment, the small grid world does not necessitate

extensive map processing. Furthermore, they do not center the map on the agent's position, which is highly beneficial [1]. In contrast to our method, control policies have to be relearned entirely in a lengthy training process for both mentioned approaches when scenario and environmental parameters change.

B. Contributions

If DRL methods are to be applied in any real-world mission, the prohibitively high training data demand poses one of the most severe challenges [26]. This is exacerbated by the fact that even minor changes in the scenario, such as in the number or location of sensor devices in data collection missions, typically requires repeating the full training procedure of the DRL agent. This is the case for existing approaches such as [7]–[9], [11]–[13], [23], [25]. Other approaches to reduce the training data demand include meta-learning [14] and transfer learning [26]. To the best of our knowledge, this is the first work that addresses this problem in path planning for multi-UAV data harvesting by proposing a DRL method that is able to generalize over a large space of scenario parameters in complex urban environments without prior knowledge of wireless channel characteristics based on centered global-local map processing.

The main contributions of this paper are the following:

- We formulate a flying time constrained multi-UAV path planning problem to maximize harvested data from IoT sensors. We consider its translation to a decentralized partially observable Markov decision process (Dec-POMDP) with full reward function description in large, complex, and realistic environments that include no-fly zones, buildings that block wireless links (some possible to be flown over, some not), and dedicated start/landing zones.
- To solve the Dec-POMDP under navigation constraints without any prior knowledge of the urban environment's challenging wireless propagation conditions, we employ deep multi-agent reinforcement learning with centralized learning and decentralized execution.
- We show the advantage in learning and adaptation efficiency to large maps and state spaces through a dual global-local map approach with map centering over more conventional scalar neural network input in a multi-UAV setting.
- As perhaps our most salient feature, our algorithm offers *parameter generalization*, which means that the learned control policy can be reused over a wide array of scenario parameters, including the number of deployed UAVs, variable start positions, maximum flying times, and number, location and data amount of IoT sensor devices, without the need to restart the training procedure as typically required by existing DRL approaches.
- Learning a generalized control policy enables us to compare performance over a large scenario parameter space directly. We analyze the influence of individual parameters on collection performance and provide some intuition about system-level benefits.

C. Organization

The paper is organized as follows: Section II introduces the multi-UAV mobility and communication channel model, which is translated to an MDP in Section III and followed by a description of the proposed map preprocessing in Section IV and multi-agent DRL learning approach in Section V. Simulation results and their discussion follow in Section VI, and we conclude the paper with a summary and outlook to future work in Section VII.

II. SYSTEM MODEL

In the following, we present the key models for the multi-UAV path planning problem. Note that some level of simplification is needed when modeling the robots' dynamics in order to enable the implementation of the RL approach. Our assumptions are explicit whenever suitable.

We consider a square grid world of size $M \times M \in \mathbb{N}^2$ with cell size c and the set of all possible positions \mathcal{M} . Discretization of the environment is a necessary condition for our map-processing approach, however note that our method can be applied to any rectangular grid world. The environment contains L designated start/landing positions given by the set

$$\mathcal{L} = \left\{ [x_i^l, y_i^l]^T, i = 1, \dots, L, : [x_i^l, y_i^l]^T \in \mathcal{M} \right\}$$

and the combination of the Z positions the UAVs cannot occupy is given by the set

$$\mathcal{Z} = \left\{ [x_i^z, y_i^z]^T, i = 1, \dots, Z, : [x_i^z, y_i^z]^T \in \mathcal{M} \right\}.$$

This includes tall buildings which the UAVs can not fly over and regulatory no-fly zones (NFZ). The number of B obstacles blocking wireless links are given by the set

$$\mathcal{B} = \left\{ [x_i^b, y_i^b]^T, i = 1, \dots, B, : [x_i^b, y_i^b]^T \in \mathcal{M} \right\},$$

representing all buildings, also smaller ones that can be flown over. The lowercase letters l, z, b indicate the coordinates of the respective set of environmental features $\mathcal{L}, \mathcal{Z}, \mathcal{B}$. An example of a grid world is depicted in Fig. 1, where obstacles, NFZs, start/landing zone, and an example of a single UAV trajectory are marked as described in the attached legend in Tab. I.

A. UAV Model

The set \mathcal{I} of I deployed UAVs moves within the limits of the grid world \mathcal{M} . The state of the i -th UAV is described through its:

- position $\mathbf{p}_i(t) = [x_i(t), y_i(t), z_i(t)]^T \in \mathbb{R}^3$ with altitude $z_i(t) \in \{0, h\}$, either at ground level or in constant altitude h ;
- operational status $\phi_i(t) \in \{0, 1\}$, either inactive or active;
- battery energy level $b_i(t) \in \mathbb{N}$.

Note that the assumption of all UAVs sharing the same flying altitude is not too restrictive and that our method allows each UAV to fly at a different altitude as long as it remains constant throughout the mission. The UAV agent's altitude can be made observable by simply adding it to the observation space

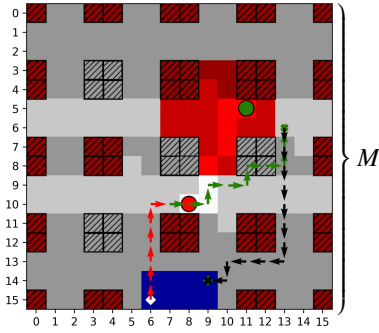


Fig. 1. Example of a single UAV collecting data from two IoT devices in an urban environment of size $M \times M$ with NFZs, a single start/landing zone, and buildings causing shadowing. Small buildings can be flown over and tall buildings act as navigation obstacles.

	Symbol	Description
DQN Input		Start and landing zone
		Regulatory no-fly zone (NFZ)
		Tall buildings* (UAVs cannot fly over)
		Small buildings* (UAVs can fly over)
		IoT device
		Other agents
		*all buildings obstruct wireless links
Visualization		Summation of building shadows
		Starting and landing positions during an episode
		UAV movement while comm. with green device
		Hovering while comm. with green device
		Actions without comm. (all data collected)

TABLE I
LEGEND FOR SCENARIO PLOTS.

along the flying time. This work only tackles 2D trajectory optimization, as the environment is dominated by high-rise buildings that would require long climbing phases to be overflown. The mission time limited by the UAVs' on-board batteries restricts the effectiveness of 3D control for the data collection performance given that climbing flight consumes more energy [27] and that the UAVs need to land at ground level at the end of the mission. The data collection mission is over after $T \in \mathbb{N}$ mission time steps for all UAVs, where the time horizon is discretized into equal mission time slots $t \in [0, T]$ of length δ_t seconds.

The action space of each UAV is defined as

$$\mathcal{A} = \left\{ \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}_{\text{hover}}, \underbrace{\begin{bmatrix} c \\ 0 \\ 0 \end{bmatrix}}_{\text{east}}, \underbrace{\begin{bmatrix} 0 \\ c \\ 0 \end{bmatrix}}_{\text{north}}, \underbrace{\begin{bmatrix} -c \\ 0 \\ 0 \end{bmatrix}}_{\text{west}}, \underbrace{\begin{bmatrix} 0 \\ -c \\ 0 \end{bmatrix}}_{\text{south}}, \underbrace{\begin{bmatrix} 0 \\ 0 \\ -h \end{bmatrix}}_{\text{land}} \right\}. \quad (1)$$

Each UAV's movement actions $\mathbf{a}_i(t) \in \tilde{\mathcal{A}}(\mathbf{p}_i(t))$ are limited to

$$\tilde{\mathcal{A}}(\mathbf{p}_i(t)) = \begin{cases} \mathcal{A}, & \mathbf{p}_i(t) \in \mathcal{L} \\ \mathcal{A} \setminus [0, 0, -h]^T, & \text{otherwise,} \end{cases} \quad (2)$$

where $\tilde{\mathcal{A}}$ defines the set of feasible actions depending on the respective UAV's position, specifically that the landing action is only allowed if the UAV is in the landing zone.

The distance the UAV travels within one time slot is equivalent to the cell size c . Mission time slots are chosen sufficiently small so that each UAV's velocity $v_i(t)$ can be considered to remain constant in one time slot. The UAVs are limited to moving with horizontal velocity $V = c/\delta_t$ or standing still, i.e. $v_i(t) \in \{0, V\}$ for all $t \in [0, T]$. Each UAV's position evolves according to the motion model given by

$$\mathbf{p}_i(t+1) = \begin{cases} \mathbf{p}_i(t) + \mathbf{a}_i(t), & \phi_i(t) = 1 \\ \mathbf{p}_i(t), & \text{otherwise,} \end{cases} \quad (3)$$

keeping the UAV stationary if inactive. The evolution of the operational status $\phi_i(t)$ of each UAV is given by

$$\phi_i(t+1) = \begin{cases} 0, & \mathbf{a}_i(t) = [0, 0, -h]^T \\ \vee \phi_i(t) = 0 \\ 1, & \text{otherwise,} \end{cases} \quad (4)$$

where the operational status becomes inactive when the UAV has safely landed. The end of the data harvesting mission T is defined as the time slot when all UAVs have reached their terminal state and are not actively operating anymore, i.e. the operational state is $\phi_i(t) = 0$ for all UAVs.

The i -th UAV's battery content evolves according to

$$b_i(t+1) = \begin{cases} b_i(t) - 1, & \phi_i(t) = 1 \\ b_i(t), & \text{otherwise,} \end{cases} \quad (5)$$

assuming a constant energy consumption while the UAV is operating and zero energy consumption when operation has terminated. This is a simplification justified by the fact that power consumption for small quadcopter UAVs is dominated by the hovering component. Using the model from [27], the ratio between the additional power necessary for horizontal flight at 10m/s and just hovering could be roughly estimated as $30W/310W \approx 10\%$, which is negligible. Considering power consumption of on-board computation and communication hardware which does not differ between flight and hovering, the overall difference becomes even smaller. In the following, we will refer to the battery content as remaining flying time, as it is directly equivalent.

The overall multi-UAV mobility model is restricted by the following constraints:

$$\mathbf{p}_i(t) \neq \mathbf{p}_j(t) \vee \phi_j(t) = 0, \quad \forall i, j \in \mathcal{I}, i \neq j, \forall t \quad (6a)$$

$$\mathbf{p}_i(t) \notin \mathcal{Z}, \quad \forall i \in \mathcal{I}, \forall t \quad (6b)$$

$$b_i(t) \geq 0, \quad \forall i \in \mathcal{I}, \forall t \quad (6c)$$

$$\mathbf{p}_i(0) \in \mathcal{L} \wedge z_i(0) = h, \quad \forall i \in \mathcal{I} \quad (6d)$$

$$\phi_i(0) = 1, \quad \forall i \in \mathcal{I} \quad (6e)$$

The constraint (6a) describes collision avoidance among active UAVs with the exception that UAVs can land at the same location. (6b) forces the UAVs to avoid collisions with tall obstacles and prevents them from entering NFZs. The constraint (6c) limits operation time of the drones, forcing UAVs to end their mission before their battery has run out. Since operation can only be concluded with the landing action as described in (4) and the landing action is only available in the landing zone as defined in (2), the constraint (6c) ensures

that each UAV safely lands in the landing zone before their batteries are empty. The starting constraint (6d) defines that the UAV start positions are in the start/landing zones and that their starting altitude is h , while (6e) ensures that the UAVs start in the active operational state.

B. Communication Channel Model

1) Link Performance Model

As communication systems typically operate on a smaller timescale than the UAVs' mission planning system, we introduce the notion of communication time slots in addition to mission time slots. We partition each mission time slot $t \in [0, T]$ into a number of $\lambda \in \mathbb{N}$ communication time slots. The communication time index is then $n \in [0, N]$ with $N = \lambda T$. One communication time slot n is of length $\delta_n = \delta_t / \lambda$ seconds. The number of communication time slots per mission time slot λ is chosen sufficiently large so that the i -th UAV's position, which is interpolated linearly between $\mathbf{p}_i(t)$ and $\mathbf{p}_i(t+1)$, and the channel gain can be considered to stay constant within one communication time slot.

The k -th IoT device is located on ground level at $\mathbf{u}_k = [x_k, y_k, 0]^T \in \mathbb{R}^3$ with $k \in \mathcal{K}$ where $|\mathcal{K}| = K$. Each IoT sensor has a finite amount of data $D_k(t) \in \mathbb{R}^+$ that needs to be picked up over the whole mission time $t \in [0, T]$. The device data volume is set to an initial value at the start of the mission $D_k(t=0) = D_{k,init}$. The data volume of each IoT node evolves depending on the communication time index n over the whole mission time, given by $D_k(n)$ with $n \in [0, N]$, $N = \lambda T$.

We follow the same UAV-to-ground channel model as used in [1]. The communication links between UAVs and the K IoT devices are modeled as LoS/NLoS point-to-point channels with log-distance path loss and shadow fading. The maximum achievable information rate at time n for the k -th device is given by

$$R_{i,k}^{\max}(n) = \log_2(1 + \text{SNR}_{i,k}(n)). \quad (7)$$

Considering the amount of data available at the k -th device $D_k(n)$, the effective information rate is given as

$$R_{i,k}(n) = \begin{cases} R_{i,k}^{\max}(n), & D_k(n) \geq \delta_n R_{i,k}^{\max}(n) \\ D_k(n) / \delta_n, & \text{otherwise.} \end{cases} \quad (8)$$

The SNR with transmit power $P_{i,k}$, white Gaussian noise power at the receiver σ^2 , UAV-device distance $d_{i,k}$, path loss exponent α_e and $\eta_e \sim \mathcal{N}(0, \sigma_e^2)$ modeled as a Gaussian random variable, is defined as

$$\text{SNR}_{i,k}(n) = \frac{P_{i,k}}{\sigma^2} \cdot d_{i,k}(n)^{-\alpha_e} \cdot 10^{\eta_e/10}. \quad (9)$$

Note that the urban environment with the set of obstacles \mathcal{B} hindering free propagation causes a strong dependence of the propagation parameters on the $e \in \{\text{LoS}, \text{NLoS}\}$ condition and that (9) is the SNR averaged over small scale fading. We would also like to point out that our DQN-based trajectory planning approach is model-free and does therefore not rely on any specific channel model. While a more accurate and complex model could be directly used with our approach, the most important features for data collection missions of the

urban channel, the dependence of SNR on $d_{i,k}$ and the $e \in \{\text{LoS}, \text{NLoS}\}$ condition, are already captured in (9).

2) Multiple Access Protocol

The multiple access protocol is assumed to follow the standard time-division multiple access (TDMA) model when it comes to the communication between one single UAV and the various ground nodes. We further assume that the communication channel between the ground nodes and a given UAV operates on resource blocks (time-frequency slots) that are orthogonal to the channels linking the ground nodes and other UAVs, so that no inter-UAV interference exists in our model and inter-UAV synchronization is not necessary. Hence, the UAVs are similar to base stations that would be assigned orthogonal spectral resources. We also assume that IoT devices are operating in multi-band mode, hence are capable of simultaneously communicating with all UAVs on the set of all orthogonal frequencies. As a consequence, scheduling decisions are not part of the action space. The number of available orthogonal subchannels for UAV-to-ground communication is one of the variable scenario parameters and equivalent to the number of deployed UAVs.

Designing multiple access protocols for UAV networks is in itself a challenging research problem [28] due to high mobility of the nodes and fast changing link performance and is out of scope for this work. However, our proposed algorithm can in principle be integrated with existing solutions and does not rely on any specific channel model or multiple access protocol. While our model avoids and does not consider inter-UAV interference, we would like to point out that the behavior of the UAV agents that emerges naturally during the learning process of dividing the data collection task geographically, as illustrated in section VI-D, would mitigate the influence of interference on the trajectory planning decisions to some extent.

Our scheduling protocol is assumed to follow the max-rate rule: in each communication time slot $n \in [0, N]$, the sensor node $k \in [1, K]$ with the highest $\text{SNR}_{i,k}(n)$ with remaining data to be uploaded is picked by the scheduling algorithm. The TDMA constraint for the scheduling variable $q_{i,k}(n) \in \{0, 1\}$ is given by

$$\sum_{k=1}^K q_{i,k}(n) \leq 1, \quad n \in [0, N], \forall i \in \mathcal{I}. \quad (10)$$

It follows that the k -th device's data volume evolves within one communication time slot according to

$$D_k(n+1) = D_k(n) - \sum_{i=1}^I q_{i,k}(n) R_{i,k}(n) \delta_n. \quad (11)$$

The achievable throughput for the i -th UAV for one mission time slot $t \in [0, T]$, comprised of λ communication time slots, is the sum of rates achieved in the communication time slots $n \in [\lambda t, \lambda(t+1) - 1]$ over K sensor nodes. It depends on the UAV's operational status $\phi_i(t)$ and is given by

$$C_i(t) = \phi_i(t) \sum_{n=\lambda t}^{\lambda(t+1)-1} \sum_{k=1}^K q_{i,k}(n) R_{i,k}(n) \delta_n. \quad (12)$$

C. Optimization Problem

Using the described UAV model in II-A and communication model in II-B, the central goal of the multi-UAV path planning problem is the maximization of throughput over the whole mission time and over all I deployed UAVs while adhering to mobility constraints (6a)-(6e) and the scheduling constraint (10). The maximization problem is given by

$$\max_{\times_i \mathbf{a}_i(t)} \sum_{t=0}^T \sum_{i=1}^I C_i(t). \quad (13)$$

$$\text{s.t. (6a), (6b), (6c), (6d), (6e), (10)}$$

optimizing over joint actions $\times_i \mathbf{a}_i(t)$.

III. MARKOV DECISION PROCESS (DEC-POMDP)

To address the aforementioned optimization problem, we translate it to a decentralized partially observable Markov decision process (Dec-POMDP) [29], which is defined through the tuple $(\mathcal{S}, \mathcal{A}_\times, P, R, \Omega_\times, \mathcal{O}, \gamma)$. In the Dec-POMDP, \mathcal{S} describes the state space, $\mathcal{A}_\times = \mathcal{A}^I$ the joint action space, and $P : \mathcal{S} \times \mathcal{A}_\times \times \mathcal{S} \mapsto \mathbb{R}$ the transition probability function. $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function mapping state, individual action, and next state to a real valued reward. The joint observation space is defined through $\Omega_\times = \Omega^I$ and $\mathcal{O} : \mathcal{S} \times \mathcal{I} \mapsto \Omega$ is the observation function mapping state and agents to one agent's individual observation. The discount factor $\gamma \in [0, 1]$ controls the importance of long vs. short term rewards.

A. State Space

The state space of the multi-agent data collection problem consists of the environment information, the state of the agents, and the state of the devices. It is given as

$$\begin{aligned} \mathcal{S} = & \underbrace{\mathcal{L}}_{\substack{\text{Landing} \\ \text{Zones}}} \times \underbrace{\mathcal{Z}}_{\text{NFZs}} \times \underbrace{\mathcal{B}}_{\text{Obstacles}} & \left. \vphantom{\mathcal{S}} \right\} \text{Environment} \\ & \times \underbrace{\mathbb{R}^{I \times 3}}_{\substack{\text{UAV} \\ \text{Positions}}} \times \underbrace{\mathbb{N}^I}_{\substack{\text{Flying} \\ \text{Times}}} \times \underbrace{\mathbb{B}^I}_{\substack{\text{Operational} \\ \text{Status}}} & \left. \vphantom{\mathcal{S}} \right\} \text{Agents} \quad (14) \\ & \times \underbrace{\mathbb{R}^{K \times 3}}_{\substack{\text{Device} \\ \text{Positions}}} \times \underbrace{\mathbb{R}^K}_{\substack{\text{Device} \\ \text{Data}}} & \left. \vphantom{\mathcal{S}} \right\} \text{Devices} \end{aligned}$$

in which the elements $s(t) \in \mathcal{S}$ are

$$s(t) = (\mathbf{M}, \{\mathbf{p}_i(t)\}, \{b_i(t)\}, \{\phi_i(t)\}, \{\mathbf{u}_k\}, \{D_k(t)\}), \quad (15)$$

$\forall i \in \mathcal{I}$ and $\forall k \in \mathcal{K}$, in which $\mathbf{M} \in \mathbb{B}^{M \times M \times 3}$ is the tensor representation of the set of start/landing zones \mathcal{L} , obstacles and NFZs \mathcal{Z} , and obstacles only \mathcal{B} . The other elements of the tuple define positions, remaining flying times, and operational status of all agents, as well as positions and available data volume of all IoT devices.

B. Safety Controller

To enforce the collision avoidance constraint (6a) and the NFZ and obstacle avoidance constraint (6b), a safety controller is introduced into the system. Additionally, the safety controller enforces the limited action space excluding the *landing* action when the respective agent is not in the landing zone as defined in (2). The safety controller evaluates the action $\mathbf{a}_i(t)$ of agent i and determines if it should be accepted or rejected. If rejected, the resulting safe action is the *hovering* action. The safe action $\mathbf{a}_{s,i}(t)$ is thus defined as

$$\mathbf{a}_{s,i}(t) = \begin{cases} [0, 0, 0]^T, & \mathbf{p}_i(t) + \mathbf{a}_i(t) \in \mathcal{Z} \\ \vee \mathbf{p}_i(t) + \mathbf{a}_i(t) = \mathbf{p}_j(t) \wedge \phi_j(t) = 1, \\ \quad \forall j, j \neq i \\ \vee \mathbf{a}_i(t) = [0, 0, -h]^T \wedge \mathbf{p}_i(t) \notin \mathcal{L} \\ \mathbf{a}_i(t), & \text{otherwise.} \end{cases} \quad (16)$$

Without path planning capabilities, the safety controller cannot enforce the flying time and safe landing constraint in (6c). Therefore, we relax the hard constraint on flight time by adding a high penalty on not landing in time instead. In the simulation, a crashed agent, i.e. an agent with $b_i(t) < 0$, is defined as not operational.

C. Reward Function

The reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ of the Dec-POMDP is comprised of the following elements:

$$r_i(t) = \alpha \sum_{k \in \mathcal{K}} (D_k(t+1) - D_k(t)) + \beta_i(t) + \gamma_i(t) + \epsilon. \quad (17)$$

The first term of the sum is a collective reward for the collected data from all devices by all agents within mission time slot t . It is parameterized through the data collection multiplier α . This is the only part of the reward function that is shared among all agents. The second addend is an individual penalty when the safety controller rejects an action and given through

$$\beta_i(t) = \begin{cases} \beta, & \mathbf{a}_i(t) \neq \mathbf{a}_{i,s}(t) \\ 0, & \text{otherwise.} \end{cases} \quad (18)$$

It is parameterized through the safety penalty β . The third term is the individual penalty for not landing in time given by

$$\gamma_i(t) = \begin{cases} \gamma, & b_i(t+1) = 0 \wedge \mathbf{p}_i(t+1) = [\cdot, \cdot, h]^T \\ 0, & \text{otherwise.} \end{cases} \quad (19)$$

and parameterized through the crashing penalty γ . The last term is a constant movement penalty parameterized through ϵ , which is supposed to incentivize the agents to reduce their flying time and prioritize efficient trajectories.

IV. MAP-PROCESSING AND OBSERVATION SPACE

To aid the agents in interpreting the large state space given in (14), we implement two map processing steps. The first is centering the map around the agent's position, shown in [1] to significantly improve the agent's learning performance. This benefit is a consequence of neurons in the layer after the convolutional layers (compare Fig. 3) corresponding to

features *relative* to the agent's position, rather than to *absolute* positions if the map is not centered. This is advantageous as one agent's actions are solely based on its relative position to features, e.g. its distance to sensor devices. The downside of this approach is that it increases the size of the maps and the observation space even further, therefore requiring larger networks with more trainable parameters.

The second map processing step is to present the centered map as a compressed global and uncompressed but cropped local map as previously evaluated in [20]. In path planning, as distant features lead to general direction decisions while close features lead to immediate actions such as collision avoidance, the level of detail passed to the agent for distant objects can be less than for close objects. The advantage is that the compression of the global map reduces the necessary neural network size considerably.

This reduction in network size directly translates to a reduction in computational load. Table II shows the number of floating point operations needed for each of the two maps under different map processing regimes as given by the TensorFlow graph profiler. Only centering increases the computational load considerably, as explained in [1], while global-local map processing offsets the increase and reduces floating point operations considerably. Considering that modern embedded processors operate in the region of giga floating point operations, it seems realistic that the required processing can be carried out even on small and energy-limited UAVs. The mathematical descriptions of the map processing functions and the observation space are detailed in the following.

Map	No Processing	Centering	Centering + Global-Local
Manhattan32	15	80	7.7
Urban50	45	217	6.5

TABLE II
MILLION FLOATING POINT OPERATIONS (MFLOPS) NEEDED FOR INFERENCE OF THE NETWORKS BASED ON MAP-PROCESSING.

A. Map-Processing

For ease of exposition, we introduce the 2D projections of the UAV and IoT device positions on the ground, $\tilde{\mathbf{u}}_k \in \mathbb{N}^2$ and $\tilde{\mathbf{p}}_k \in \mathbb{N}^2$ respectively, given by

$$\tilde{\mathbf{u}}_k = \left\lfloor \begin{pmatrix} \frac{1}{c} & 0 & 0 \\ 0 & \frac{1}{c} & 0 \end{pmatrix} \mathbf{u}_k \right\rfloor, \quad \tilde{\mathbf{p}}_i = \left\lfloor \begin{pmatrix} \frac{1}{c} & 0 & 0 \\ 0 & \frac{1}{c} & 0 \end{pmatrix} \mathbf{p}_i \right\rfloor \quad (20)$$

rounded to integer grid coordinates.

1) Mapping

The centering and global-local mapping algorithms are based on map-layer representations of the state space. To represent any state with a spatial aspect given by a position and a corresponding value as a map-layer, we define a general mapping function

$$f_{\text{mapping}} : \mathbb{N}^{Q \times 2} \times \mathbb{R}^Q \mapsto \mathbb{R}^{M \times M}. \quad (21)$$

In this function, a map layer $\mathbf{A} \in \mathbb{R}^{M \times M}$ is defined as

$$\mathbf{A} = f_{\text{mapping}}(\{\tilde{\mathbf{p}}_q\}, \{v_q\}), \quad (22)$$

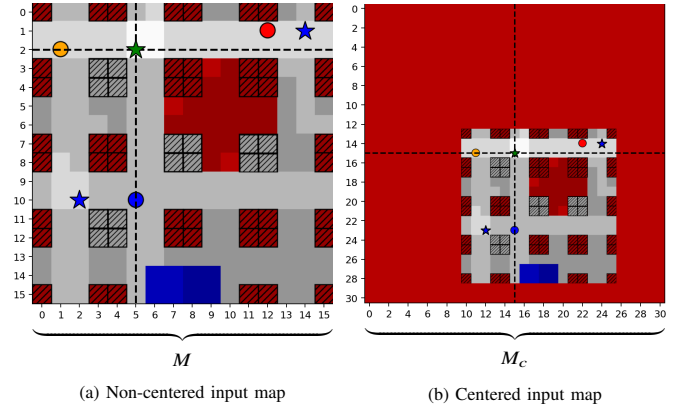


Fig. 2. Comparison of non-centered and centered input maps, with UAV position represented by the green star and the intersection of the dashed lines.

with a set of grid coordinates $\{\tilde{\mathbf{p}}_q\}$ and a set of corresponding values $\{v_q\}$. The elements of \mathbf{A} are given through

$$a_{\tilde{p}_{q,0}, \tilde{p}_{q,1}} = v_q, \quad \forall q \in [0, \dots, Q-1] \quad (23)$$

or 0 if the index is not in the grid coordinates. With this general function, we define the map-layers

$$\mathbf{D}(t) = f_{\text{mapping}}(\{\tilde{\mathbf{u}}_k\}, \{D_k(t)\}) \quad (24a)$$

$$\mathbf{B}(t) = f_{\text{mapping}}(\{\tilde{\mathbf{p}}_i(t)\}, \{b_i(t)\}) \quad (24b)$$

$$\Phi(t) = f_{\text{mapping}}(\{\tilde{\mathbf{p}}_i(t)\}, \{\phi_i(t)\}) \quad (24c)$$

for device data, UAV flying times, and UAV operational status respectively. If the map-layers are of same type they can be stacked to form a tensor of $\mathbb{R}^{M \times M \times n}$ for ease of representation.

2) Map Centering

Given a tensor $\mathbf{A} \in \mathbb{R}^{M \times M \times n}$ describing the map-layers, a centered tensor $\mathbf{B} \in \mathbb{R}^{M_c \times M_c \times n}$ with $M_c = 2M - 1$ is defined through

$$\mathbf{B} = f_{\text{center}}(\mathbf{A}, \tilde{\mathbf{p}}, \mathbf{x}_{\text{pad}}), \quad (25)$$

with the centering function defined as

$$f_{\text{center}} : \mathbb{R}^{M \times M \times n} \times \mathbb{N}^2 \times \mathbb{R}^n \mapsto \mathbb{R}^{M_c \times M_c \times n}. \quad (26)$$

The elements of \mathbf{B} with respect to the elements of \mathbf{A} are defined as

$$\mathbf{b}_{i,j} = \begin{cases} \mathbf{a}_{i+\tilde{p}_0-M+1, j+\tilde{p}_1-M+1}, & M \leq i + \tilde{p}_0 + 1 < 2M \\ \quad \wedge M \leq j + \tilde{p}_1 + 1 < 2M \\ \mathbf{x}_{\text{pad}}, & \text{otherwise,} \end{cases} \quad (27)$$

effectively padding the map layers of \mathbf{A} with the padding value \mathbf{x}_{pad} . Note that $\mathbf{a}_{i,j}$, $\mathbf{b}_{i,j}$, and \mathbf{x}_{pad} are vector valued of dimension \mathbb{R}^n . An illustration of the centering on a 16×16 map ($M = 16$, $M_c = 31$) can be seen in Figure 2 with the legend in Table I.

3) Global-Local Map

The tensor $\mathbf{B} \in \mathbb{R}^{M_c \times M_c \times n}$ resulting from the map centering function is processed in two ways. The first is creating a local map according to

$$\mathbf{X} = f_{\text{local}}(\mathbf{B}, l) \quad (28)$$

with the local map function defined by

$$f_{\text{local}} : \mathbb{R}^{M_c \times M_c \times n} \times \mathbb{N} \mapsto \mathbb{R}^{l \times l \times n}. \quad (29)$$

The elements of \mathbf{X} with respect to the elements of \mathbf{B} are defined as

$$\mathbf{x}_{i,j} = \mathbf{b}_{i+M-\lceil \frac{l}{2} \rceil, j+M-\lceil \frac{l}{2} \rceil} \quad (30)$$

This operation is effectively a central crop of size $l \times l$.

The second processing creates a global map according to

$$\mathbf{Y} = f_{\text{global}}(\mathbf{B}, g) \quad (31)$$

with the global map function

$$f_{\text{global}} : \mathbb{R}^{M_c \times M_c \times n} \times \mathbb{N} \mapsto \mathbb{R}^{\lfloor \frac{M_c}{g} \rfloor \times \lfloor \frac{M_c}{g} \rfloor \times n} \quad (32)$$

The elements of \mathbf{Y} with respect to the elements of \mathbf{B} are defined as

$$\mathbf{y}_{i,j} = \frac{1}{g^2} \sum_{u=0}^{g-1} \sum_{v=0}^{g-1} \mathbf{b}_{gi+u, gj+v} \quad (33)$$

This operation is equal to an average pooling operation with pooling cell size g .

The functions f_{local} and f_{global} are parameterized through l and g , respectively. Increasing l increases the size of the local map, whereas increasing g increases the size of the average pooling cells, therefore decreasing the size of the global map.

B. Observation Space

Using the map processing functions, the observation space can be defined. The observation space Ω , which is the input space to the agent, is given as

$$\Omega = \underbrace{\Omega_l}_{\text{Local Map}} \times \underbrace{\Omega_g}_{\text{Global Map}} \times \underbrace{\mathbb{N}}_{\text{Flying Time}}$$

containing the local map

$$\Omega_l = \mathbb{B}^{l \times l \times 3} \times \mathbb{R}^{l \times l} \times \mathbb{N}^{l \times l} \times \mathbb{B}^{l \times l}$$

and the global map

$$\Omega_g = \mathbb{R}^{\bar{g} \times \bar{g} \times 3} \times \mathbb{R}^{\bar{g} \times \bar{g}} \times \mathbb{R}^{\bar{g} \times \bar{g}} \times \mathbb{R}^{\bar{g} \times \bar{g}}.$$

with $\bar{g} = \lfloor \frac{M_c}{g} \rfloor$. Note that the compression of the global map through average pooling transforms all map layers into \mathbb{R} . Observations $o_i(t) \in \Omega$ are defined through the tuple

$$o_i(t) = (\mathbf{M}_{l,i}(t), \mathbf{D}_{l,i}(t), \mathbf{B}_{l,i}(t), \Phi_{l,i}(t), \mathbf{M}_{g,i}(t), \mathbf{D}_{g,i}(t), \mathbf{B}_{g,i}(t), \Phi_{g,i}(t), b_i(t)). \quad (34)$$

In one observation tuple, $\mathbf{M}_{l,i}(t)$ is the local observation of agent i of the environment, $\mathbf{D}_{l,i}(t)$ is the local observation of the data to be collected, $\mathbf{B}_{l,i}(t)$ is the local observation of the remaining flying time of all agents, and $\Phi_{l,i}(t)$ is the local observation of the operational status of the agents. $\mathbf{M}_{g,i}(t)$, $\mathbf{D}_{g,i}(t)$, $\mathbf{B}_{g,i}(t)$, and $\Phi_{g,i}(t)$ are the respective global observations. $b_i(t)$ is the remaining flying time of agent i , which is equal to the one in the state space. Note that the environment map's local and global observations are dependent on time, as they are centered around the UAV's time-dependent position. Additionally, it should be noted that the remaining flying time

of agent i is given in the center of $\mathbf{B}_{l,i}(t)$ and additionally as a scalar $b_i(t)$. This redundancy in representation helps the agent to interpret the remaining flying time.

Consequently, the complete mapping from state to observation space is given by

$$O : \mathcal{S} \times \mathcal{I} \mapsto \Omega \quad (35)$$

in which the elements of $o_i(t)$ are defined as follows:

$$\mathbf{M}_{l,i}(t) = f_{\text{local}}(f_{\text{center}}(\mathbf{M}, \mathbf{p}_i(t), [0, 1, 1]^T), l) \quad (36a)$$

$$\mathbf{D}_{l,i}(t) = f_{\text{local}}(f_{\text{center}}(\mathbf{D}(t), \mathbf{p}_i(t), 0), l) \quad (36b)$$

$$\mathbf{B}_{l,i}(t) = f_{\text{local}}(f_{\text{center}}(\mathbf{B}(t), \mathbf{p}_i(t), 0), l) \quad (36c)$$

$$\Phi_{l,i}(t) = f_{\text{local}}(f_{\text{center}}(\Phi(t), \mathbf{p}_i(t), 0), l) \quad (36d)$$

$$\mathbf{M}_{g,i}(t) = f_{\text{global}}(f_{\text{center}}(\mathbf{M}, \mathbf{p}_i(t), [0, 1, 1]^T), g) \quad (36e)$$

$$\mathbf{D}_{g,i}(t) = f_{\text{global}}(f_{\text{center}}(\mathbf{D}(t), \mathbf{p}_i(t), 0), g) \quad (36f)$$

$$\mathbf{B}_{g,i}(t) = f_{\text{global}}(f_{\text{center}}(\mathbf{B}(t), \mathbf{p}_i(t), 0), g) \quad (36g)$$

$$\Phi_{g,i}(t) = f_{\text{global}}(f_{\text{center}}(\Phi(t), \mathbf{p}_i(t), 0), g) \quad (36h)$$

By passing the observation space Ω into the agent instead of the state space \mathcal{S} as done in the previous approaches [19] and [1], the presented path planning problem is artificially converted into a partially observable MDP. Partial observability is a consequence of the restricted size of the local map and the compression of the global map. However, as shown in [20], partial observability does not render the problem infeasible, even for a memory-less agent. Instead, the compression greatly reduces the neural network's size, leading to a significant reduction in training time.

V. MULTI-AGENT REINFORCEMENT LEARNING (MARL)

A. Q-Learning

Q-learning is a model-free RL method [30] where a cycle of interaction between one or multiple agents and the environment enables the agents to learn and optimize a behavior, i.e. the agents observe state $s_t \in \mathcal{S}$ and each performs an action $a_t \in \mathcal{A}$ at time t and the environment subsequently assigns a reward $r(s_t, a_t) \in \mathbb{R}$ to the agents. The cycle restarts with the propagation of the agents to the next state s_{t+1} . The agents' goal is to learn a behavior rule, referred to as a policy that maximizes their reward. A probabilistic policy $\pi(a|s)$ is a distribution over actions given the state such that $\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. In the deterministic case, it reduces to $\pi(s)$ such that $\pi : \mathcal{S} \rightarrow \mathcal{A}$.

Q-learning is based on iteratively improving the state-action value function or Q-function to guide and evaluate the process of learning a policy π . It is given as

$$Q^\pi(s, a) = \mathbb{E}_\pi [G_t | s_t = s, a_t = a] \quad (37)$$

and represents an expectation of the discounted cumulative return G_t from the current state s_t up to a terminal state at time T given by

$$G_t = \sum_{k=t}^T \gamma^{k-t} r(s_k, a_k) \quad (38)$$

with $\gamma \in [0, 1]$ being the discount factor, balancing the importance of immediate and future rewards. For the ease of

exposition, s_t and a_t are abbreviated to s and a , while s_{t+1} and a_{t+1} are abbreviated to s' and a' in the following.

B. Double Deep Q-learning and Combined Experience Replay

As demonstrated in [23], representing the Q-function (37) as a table of values is not efficient in the large state and action spaces of UAV trajectory planning. Instead, a deep Q-network (DQN) parameterizing the Q-function with the parameter vector θ can be trained to minimize the expected temporal difference (TD) error. While a neural network is significantly more data efficient compared to a Q-table due to its ability to generalize, the *deadly triad* [30] of function approximation, bootstrapping and off-policy training can make its training unstable and cause divergence.

Mnih *et al.* [31] applied stabilizing techniques to the DQN training process, such as experience replay, reducing correlations in the sequence of training data. New experiences made by the agent, represented by quadruples of (s, a, r, s') , are stored in the replay memory \mathcal{D} . During training, a minibatch of size m is sampled uniformly from \mathcal{D} and used to compute the loss. The size of the replay memory $|\mathcal{D}|$ was shown to be an essential hyperparameter for the agent's learning performance and typically must be carefully tuned for different tasks or scenarios. Zhang and Sutton [32] proposed combined experience replay as a remedy for this sensitivity with very low computational complexity $O(1)$. In this extension to the replay memory method, only $m - 1$ samples of the minibatch are sampled from memory, and the latest experience the agent made is always added. This corrected minibatch is then used to train the agent. Therefore, all new transitions influence the agent immediately, making the agent less sensitive to the selection of the replay buffer size in our approach.

In addition to experience replay, Mnih *et al.* used a separate target network for the estimation of the next maximum Q-value, giving the loss as

$$L^{\text{DQN}}(\theta) = \mathbb{E}_{s,a,s' \sim \mathcal{D}} [(Q_\theta(s, a) - Y^{\text{DQN}}(s, a, s'))^2] \quad (39)$$

with target value

$$Y^{\text{DQN}}(s, a, s') = r(s, a) + \gamma \max_{a'} Q_{\bar{\theta}}(s', a'). \quad (40)$$

$\bar{\theta}$ represents the parameters of the target network. The parameters of the target network $\bar{\theta}$ can either be updated as a periodic hard copy of θ or as in our approach with a soft update

$$\bar{\theta} \leftarrow (1 - \tau)\bar{\theta} + \tau\theta \quad (41)$$

after each update of θ . $\tau \in [0, 1]$ is the update factor determining the adaptation pace.

Further improvements to the training process were suggested in [33], resulting in the inception of double deep Q-networks (DDQNs). With the application of this extension, we avoid the overestimation of action values under certain conditions in standard DQN and arrive at the loss function for our network given by

$$L^{\text{DDQN}}(\theta) = \mathbb{E}_{s,a,s' \sim \mathcal{D}} [(Q_\theta(s, a) - Y(s, a, s'))^2] \quad (42)$$

where the target value is given by

$$Y^{\text{DDQN}}(s, a, s') = r(s, a) + \gamma Q_{\bar{\theta}}(s', \operatorname{argmax}_{a'} Q_\theta(s', a')). \quad (43)$$

C. Multi-agent Q-learning

The original table-based Q-learning algorithm was extended to the cooperative multi-agent setting by Claus and Boutilier in 1998 [34]. Without changing the underlying principle, it can also be applied to DDQN-based multi-agent cooperation. With the taxonomy from [35], our agents can be classified as homogeneous and non-communicating. Homogeneity is a consequence of deploying a team of identical UAVs with the same internal structure, domain knowledge, and identical action spaces. Non-communication is to be interpreted in a multi-agent system sense, i.e. that the agents can not coordinate their actions or choose what to communicate. However, as they all perceive state information that includes other UAVs' positions, in a practical sense, position information would most likely be communicated via the command and control links of the UAVs, that especially autonomous UAVs would have to maintain for regulatory purposes in any case.

The best way to describe our learning approach is by decentralized deployment or execution with centralized training. As DDQN learning requires an extensive experience database to train the neural networks on, it is reasonable to assume that the experiences made by independently acting agents can be centrally pooled throughout the training phase. After training has concluded, the control systems are individually deployed to the distributed drone agents. The rationale behind this concept is that we investigate a team of homogeneous UAVs with identical capabilities and tasks, therefore all experiences are useful for the training of all agents. In a real-world deployment of a team of quadcopter UAVs, all UAVs would be required to regularly return to a charging station, as flying time remains strongly limited by available on-board battery capacity. While being recharged, the UAVs would upload their experience data to a central server with larger memory and computation resources.

Our setting can not be characterized as fully cooperative as our agents do not share a common reward [36]. Instead, each agent has an individual but identical reward function. As the main component of the reward function is based on the jointly collected data from the IoT devices described in Section III-C, they do share a common goal, leading to the classification of our setting as a simple cooperative one.

D. Neural Network Model

We use a neural network model very similar to the one presented in [20]. Fig. 3 shows the DQN structure and the map centering and global-local map processing. The map information of the environment, NFZs, obstacles, and start/landing area is stacked with the IoT device map and the map with the other UAVs' flying times and operational status. According to Section IV-A, the map is centered on the UAV's position and split into a global and local map. The global and local maps are fed through convolutional layers with ReLU activation and then flattened and concatenated with the scalar input indicating battery content or remaining flight time. After passing through fully connected layers with ReLU activation, the data reaches the last fully-connected layer of size $|\mathcal{A}|$ without activation function, directly representing the Q-values for each action

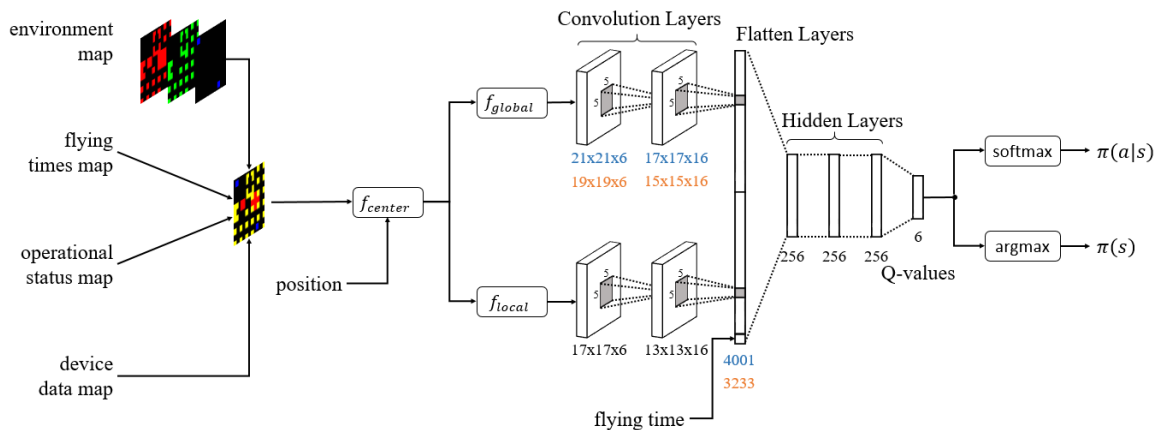


Fig. 3. DQN architecture with map centering and global and local map processing. Layer sizes are shown in blue for the smaller 'Manhattan32' scenario and orange for the larger 'Urban50' scenario.

Parameter	32×32	50×50	Description
$ \theta $	1,175,302	978,694	trainable parameters
N_{\max}	3,000,000	4,000,000	maximum training steps
l	17	17	local map scaling
g	3	5	global map scaling
$ \mathcal{D} $		50,000	replay memory buffer size
m		128	minibatch size
τ		0.005	soft update factor in (41)
γ		0.95	discount factor in (43)
β		0.1	temperature parameter (45)

TABLE III
DDQN HYPERPARAMETERS FOR 32×32 AND 50×50 MAPS.

given the input observation. The argmax of the Q-values, the greedy policy is given by

$$\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q_{\theta}(s, a). \quad (44)$$

It is deterministic and used when evaluating the agent. During training, the soft-max policy

$$\pi(a_i|s) = \frac{e^{Q_{\theta}(s, a_i)/\beta}}{\sum_{a_j \in \mathcal{A}} e^{Q_{\theta}(s, a_j)/\beta}} \quad (45)$$

is used. The temperature parameter $\beta \in \mathbb{R}$ scales the balance of exploration versus exploitation. Hyperparameters are listed in Tab. III.

VI. SIMULATIONS

A. Simulation Setup

In this work, we aim to provide an algorithm¹ that is able to generalize the learned UAV control policy over a large parameter space that defines the specific data collection scenario. That means that at the start of a new training episode, a set of scenario parameters is sampled randomly from a given

¹The Python code for this work is available under https://github.com/hbayerlein/uav_data_harvesting.

range of possible values defining the mission. Then the mission starts and the agents are deployed to collect as much data as possible in the given circumstances. Specifically, we define a new mission through the following randomly varying scenario parameters:

- Number of UAVs deployed;
- Number and position of IoT sensor nodes;
- Amount of data to be collected from IoT nodes;
- Flying time available for UAVs at mission start;
- UAV start positions.

The exact value ranges from which these parameters are sampled are given in the following Sections VI-C and VI-D depending on the map. We deploy our system on two different maps. In 'Manhattan32', the UAVs fly inside 'urban canyons' through a dense city environment discretized into 32×32 cells, whereas 'Urban50' is an example of a less dense but larger 50×50 urban area. Note that we only trained a single agent on each of these maps, meaning that all results discussed in the following are a result of only two trained agents. Generalization over this large parameter space is possible in part due to the learning efficiency benefits from feeding map information centered on the agents' respective positions into the network, as we have described previously in [1].

We use the following metrics to evaluate the agents' performance on different maps and under different scenario instances:

- *Successful landing*: records whether all agents have landed in time at the end of an episode;
- *Collection ratio*: the ratio of total collected data at the end of the mission to the total device data that was available at the beginning of the mission;
- *Collection ratio and landed*: the product of *successful landing* and *collection ratio* per episode.

Evaluation is challenging as we train a single control policy to generalize over a large scenario parameter space. During training, we evaluate the agents' training progress in a randomly selected scenario every five episodes and form an average over multiple evaluations. A single evaluation could

be tainted by unusually easy conditions, e.g. when all devices are placed very close to each other by chance. Therefore, only an average over multiple evaluations can be indicative of the agents' learning progress. As it is computationally infeasible to evaluate the trained system on all possible scenario variations, we perform Monte Carlo analysis on a large number of randomly selected scenario parameter combinations.

Irrespective of the map, the grid cell size is $c = 10\text{m}$ and the UAVs fly at a constant altitude of $h = 10\text{m}$ over city streets. The UAVs are not allowed to fly over tall buildings, enter NFZs, or leave the respective grid worlds. Each mission time slot $t \in [0, T]$ contains $\lambda = 4$ scheduled communication time slots $n \in [0, N]$. Propagation parameters (see II-B) are chosen in-line with [37] according to the urban micro scenario with $\alpha_{\text{LoS}} = 2.27$, $\alpha_{\text{NLoS}} = 3.64$, $\sigma_{\text{LoS}}^2 = 2$ and $\sigma_{\text{NLoS}}^2 = 5$.

Due to the drones flying below or slightly above building height, the wireless channel is characterized by strong LoS/NLoS dependency and shadowing. The shadowing maps used for simulation of the environment were computed using ray tracing from and to the center points of cells based on a variation of Bresenham's line algorithm. Transmission and noise powers are normalized by defining a cell-edge SNR for each map, which describes the SNR between one drone on ground level at the center of the map and an unobstructed IoT device maximally far apart at one of the grid corners. The agents have absolutely no prior knowledge of the shadowing maps or wireless channel characteristics.

B. Training with Map-based vs. Scalar Inputs

In this section, we show that our map-based approach has a good complexity-performance trade-off in comparison to classical scalar input neural network approaches from the literature despite the added complexity through map-processing. To illustrate that it is in fact imperative for training success to feed map information instead of concatenated scalar values as state input to the agent, we extend our previous analysis from [1] and [20] by comparing our proposed centered global-local map approach to agents trained only on scalar inputs. This is not an entirely fair comparison as the location of NFZs, buildings, and start/landing zones can not be efficiently represented by scalar inputs and must be therefore learned by the scalar agents through trial and error. However, the comparison illustrates the need for state space representations that are different from the traditional scalar inputs and confirms that scalar agents are not able to solve the multi-UAV path planning problem over the large scenario parameter space presented. Conversely, the alternative comparison of map-based and scalar agents trained on a *single* data harvesting scenario would not yield meaningful results as our method is specifically designed to generalize over a large variety of scenarios and would require tweaking in exploration behavior and reward balance to find the optimal solution to a single scenario. Note that most of the previous work discussed in section I-A is precisely focused on finding optimal DRL solutions to single scenario instances.

The observation space of the agents trained with concate-

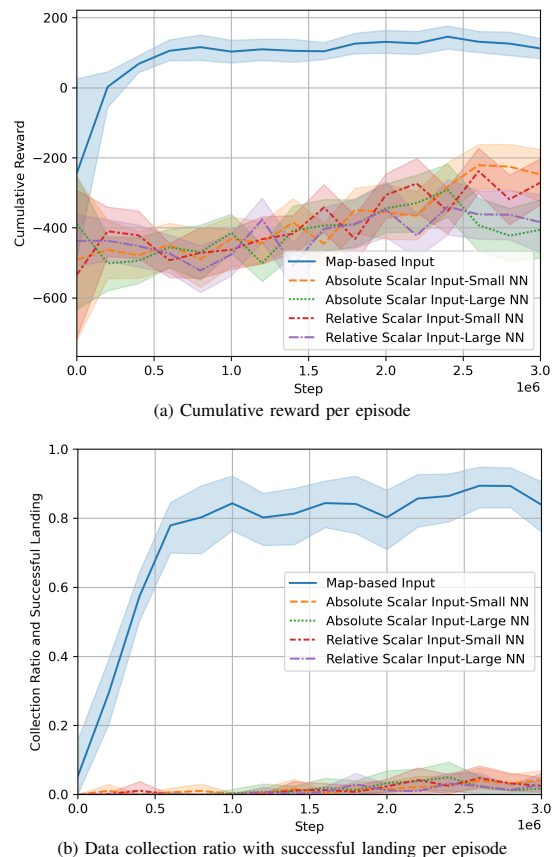


Fig. 4. Training process comparison between *map-based* DRL path planning and *scalar* input DRL path planning. Scalar inputs to the neural networks (NNs) are either encoded as *absolute* coordinate values or *relative* distances from the respective agent. We compare two different scalar input network architectures with *large* and *small* numbers of trainable parameters. The average and 99% quantiles are shown with metrics per training episode grouped in bins of $2e5$ step width. Note that the metrics are plotted over training steps as training episode length is variable.

nated scalar inputs is described by

$$\begin{aligned}
 O_{\text{scalar}} = & \underbrace{\mathbb{N}^2}_{\text{Ego Position}} \times \underbrace{\mathbb{N}}_{\text{Ego Flying Time}} \quad \left. \vphantom{\mathbb{N}^2} \right\} \text{Ego agent} \\
 & \times \underbrace{\mathbb{N}^{I \times 2}}_{\text{UAV Positions}} \times \underbrace{\mathbb{N}^I}_{\text{Flying Times}} \times \underbrace{\mathbb{B}^I}_{\text{Operational Status}} \quad \left. \vphantom{\mathbb{N}^{I \times 2}} \right\} \text{Other agents} \quad (46) \\
 & \times \underbrace{\mathbb{N}^{K \times 2}}_{\text{Device Positions}} \times \underbrace{\mathbb{R}^K}_{\text{Device Data}} \quad \left. \vphantom{\mathbb{N}^{K \times 2}} \right\} \text{Devices}
 \end{aligned}$$

forming the input of the neural network as concatenated scalar values. Since the number of agents and devices is variable, the scalar input size is fixed to the maximum number of agents and devices. The agent and device positions are either represented as *absolute* values in the grid coordinate frame or *relative* as distances from the ego agent. The neural network is either *small*, containing the same number of hidden layers as in Fig. 3, or *large*, for which the number and size of hidden layers is adapted such that the network has as many trainable parameters as the map-based 32×32 agent in Tab. III.

Metric	Manhattan32	Urban50
Successful Landing	99.4%	98.8%
Collection Ratio	88.0%	82.1%
Collection Ratio and Landed	87.5%	81.1%

TABLE IV
PERFORMANCE METRICS AVERAGED OVER 1000 RANDOM SCENARIO
MONTE CARLO ITERATIONS.

Fig. 4 shows the cumulative reward and the collection ratio with successful landing metric over training time on the 'Manhattan32' map for the five different network architectures. It is clear that the scalar agents are not able to effectively adapt to the changing scenario conditions. The *small* neural network agents seem to have a slight edge over the *large* agents, but representing the positions as *absolute* or *relative* does not influence the results.

Referring further to Fig. 4, the map-based agent converges to final performance metric levels after the first 20% of the training steps. However we observed that additional training is needed after that to optimize the trajectories in a more subtle way for flight time efficiency and multi-UAV coordination. The overall training time for the full 3 million training steps was around 40 hours on a 2017 Nvidia Titan Xp GPU.

C. 'Manhattan32' Scenario

The scenario, as shown in Fig. 5 is defined by a Manhattan-like city structure containing mostly regularly distributed city blocks with streets in between, as well as two NFZ districts and an open space in the upper left corner, divided into $M = 32$ cells in each grid direction. This is double the size of the otherwise similarly designed single UAV scenario in [1]. We are able to solve the larger scenario without increasing network size, thanks to the global-local map approach. The value ranges from which the randomized scenario parameters are chosen as follows: number of deployed UAVs $I \in \{1, 2, 3\}$, number of IoT sensors $K \in [3, 10]$, data volume to be collected $D_{k,init} \in [5.0, 20.0]$ data units per device, maximum flying time $b_0 \in [50, 150]$ steps, and 18 possible starting positions. The IoT device positions are randomized throughout the unoccupied map space.

The performance on both maps is evaluated using Monte Carlo simulations on their respective full range of scenario parameters with overall average performance metrics shown in Table IV. Both agents show a similarly high successful landing performance. It is expected that the collection ratio cannot reach 100% in some scenario instances depending on the randomly assigned maximum flying time, number of deployed UAVs, and IoT device parameters.

In Fig. 5, three scenario instances chosen from the random Monte Carlo evaluation for number of deployed UAVs $I \in \{1, 2, 3\}$ for 5a through 5c illustrate how the path planning adapts to the increasing number of deployed UAVs. All other scenario parameters are kept fixed. It is a fairly complicated scenario with a large number of IoT devices spread out over the whole map, including the brown and purple device inside an NFZ. The agents have no access to the shadowing map and

have to deduce shadowing effects from building and device positions.

In Fig. 5a, only one UAV starting in the upper left corner is deployed. Due to its flight time constraint, the agent ignores the blue, red, purple, and brown IoT devices while collecting all data from the other devices on an efficient trajectory to the landing zone in the lower right corner. When a second UAV is deployed in Fig. 5b, the data collection ratio increases to 76.5%. While the first UAV's behavior is almost unchanged compared to the single UAV deployment, the second UAV flies to the landing zone in the lower right corner via an alternative trajectory collecting data from the devices the first UAV ignores. With the number of deployed UAVs increased to three (two starting from the upper left and one from the lower right zone) in Fig. 5c, all data can be collected. The second UAV modifies its behavior slightly, accounting for the fact that the third UAV can collect the cyan device's data now. The three UAVs divide the data harvesting task fairly among themselves, leading to full data collection with in-time landing on efficient trajectories while avoiding the NFZs.

D. 'Urban50' Scenario

Fig. 6 shows three example trajectories for UAV counts of $I \in \{1, 2, 3\}$ for 6a through 6c in the large 50×50 urban map. The scenario is defined by an urban structure containing irregularly shaped large buildings, city blocks and an NFZ, with the start/landing zone surrounding a building in the center, divided into $M = 50$ cells in each grid direction. The map has an order of magnitude more cells than the scenarios in [1]. The ranges for randomized scenario parameters are chosen as follows: number of deployed UAVs $I \in \{1, 2, 3\}$, number of IoT sensors $K \in [5, 10]$, data volume to be collected $D_{k,init} \in [5.0, 20.0]$ data units, maximum flying time $b_0 \in [100, 200]$ steps, and 40 possible starting positions. The IoT device positions are randomized throughout the unoccupied map space.

Fig. 6a shows a single agent trying to collect as much data as possible during the allocated maximum flying time. The agent focuses on collecting the data from the relatively easily reachable device clusters on the right and lower half before safely landing. With a second UAV assigned to the mission as shown in Fig. 6b, one UAV services the devices on the lower left of the map, while the other one collects data from the devices on the lower right, ignoring the more isolated blue and orange device in the top half of the map. A third UAV makes it possible to divide the map into three sectors and collect all IoT device data, as shown in Fig. 6c.

This map's primary purpose is to showcase the significant advantages in terms of training time efficiency and the required network size from the global-local map approach. Thanks to a higher global map scaling or compression factor g (see Table III), the number of trainable parameters of the network employed in this scenario is even smaller compared to the network used for 'Manhattan32'. A network without a map scaling approach would need to be of size 34,061,446, hence a size that is infeasible to train using reasonable resources.

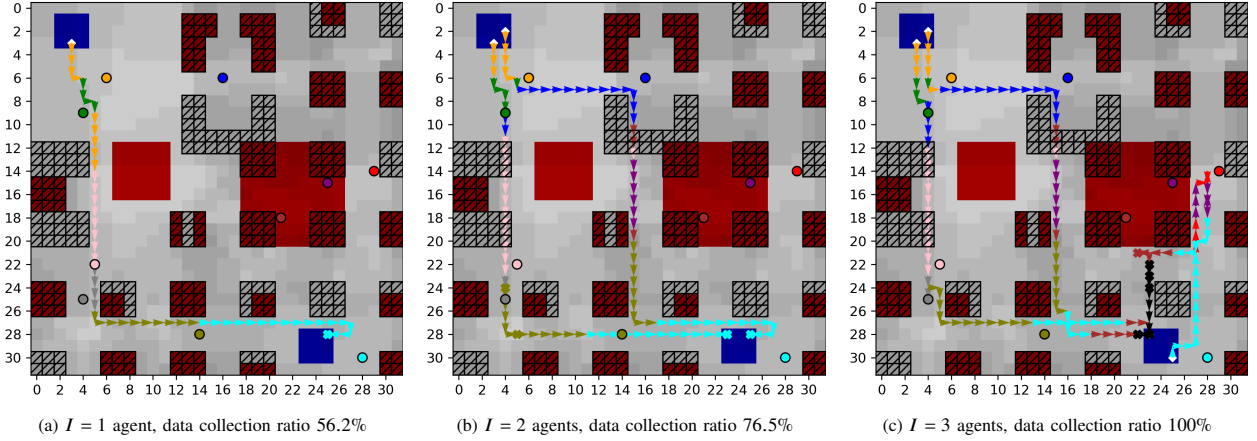


Fig. 5. Example trajectories for 'Manhattan32' map with $K = 10$ IoT devices, all with $D_{k,init} = 15$ data units to be picked up and a maximum flying time of $b_0 = 60$ steps. The color of the UAV movement arrows shows with which device the drone is communicating at the time (see legend in Table I).

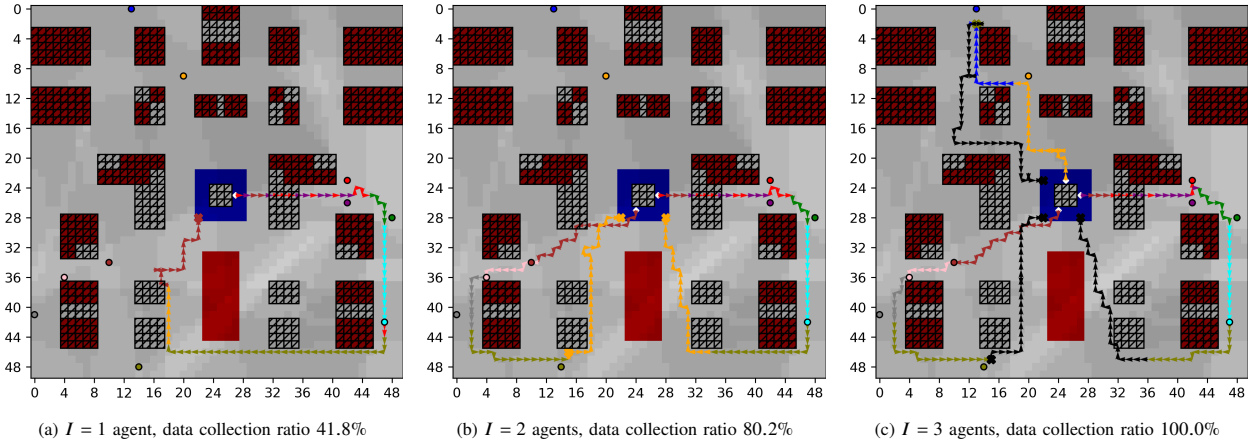


Fig. 6. Example trajectories for 'Urban50' map with $K = 10$ IoT devices, all with $D_{k,init} = 15$ data units to be picked up and a maximum flying time of $b_0 = 100$ steps for all UAVs (legend in table I).

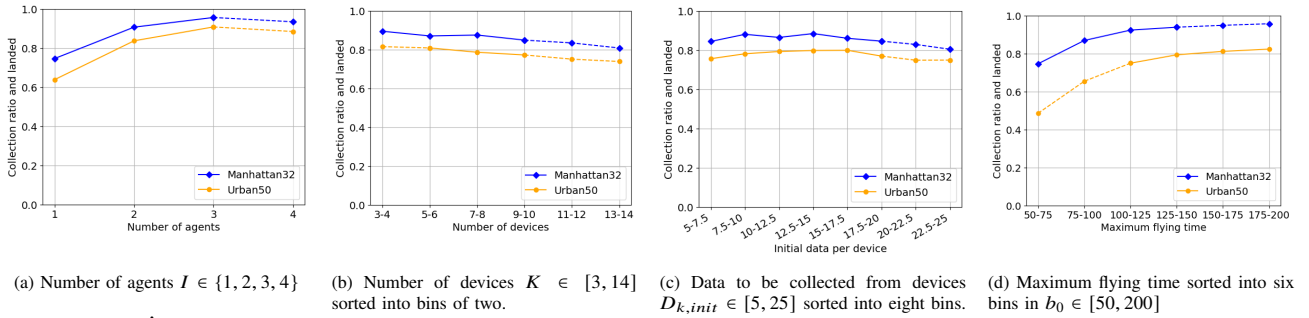


Fig. 7. Influence of specific scenario parameters on the data collection ratio with successful landing of all agents. Each data point is an average of 500 Monte Carlo iterations over the respective parameter spaces for the 'Manhattan32' and 'Urban50' map. The parameters within the training range are rendered in solid lines and the out-of-distribution parameter evaluation in dashed lines.

E. Influence of Scenario Parameters on Performance and System-level Benefits

An advantage of our approach to learn a generalized path planning policy over various scenario parameters is the possibility to analyze how performance indicators change over a variable parameter space. This makes it possible for an operator to decide on system-level trade-offs, e.g. how many drones to deploy vs. collected data volume. An excellent example that we found for the 'Manhattan32' map was that deploying multiple coordinating drones can trade-off the cost of extra equipment (i.e. the extra drones) for substantially reduced mission time. For instance, it takes twice the flying time ($b_0 = 150$) for a single UAV to complete the data collection mission that two coordinating UAVs will require ($b_0 = 75$) to conclude successfully. Specifically, that means that for both scenarios the average data collection ratio with in-time successful landing stays at the same performance level of around 88%.

We first analyze the performance of the agent in Fig. 7 within the training range of the scenario parameters (solid lines), then extend the analysis to out-of-distribution scenarios (dashed lines) in the last paragraph of this section. Fig. 7 shows the influence of single scenario parameters on the average data collection ratio with successful landing of all agents. As already evident from the example trajectories shown previously, Fig. 7a indicates the increase in collection performance when more UAVs are deployed. At the same time, more UAVs lead to increased collision avoidance requirements, as we observed through more safety controller activations in the early training phases. As IoT devices are positioned randomly throughout the unoccupied map space, an increase in devices leads to more complex trajectory requirements and a drop in performance, as depicted in Fig. 7b.

Fig. 7c shows the influence of increasing initial data volume per device on the overall collection performance. It appears that higher initial data volumes per device are beneficial roughly up to the point of $D_{k,init} \in [10, 12.5]$ data units, after which flying time constraints force the UAVs to abandon some of the data, and the collection ratio shows a slightly negative trend. An increase in available flying time is clearly beneficial to the collection performance, as indicated in Fig. 7d. However, the effect becomes smaller when most of the data is collected, and the UAVs start to prioritize minimizing overall flight time and safe landing over the collection of the last bits of data.

It is further shown in Fig. 7 how the agents react to scenario parameters which were not encountered during training. The corresponding values are highlighted with dashed lines. It can be seen that the performance of the agents follows the same trend as in the rest of the data, when increasing the number of devices (Fig. 7b) or initial data per device (Fig. 7c) out of the trained region. When increasing the maximum flying time (Fig. 7d) for the Manhattan32 agents, or decreasing it for Urban50 agents, the collection ratio with successful landing performance, increases or decreases accordingly. Incrementing the number of agents to four (Fig. 7a) reduces the performance slightly. The reason is the decrease in landing performance.

However, this is to be expected since the probability of all agents landing decreases with the number of agents. Since the collection ratio is nearly saturated for the scenarios with three agents, the drop in overall landing performance decreases the collection ratio and landed performance. In general, it is evident that the proposed approach cannot only generalize over the whole range of scenario parameters encountered during training but can also extrapolate successfully to out-of-distribution parameters.

VII. CONCLUSION

We have introduced a multi-agent reinforcement learning approach that allows us to control a team of cooperative UAVs on a data harvesting mission in a large variety of scenarios without the need for recomputation or retraining when the scenario changes. By leveraging a DDQN with combined experience replay and convolutionally processing dual global-local map information centered on the agents' respective positions, the UAVs are able to find efficient trajectories that balance data collection with safety and navigation constraints without any prior knowledge of the challenging wireless channel characteristics in the urban environments. We have also presented a detailed description of the underlying path planning problem and its translation to a decentralized partially observable Markov decision process. In future work, we will extend the UAVs' action space to altitude and continuous control, requiring an RL algorithm different from Q-learning with a continuous action space and adding height information to the agents' observations space. Moreover, we will investigate if attention-based mechanisms can be used for map processing, assessing their viability with respect to performance and computational requirements in this context. Further improvements in learning efficiency could be achieved when combining our approach with multi-task reinforcement learning or transfer learning [26], a step that would bring RL-based autonomous UAV control in the real-world even closer to realization.

REFERENCES

- [1] H. Bayerlein, M. Theile, M. Caccamo, and D. Gesbert, "UAV path planning for wireless data harvesting: A deep reinforcement learning approach," in *IEEE Global Communications Conference (GLOBECOM)*, 2020.
- [2] Y. Zeng, Q. Wu, and R. Zhang, "Accessing from the sky: A tutorial on UAV communications for 5G and beyond," *Proceedings of the IEEE*, vol. 107, no. 12, pp. 2327–2375, 2019.
- [3] K. Namuduri, "Flying cell towers to the rescue," *IEEE Spectrum*, vol. 54, no. 9, pp. 38–43, sep 2017.
- [4] M. Minevich, "How Japan is tackling the national & global infrastructure crisis & pioneering social impact - [online]," *Forbes*, 21 Apr 2020.
- [5] J. Cui, Z. Ding, Y. Deng, and A. Nallanathan, "Model-free based automated trajectory optimization for UAVs toward data transmission," in *IEEE Global Communications Conference (GLOBECOM)*, 2019.
- [6] Y. Pan, Y. Yang, and W. Li, "A deep learning trained by genetic algorithm to improve the efficiency of path planning for data collection with multi-UAV," *IEEE Access*, vol. 9, pp. 7994–8005, 2021.
- [7] J. Tang, J. Song, J. Ou, J. Luo, X. Zhang, and K.-K. Wong, "Minimum throughput maximization for multi-UAV enabled WPCN: A deep reinforcement learning method," *IEEE Access*, vol. 8, pp. 9124–9132, 2020.
- [8] Y. Zhang, Z. Mou, F. Gao, L. Xing, J. Jiang, and Z. Han, "Hierarchical deep reinforcement learning for backscattering data collection with multiple UAVs," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3786–3800, 2021.

- [9] C. H. Liu, Z. Dai, Y. Zhao, J. Crowcroft, D. O. Wu, and K. Leung, "Distributed and energy-efficient mobile crowdsensing with charging stations by deep reinforcement learning," *IEEE Transactions on Mobile Computing*, vol. 20, no. 1, pp. 130–146, 2021.
- [10] M. Yi, X. Wang, J. Liu, Y. Zhang, and B. Bai, "Deep reinforcement learning for fresh data collection in UAV-assisted IoT networks," in *IEEE Conference on Computer Communications Workshops*, 2020.
- [11] F. Wu, H. Zhang, J. Wu, L. Song, Z. Han, and H. V. Poor, "UAV-to-device underlay communications: Age of information minimization by multi-agent deep reinforcement learning," *IEEE Transactions on Communications* (early access), 2021.
- [12] J. Hu, H. Zhang, L. Song, R. Schober, and H. V. Poor, "Cooperative Internet of UAVs: Distributed trajectory design by multi-agent deep reinforcement learning," *IEEE Transactions on Communications*, vol. 68, no. 11, pp. 6807–6821, 2020.
- [13] M. A. Abd-Elmagid, A. Ferdowsi, H. S. Dhillon, and W. Saad, "Deep reinforcement learning for minimizing age-of-information in UAV-assisted networks," in *IEEE Global Communications Conference (GLOBECOM)*, 2019.
- [14] Y. Hu, M. Chen, W. Saad, H. V. Poor, and S. Cui, "Distributed multi-agent meta learning for trajectory design in wireless drone networks," to appear in *IEEE Journal on Selected Areas in Communications*, arXiv:2012.03158 [cs.LG], 2020.
- [15] X. Li, H. Yao, J. Wang, S. Wu, C. Jiang, and Y. Qian, "Rechargeable multi-UAV aided seamless coverage for QoS-guaranteed IoT networks," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10902–10914, 2019.
- [16] X. Liu, Y. Liu, and Y. Chen, "Reinforcement learning in multiple-UAV networks: Deployment and movement design," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 8036–8049, 2019.
- [17] R. Shakeri, M. A. Al-Garadi, A. Badawy, A. Mohamed, T. Khattab, A. K. Al-Ali, K. A. Harras, and M. Guizani, "Design challenges of multi-UAV systems in cyber-physical applications: A comprehensive survey and future directions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3340–3385, 2019.
- [18] W. Saad, M. Bennis, M. Mozaffari, and X. Lin, *Wireless Communications and Networking for Unmanned Aerial Vehicles*. Cambridge University Press, 2020.
- [19] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, "UAV coverage path planning under varying power constraints using deep reinforcement learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.
- [20] —, "UAV path planning using global and local map information with deep reinforcement learning," arXiv:2010.06917 [cs.RO], 2021.
- [21] Z. Ullah, F. Al-Turjman, and L. Mostarda, "Cognition in UAV-aided 5G and beyond communications: A survey," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 3, pp. 872–891, 2020.
- [22] M.-A. Lahmeri, M. A. Kishk, and M.-S. Alouini, "Artificial intelligence for UAV-enabled wireless networks: A survey," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 1015–1040, 2021.
- [23] H. Bayerlein, P. De Kerret, and D. Gesbert, "Trajectory optimization for autonomous flying base station via reinforcement learning," in *IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2018.
- [24] F. Venturini, F. Mason, F. Pase, F. Chiariotti, A. Testolin, A. Zanella, and M. Zorzi, "Distributed reinforcement learning for flexible UAV swarm control with transfer learning capabilities," in *Proceedings of the 6th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*, 2020.
- [25] C. H. Liu, X. Ma, X. Gao, and J. Tang, "Distributed energy-efficient multi-UAV navigation for long-term communication coverage by deep reinforcement learning," *IEEE Transactions on Mobile Computing*, vol. 19, no. 6, pp. 1274–1285, 2020.
- [26] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," *International Conference on Machine Learning Workshop on Real-Life RL*, arXiv:1904.12901 [cs.LG], 2019.
- [27] Z. Liu, R. Sengupta, and A. Kurzhanskiy, "A power consumption model for multi-rotor small unmanned aircraft systems," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2017.
- [28] A. I. Hentati and L. C. Fourati, "Comprehensive survey of uavs communication networks," *Computer Standards & Interfaces*, p. 103451, 2020.
- [29] F. A. Oliehoek and C. Amato, *A concise introduction to decentralized POMDPs*. Springer, 2016.
- [30] R. S. Sutton and A. G. Barto, *Reinforcement Learning: an introduction*, 2nd ed. MIT Press, 2018.
- [31] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [32] S. Zhang and R. S. Sutton, "A deeper look at experience replay," arXiv:1712.01275 [cs.LG], 2017.
- [33] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016, pp. 2094–2100.
- [34] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," *AAAI/IAAI*, vol. 1998, no. 746-752, p. 2, 1998.
- [35] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.
- [36] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," arXiv:1911.10635 [cs.LG], to be published as chapter in *Handbook on RL and Control (Springer)*, 2019.
- [37] 3GPP TR 38.901 version 14.0.0 Release 14, "Study on channel model for frequencies from 0.5 to 100 GHz," ETSI, Tech. Rep., May 2017.

Chapter 5

Reinforcement Learning for Real-world Control Challenges

5.1 Cloud-Edge Training Architecture for Sim-to-Real Deep Reinforcement Learning

Reference

H. Cao, M. Theile, F. G. Wyrwal, and M. Caccamo, “Cloud-edge training architecture for sim-to-real deep reinforcement learning,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9363–9370, IEEE, 2022

DOI: <https://doi.org/10.1109/IROS47612.2022.9981565>

Abstract

Deep reinforcement learning (DRL) is a promising approach to solve complex control tasks by learning policies through interactions with the environment. However, the training of DRL policies requires large amounts of training experiences, making it impractical to learn the policy directly on physical systems. Sim-to-real approaches leverage simulations to pretrain DRL policies and then deploy them in the real world. Unfortunately, the direct real-world deployment of pretrained policies usually suffers from performance deterioration due to the different dynamics, known as the reality gap. Recent sim-to-real methods, such as domain randomization and domain adaptation, focus on improving the robustness of the pretrained agents. Nevertheless, the simulation-trained policies often need to be tuned with real-world data to reach optimal performance, which is challenging due to the high cost of real-world samples. This work proposes a distributed cloud-edge architecture to train DRL agents in the real world in real-time. In the architecture, the inference and training are assigned to the edge and cloud, separating the real-time control loop from the computationally expensive training loop. To overcome the reality gap, our architecture exploits sim-to-real transfer strategies to continue the training of simulation-pretrained agents on a physical system. We demonstrate its applicability on a physical inverted-pendulum control system, analyzing critical parameters. The real-world experiments show that our architecture can adapt the pretrained DRL agents to unseen dynamics consistently and efficiently.

Contributions to this paper

- Shared conceptualization of the cloud-edge training framework
- Shared implementation of the framework for the inverted pendulum system
- Share of paper writing

Copyright

© 2022 IEEE. Reprinted, with permission, from Hongpeng Cao, Mirco Theile, Federico G Wyrwal, and Marco Caccamo, “Cloud-Edge Training Architecture for Sim-to-Real Deep Reinforcement Learning”, 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), October 2022.

See Appendix A.7 for the reuse statement. The following shows the accepted version.

Cloud-Edge Training Architecture for Sim-to-Real Deep Reinforcement Learning

Hongpeng Cao, Mirco Theile, Federico G. Wyrwal, and Marco Caccamo

Abstract—Deep reinforcement learning (DRL) is a promising approach to solve complex control tasks by learning policies through interactions with the environment. However, the training of DRL policies requires large amounts of training experiences, making it impractical to learn the policy directly on physical systems. Sim-to-real approaches leverage simulations to pretrain DRL policies and then deploy them in the real world. Unfortunately, the direct real-world deployment of pretrained policies usually suffers from performance deterioration due to the different dynamics, known as the *reality gap*. Recent sim-to-real methods, such as domain randomization and domain adaptation, focus on improving the robustness of the pretrained agents. Nevertheless, the simulation-trained policies often need to be tuned with real-world data to reach optimal performance, which is challenging due to the high cost of real-world samples.

This work proposes a distributed cloud-edge architecture to train DRL agents in the real world in real-time. In the architecture, the inference and training are assigned to the edge and cloud, separating the real-time control loop from the computationally expensive training loop. To overcome the *reality gap*, our architecture exploits sim-to-real transfer strategies to continue the training of simulation-pretrained agents on a physical system. We demonstrate its applicability on a physical inverted-pendulum control system, analyzing critical parameters. The real-world experiments show that our architecture can adapt the pretrained DRL agents to unseen dynamics consistently and efficiently.¹

I. INTRODUCTION

Deep reinforcement learning (DRL) enables robots to master complicated tasks with human-level performance. Instead of learning from labeled data, the DRL agent learns control policies via interactions with the environment, aiming to maximize cumulative rewards. Recent progress shows that DRL can achieve impressive performances in the robotic control domain, e.g., in locomotion [1], grasping [2], and manipulation [3].

However, due to the high demand for data for DRL, direct training on physical systems presents many challenges, as discussed in [4], [5]. Collecting training data in the real world is expensive concerning time and labor. Human supervision is usually needed to reset the system and monitor its hardware maintenance and safety status. Recent work in the field aims to address these challenges. In [1], an improved soft actor-critic algorithm [6] is proposed to reduce the learning sensitivities to hyperparameter settings, making

training on physical systems more stable. Approaches of off-policy training with replay memory [7], [8] and model-based reinforcement learning [9], [10] can significantly increase real-world sampling efficiency. Moreover, training using demonstrations [11] and scripted policies [12] can ease the real-world exploration. When training on hardware, additional problems are that the dynamics of the systems might be non-stationary due to hardware wear and tear, making long training harder or even impossible. Environmental noise and disturbances can further exacerbate this issue. Moreover, physical systems' inherent sensing, actuating, computation, and data communication delays often violate the Markovian assumption, a prerequisite for reinforcement learning.

In contrast, modern simulations can simulate complex systems and various environments. Recent sim-to-real approaches pretrain agents in simulations and then directly deploy the learned policies for real-world applications without further training. Simulation-based training boosts sampling efficiency significantly since simulations run faster than physical systems and can be further improved via parallel training. Additionally, the system can reset automatically without human intervention. Simulation training also does not require hardware maintenance and safety measures, drastically reducing the need for human supervision. Unfortunately, direct deployment of simulation-trained agents often fails in real-world applications due to the dynamic divergence between the real world and the simulated environment, the so-called *reality gap*. The *reality gap* arises mainly from system under-modeling [4], [13], where the complex dynamics, e.g., the contact and friction effects, are difficult to measure and model. Moreover, computation and communication delays and environmental noise introduce extra modeling errors. System identification approaches such as [14] aim to create more accurate simulations, mitigating modeling errors. However, most of the sim-to-real approaches utilize domain randomization [15], [16], [17] or domain adaptation [18], [19] to improve the agents' robustness in simulations, resulting in a successful direct real-world deployment. In many scenarios, the domain randomization strategies might not be feasible since there might not exist a single policy that can solve all problems within the domain.

Pretraining in simulation to learn a sub-optimal policy and then continuing the training in the real world can take advantage of the efficient simulation training and the real dynamics. With this approach, all the advantages of simulation learning can be adopted, and the *reality gap* can be closed while training in the real world. We follow the continuous sim-to-real training paradigm in this work,

Hongpeng Cao, Mirco Theile, Federico G. Wyrwal, and Marco Caccamo are with the Technical University of Munich (TUM), School of Engineering and Design, Munich, Germany {cao.hongpeng, mirco.theile, federico.wyrwal, mcaccamo}@tum.de

¹A video showing a real-world training process under the proposed method can be found from <https://youtu.be/hMY9-c0SST0>.

as we believe it is the most promising approach for real-world DRL. However, the continuous sim-to-real training paradigm suffers from two main problems. First, DRL requires high-performance computation and benefits from dedicated devices such as GPUs or TPUs for its training loop. In many real-time control systems, e.g., unmanned aerial vehicles or other mobile robots, the plant cannot have a high-performance device onboard due to power, weight, and space constraints. It cannot be controlled directly from a remote high-performance device, as perfect, loss-less, and low-delay communication cannot be guaranteed. The second problem is that the transfer of a pretrained agent to the real physical system is non-trivial. The dynamics can change abruptly, making the value estimates of the DRL agent inaccurate, leading to deteriorating performance.

This work addresses the real-world training problem by introducing a novel distributed cloud-edge architecture. The real-time control loop on the edge is decoupled from the computationally intensive training loop on the cloud. The agent on the edge collects experiences in real-time, sending them to the trainer on the cloud, which periodically updates the edge with the optimized parameters. The agent is double-buffered such that the real-time inference loop is not interrupted by the policy updates. We address the sim-to-real transfer problem by delaying the neural network training at the beginning of the real-world interactions. Specifically, we start the policy optimization later than the value estimate training to avoid policy deterioration based on unstable value estimates.

We evaluate our approach on a physical inverted-pendulum system controlled by a DRL agent deployed on a Raspberry Pi 4B. The training loop is offloaded to a high-performance workstation. The agents are pretrained in intentionally under-modeled simulations to induce different levels of the *reality gap* to analyze the sim-to-real transfer. We further analyze the impact of the neural network optimization delays, highlighting their necessity. Additionally, we investigate the relevance of the edge update frequency on training performance to show that the architecture can work in constrained bandwidth settings, albeit with an impact on training time. Our contributions can be summarized as follows:

- Design of a distributed cloud-edge architecture that enables continuous sim-to-real training for DRL agents;
- Conception and evaluation of sim-to-real transfer methods that mitigate policy performance deterioration after deployment to the physical system;
- Evaluation and analysis of our approach using an off-policy actor-critic DRL method to control a physical inverted pendulum system with the actor deployed on an embedded system.

This work is structured as follows: Section II describes the background needed for the proposed architecture and sim-to-real transfer strategies in Section III. The proposed approach is evaluated with experimental setup discussed in Section IV and results presented in Section V. Section VI concludes the work and gives a brief outlook on future work.

II. BACKGROUND

This section introduces the basics of RL methods and describes the foundations of the Deep Deterministic Policy Gradient (DDPG) algorithm [20], an off-policy algorithm, which we adapted for the proposed cloud-edge architecture.

A. Reinforcement Learning

A DRL agent is interacting with its environment in discrete timesteps, which can be formulated as a Markov Decision Process (MDP) with $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, P, R, \gamma\}$. In the MDP, \mathcal{S} represents a set of states, \mathcal{A} a set of actions, and $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ the state-transition probability function indicating the probability of a state-action pair leading to a specific next state. The reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ maps a state-action-next state triple to a real-valued reward. The discount factor $\gamma \in [0, 1]$ controls the relative importance of immediate and future rewards. The goal in DRL is to find a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$, mapping a state to an action that maximizes the expected return from step t

$$G_t = \sum_{i=t}^{\infty} \gamma^{i-t} R(s_i, a_i, s_{i+1}). \quad (1)$$

To find a policy that maximizes the return, Q-learning approaches estimate the state-action value

$$Q^\pi(s, a) = \mathbb{E}_\pi [G_t | s_t = s, a_t = a], \quad (2)$$

as the return if taking action a at state s and following policy π afterwards.

B. Deep Deterministic Policy Gradient

DDPG is an off-policy actor-critic algorithm, in which the actor is parameterized using a deep neural network with parameters θ , creating the policy approximate π_θ . The critic is parameterized by a deep neural network with parameters ϕ to estimate the state-action value $Q_\phi(s, a)$. During training, the critic network is trained to minimize the expectation of the temporal difference (TD) error

$$L(\phi) = \mathbb{E} [(Q_\phi(s_t, a_t) - y_t)^2], \quad (3)$$

where

$$y_t = R(s_t, a_t, s_{t+1}) + \gamma(1 - \beta(s_{t+1}))Q_{\bar{\phi}}(s_{t+1}, \pi_{\bar{\theta}}(s_{t+1})), \quad (4)$$

with $\beta(s_{t+1})$ indicating whether s_{t+1} is a terminal state. The value and action at the next state are estimated by the critic and actor target networks parameterized with $\bar{\phi}$ and $\bar{\theta}$, respectively.

The actor is aiming to maximize the value estimate of the critic and is optimized using the deterministic gradient

$$\nabla_\theta Q_\phi(s_t, \pi_\theta(s_t)) = \nabla_a Q_\phi(s_t, a) \nabla_\theta \pi_\theta(s_t). \quad (5)$$

DDPG also proposes to soft update the target networks by slowly tracking the learned network: $\bar{\phi} \leftarrow \tau \phi + (1 - \tau) \bar{\phi}$ with $\tau \in (0, 1]$ and similarly for $\bar{\theta}$. The resulting moving-average over the network parameters stabilizes training.

Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm [21], a variant of the DDPG family, utilizes

delayed policy optimization, target policy smoothing regularization, and clipped double Q-learning. In our simulation, we found that only policy smoothing regularization helps the training converge faster, while the others do not show improvements. Thus, we only adopt the idea of policy smoothing regularization used in TD3 by replacing the $\pi_{\bar{\theta}}(s_{t+1})$ in (4) with

$$\text{clip}(\pi_{\bar{\theta}}(s_{t+1}) + a_{\mathcal{N}}, -a_{\max}, a_{\max}), \quad (6)$$

which regularizes the target action by adding a clipped action noise $a_{\mathcal{N}} = \text{clip}(\epsilon, -c, c)$ where ϵ is sampled from a Gaussian distributions $\mathcal{N}(0, \sigma)$. The $\text{clip}(z, z_{\min}, z_{\max})$ function constrains the value of z to the range $[z_{\min}, z_{\max}]$.

C. Combined Experience Replay

Experience Replay (ER) is a technique to train the agent with a batch of B experiences sampled from a buffer of previous transitions. ER is used in DDPG and other recent off-policy RL algorithms to provide uncorrelated data for training deep neural networks [22] and improve sampling efficiency significantly [23].

A common method for improving replay memory efficiency is to use prioritized experience replay [24]. We chose not to use prioritized replay because we hypothesize that for experiences collected in the real world, prioritization may increase the sampling probability of corrupted or abnormal experiences, which arise from noise or other disturbances. We will investigate this hypothesis in future work.

Instead, we use Combined Experience Replay (CER), proposed by Zhang et al. in [25], which aims to remedy the training sensitivity to the buffer capacity with very low computational complexity $\mathcal{O}(1)$. In CER, only $B-1$ samples are sampled from memory, and the agent's latest experience is always added. This allows the training process to react quickly to newly observed transitions.

III. METHODOLOGY

This section presents our continuous training method: a distributed cloud-edge training architecture and the sim-to-real transfer learning strategies.

A. Remote Training Architecture

We devised a distributed cloud-edge training architecture to minimize the computational load on the embedded device used to control the plant. To this end, most of the computation is offloaded to a high-performance device, the cloud, as shown in Figure 1. The diagram shows the three devices, the cloud, the edge, and the plant. The cloud contains the reward function, replay memory, and the full actor-critic setup of the DDPG algorithm. The edge only contains the actor network. The plant represents the physical control system, including actuators and sensors, and is directly connected to the edge. The cloud and edge can be connected via any networking protocol. Two decoupled interaction loops arise by double-buffering the actor on the edge, a real-time control loop between the edge and plant, and a training loop between the cloud and edge.

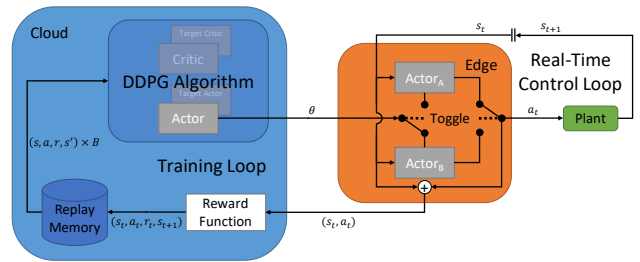


Fig. 1: Distributed cloud-edge training architecture displaying the cloud, edge, and plant with the cloud and edge interacting through the training loop, and the edge and plant interacting through the real-time control loop; the loops are disconnected by double-buffering the actor on the edge.

1) *Real-Time Control Loop*: In the real-time control loop, the currently active actor-network on the edge (Actor_A in Figure 1) infers an action a_t based on the observed state s_t , which is passed to the actuators of the plant. This interaction loop is real-time critical, meaning that it has to be temporally deterministic and predictable. This requirement arises from the necessity of the Markov property for reinforcement learning. In a system that fulfills the Markov property, the transition probability function P only depends on the current state s_t and current action a_t and is independent of the past states and actions [26]. On physical control systems, this property can be easily violated by sensing, computation, actuation, and communication delays, which can make the next state depending on previous actions. This problem can be mitigated by making past actions or states part of the observation space or adding recurrence to the policy [5].

However, these augmentations only create a Markovian observation under consistent or predictable delays. Consistent delays are achieved by scheduling computation and communication in fixed time slots with a fixed period. This mimics traditional real-time control systems [27]. Fixed periods can only be achieved if the computation time has an upper bound smaller than the period. In the proposed architecture, the computation time is given by the inference time of the actor-network on the edge. By decoupling the inference from the training utilizing the cloud and from policy updates by double-buffering the actor, we enable its real-time capabilities.

2) *Training Loop*: The state-action pairs (s_t, a_t) created through the interaction in the control loop are sent to the cloud. The reward function R uses the state s_t , the action a_t , and the next state s_{t+1} to compute the reward r_t for the experience tuple (s_t, a_t, r_t, s_{t+1}) , which is added to the replay memory. The time index is removed in the replay memory, as it is not needed for off-policy training. A batch of B experiences is sampled from the replay memory and passed to the DDPG algorithm. The relative next states in the batch are indicated with s' . The algorithm updates the critics and actors, sending the new actor parameters θ to the edge. On the edge, the parameters are applied to the inactive actor (Actor_B in Figure 1), toggling the active actor on completion.

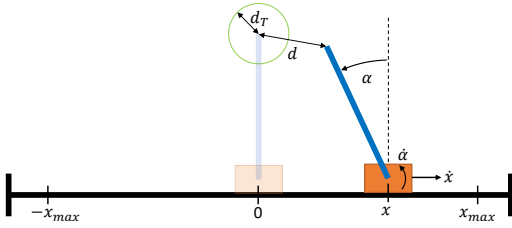


Fig. 2: Inverted pendulum system consisting of a cart and a pole with the target configuration faded out in the middle. The Cartesian Euclidean distance from the tip of the pendulum to the target is indicated by d .

B. Sim-to-Real Transfer

As mentioned before, we augmented the observation space with the previous five actions to overcome non-Markovianess due to delays. However, the agent learned to ignore these extra observations in simulation, as they were not needed in the non-delayed simulation environment. We introduced a one-step delay in simulation, such that the extra observations cannot be ignored.

The main problem when transferring to the real world arises from the replay buffer. A certain amount of experiences is needed in the buffer, such that when sampling from it, the experience batch is uncorrelated. One potential remedy would be to keep experiences from simulation and gradually replace them with real-world data. However, this approach could be sensitive to the size of the *reality gap* because the transition probability function P might be significantly different. We will investigate this assumption in the future. Instead, we adopted a solution that is independent of the size of the *reality gap* by starting with an empty replay memory and collecting N_c real-world samples without training. After N_c steps, the training starts.

An additional problem is that the temporal difference error of the value estimate is very high, which is probably caused by the reality gap. Consequently, the critic is changing significantly at the beginning of the real-world training. Training the actor on the varying critic often leads to losses in performance. Therefore, we further delay optimization steps of the actor to N_a steps in total, with $N_a \geq N_c$.

IV. EXPERIMENTAL SETUP

This section describes the experimental setup for analyzing our proposed architecture and sim-to-real transfer strategies.

A. Inverted Pendulum Control System

The inverted pendulum control system, schematically shown in Figure 2, is a classical benchmark that has been widely studied in the control and real-time domain [28]. The goal in the inverted pendulum control task is to swing up a pendulum and balance it at the upright position under physical constraints. The inverted pendulum is an inherently unstable non-linear system, making it non-trivial to derive a controller that can swing the pendulum up and balance it. In this work, we formulate the inverted pendulum control

problem as a discounted, continuing finite MDP and train a DDPG agent to solve it.

1) *State, Action and Observation Spaces*: The state of the inverted pendulum consists of four elements $s_t = (x_t, \dot{x}_t, \alpha_t, \dot{\alpha}_t)$, with the horizontal cart position x_t , its horizontal velocity \dot{x}_t , the angle of the pole with respect to the upright position $\alpha \in (-\pi, \pi]$, and its angular velocity $\dot{\alpha}_t$. We convert the measured angle α_t into $\sin \alpha_t$ and $\cos \alpha_t$ to simplify the learning process. The control action $a_t \in [-1, 1]$ is the scaling factor of the supply voltage of a DC motor, which drives the cart. Since the delays of the system violate the Markov property, we augment the observation by combing the state observation with the last five actions to make the system observable [5]. The observation space can be expressed as

$$o_t = (x_t, \dot{x}_t, \sin \alpha_t, \cos \alpha_t, \dot{\alpha}_t, a_{t-5}, \dots, a_{t-1}), \quad (7)$$

in which a_{t-5}, \dots, a_{t-1} are the previous five actions. We add terminal states to the discounted infinite MDP that stop the process when safety bounds are violated. A state is terminal if

$$\beta(s_t) = \begin{cases} 1, & \text{if } |x_t| \geq x_{max} \text{ or } |\dot{\alpha}_t| \geq \dot{\alpha}_{max} \\ 0, & \text{otherwise,} \end{cases} \quad (8)$$

i.e., if either its position is exceeding the bounds of the track or the angular velocity is too high, posing a threat to humans or the system.

2) *Reward Function*: In the MDP, the agent aims to minimize the Cartesian Euclidean distance $d(s)$ between the tip of the pole and the target position, as shown in Figure 2. The target position is given by the position of the tip of the pole when $x = \alpha = 0$. We define the reward function as

$$R(s_t, a_t, s_{t+1}) = e^{-\delta d(s_t)} - u a_t^2 - v \beta(s_{t+1}). \quad (9)$$

The first term is rewarding smaller distances to the target. We chose the exponential function because it limits this term to $(0, 1]$ and has strong gradients when the distance is close to zero. This helps the agent to balance the pendulum exactly on target, avoiding drifts. The second term is penalizing high actions and the third term penalizes safety violations. The parameter δ stretches the exponential and u and v balance the importance of the action and safety penalties.

B. Evaluation Metrics

We split the training process into episodes with a maximum length of T_e steps to evaluate the training performance. After every five training episodes, we run an evaluation episode, in which no exploration noise is applied. In these evaluation episodes, we determine if the agent can swing up and balance the pendulum, defined as follows.

The pendulum is defined to be on target if the distance $d(s_t)$ is smaller than a predefined target distance d_T . The consecutive steps that the pendulum is on target are called the consecutive on-target-steps and its evolution is defined as

$$n_{t+1} = \begin{cases} n_t + 1, & d(s_t) \leq d_T \\ 0, & \text{otherwise.} \end{cases} \quad (10)$$

We define that the agent can swing up and balance the pendulum if at step T_e the consecutive on-target steps are $n_{T_e} \geq T_g$.

The training is finished if the agent can reach five consecutive successful evaluation episodes. At this point, we define that the agent converged. The convergence time is the cumulative number of steps in training episodes until the agent converges. During evaluation episodes, no steps are accumulated since no experiences are collected.

C. Simulation Pretraining

The simulation environment used for pretraining is adapted from the OpenAI-Gym cart-pole environment [29], with the parameters set to the values of the physical pendulum. We adapted the original OpenAI-Gym cart-pole environment to use continuous actions and incorporate friction on the cart and pole. To induce different *reality gaps*, we vary a friction factor k_f in simulation, setting the friction at the cart to k_f and at the pole mount to $k_f \times 10^{-4}$.

The actor and critic networks in the DDPG algorithm are both implemented as a Multi-Layer Perceptron (MLP) with three fully connected hidden layers of 256, 128, and 64 neurons activated with ReLU. The observations o_t form the input of the actor, and the observations o_t and the action a_t form the input of the critic. The output layers of the actor and critic are a single neuron activated with Tanh and without activation, respectively.

For simulation training, we use the commonly used OU-noise [30] with decaying noise magnitude. We reset episodes to a random initial state after either T_e is reached, or if $\beta(s_t) = 1$. We further reset the episode when n_t exceeds 100 steps because the experiences collected when the pendulum is on target are very similar and do not improve the learning. These experiences reduce the efficiency of the replay memory as more crucial experiences of the swing-up phase get sampled less likely. The models are pretrained for 1 million steps, with a moving average of the performance determining at which step the best model is saved.

D. Architecture Setup

The distributed architecture follows a cloud-edge pattern. The cloud device is a workstation with a Xeon Silver-CPU (2.1GHz) and an NVIDIA Quadro RTX 8000 Graphics card. The edge device is a Raspberry Pi 4B board without DNN accelerators. The plant is a linear inverted pendulum built by Quanser [31]. The edge device is connected with the plant via USB, through which the sensor readings and control actuation are passed. The cloud and edge are physically connected via Ethernet in a local network. The data communication between the edge and cloud devices is based on TCP packets handled by Redis [32] hosted on the cloud.

The double buffered² actors are deployed on the edge device and interact with the physical system at 30 Hz. The trainer on the cloud optimizes once per experience

²When implementing the double-buffering in a multi-threaded Python environment, the global interpreter lock (GIL) needs to be taken into consideration.

Parameter	Value	Parameter	Value
x_{\max}	0.34 m	δ	5
$\dot{\alpha}_{\max}$	20 rad/s	u	0.1
d_T	0.05 m	v	20
k_f	10	N_c	3500
T_e	1000	N_a	5000
T_g	750	B	128

TABLE I: Default parameters for the experiments.

received from the edge. If the cloud is slower than the edge interactions, a backlog is accumulated and consumed during physical system resets or evaluation episodes where no new experiences are collected. The edge requests new weights from the cloud whenever it finishes applying the previous weights.

As in simulation, we reset the pendulum after T_e steps, or if $\beta(s_t) = 1$, or if n_t exceeds 100. The reset is conducted with a simple P-controller which moves the cart to a random position along the track. The angle and angular rate are naturally randomized. Additionally, every 10,000 steps, we calibrate the angular encoder by letting the pendulum settle and reset the angle to π radians. This is to prevent angle drift over long training sessions. A list of the parameters can be found in Table I.

V. RESULTS

In this work, we are interested in the following three questions:

- 1) How are different sizes of the *reality gap* affecting training time?
- 2) How do various combinations of optimization delays influence training performance on the physical system?
- 3) Is the communication latency between the cloud and edge affecting training performance?

We assume that these experimental variables are orthogonal to each other. Therefore, we vary these parameters independently and compare the convergence time of corresponding real-world training instances. A comparison between pretraining in simulation and directly training in the real world was not possible. When training from scratch, the randomly initialized agent was too aggressive, repeatedly causing damage to the physical system.

For all the experiments, if not indicated otherwise, the parameters used are listed in Table I. We conducted five real-world training instances for each configuration, on average trained for around one hour each.

A. Reality Gap

To study the influence of the *reality gap*, we vary the friction properties of the system, as they are hard to model and can thus be an essential contributor to the *reality gap* [33]. We vary the friction factor k_f in simulation, setting its value to one of $\{0, 5, 10, 12, 16\}$. For $k_f = 0$, the system is frictionless, and $k_f = 16$ results in the maximum friction under which the agent can learn to swing up the pendulum.

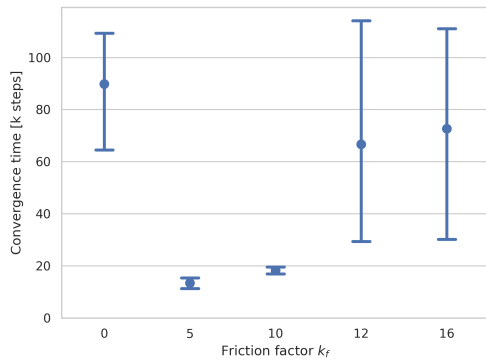


Fig. 3: Convergence time comparison of real-world training instances with different models pretrained with different friction factors.

For $k_f > 16$, the agent does not learn repeatably, which we attribute to the high energy losses.

The convergence times for each configuration are depicted in Figure 3. The results show that the convergence time is lowest for $k_f = 5$, closely followed by $k_f = 10$. When training with $k_f = 0$, the pendulum does not reach the target at the beginning of the training where it overshoots the target with $k_f > 10$. This is expected as, compared with the simulation, the friction decelerates the pendulum stronger or weaker, respectively.

Taking a closer look at the training process, Figure 4 shows the training time-series for the training instances that resulted in the median convergence time of each configuration. It can be seen that the training instances for $k_f = 5$ and $k_f = 10$ converged very fast after only a few evaluation episodes. The other configurations required significantly more training time. The frictionless pretrained agent needs to learn to swing up but is already sufficiently good at balancing, since in the frictionless simulation, the swing-up is easier and the balancing is harder. This can be observed in the training time-series, as the $k_f = 0$ agent requires many training steps to succeed once, corresponding to learning to swing up, but then converges quickly as it can already balance well. The pretrained agents with $k_f = 12$ and $k_f = 16$ succeed earlier, indicating that they can swing up but require more training steps to learn how to stabilize the pendulum consistently. This might have been exacerbated by our early resetting policy, which was supposed to balance the replay memory by reducing the experiences with a stabilized pendulum. This strategy could have inadvertently favored pretrained agents that were already good at balancing and needed to learn to swing up and disadvantaged agents that needed to learn to stabilize the pendulum.

Generally, it can be observed that using the proposed distributed cloud-edge architecture with the transfer strategies, pretrained models with different *reality gaps* can learn to swing up and balance the inverted pendulum in the real world repeatably and consistently. For the following experiments, we selected the pretrained model with $k_f = 10$ as it cannot

succeed immediately, requiring real-world training. It further has low variance, making the following comparisons easier.

B. Optimization Delay

To study the effect of the optimization delays N_c and N_a on the convergence time we compare different delay combinations out of $\{128, 3500, 5000\}$ for N_c and N_a in Figure 5. The minimal delay is the batch size $B = 128$. Since the actor is trained on the critic, we only investigate configurations with $N_a \geq N_c$. Additionally, as the actor optimization delay has similarities with the delayed training in TD3, we tested one configuration with the actor delay implementation of TD3.

The first conclusion from the data is that an $N_c = B$ is not sufficient since the replay memory cannot decorrelate experience samples, leading to unpredictable learning behavior of the critic. The bad learning performance of the critic cannot be recovered by delaying the actor optimization as seen for $N_a = 5000$. Additionally, a high $N_c = 5000$, appears to be less beneficial than $N_c = 3500$. However, this effect is marginal, possibly caused by statistical errors due to few samples. The second observation is that the additional actor delay $N_a > N_c$ seems to give a slight advantage since the combination $N_c = 3500$ and $N_a = 5000$ performed the best. The introduced TD3-delay did not improve training performance.

This experiment concludes that the prefilling of the replay memory by delaying the critic optimization is essential for stable learning performance. The additional actor optimization delay may slightly improve the learning performance, but the effect is marginal.

C. Cloud-Edge Communication Latency

With the last experiments, we aim to analyze the sensitivity of the proposed architecture to the data bandwidth between the cloud and the edge. A constrained bandwidth reduces the actor update frequency, potentially changing learning performance. To this end, we artificially constrained the bandwidth by delaying actor packets accordingly. For the experiments, we selected bandwidths from $\{0.06, 0.1, 0.5, 5, 10, 15, >50\}$ Mbit/s which correspond to an actor update roughly every $\{681, 408, 81, 8, 4, 2, 1\}$ interaction steps. The lowest value, 0.06 Mbit/s, is the minimum bandwidth needed from the edge to the cloud to send state-action pairs continuously. The networked Ethernet setup in the previous experiments corresponds to >50 Mbit/s. The bandwidth values can be split into two groups, a low-bandwidth region ($\{0.06, 0.1, 0.5\}$) and a high-bandwidth region ($\{5, 10, 15, >50\}$). Since we suspected that combined experience replay (CER) should be affected by the bandwidth, we conducted all experiments with and without the usage of CER.

Figure 6 shows a clear trend in the high-bandwidth region that for lower bandwidths, average convergence time and variance increase. The convergence time average and variance do not increase further in the low-bandwidth region. This trend is similar with and without CER. A noteworthy observation is that CER only improves performance

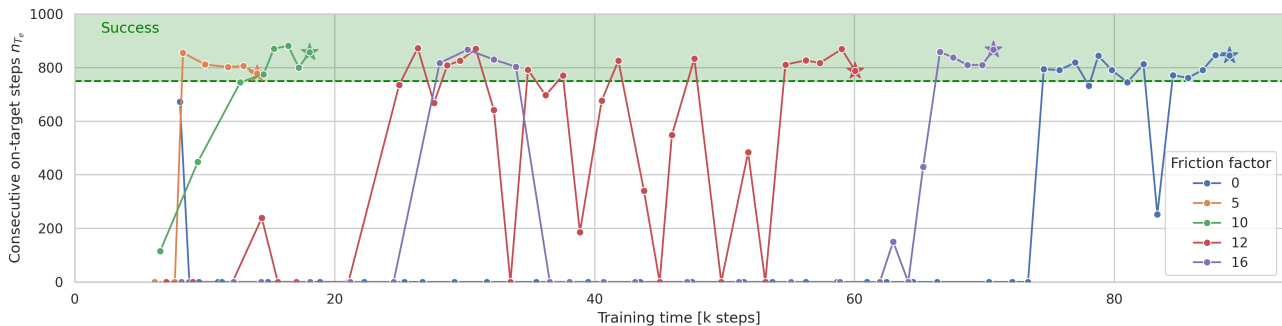


Fig. 4: Time-series plot showing the learning progress of five models pretrained with different friction factors, indicated by the color. Each data point corresponds to an evaluation episode with the horizontal axis’ cumulative training steps and the vertical axis’ consecutive on-target steps. Data points with more than 750 consecutive on-target steps are classified as a success. The fifth success data point in a row is highlighted with a star, indicating that the training converged.

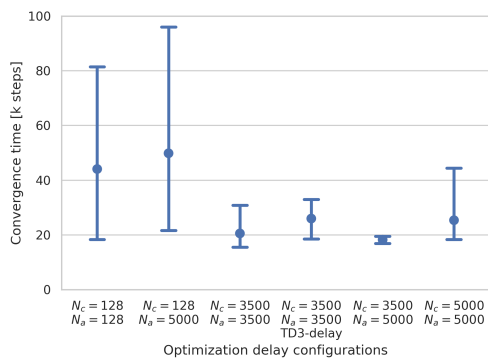


Fig. 5: Convergence time of different optimization delay configurations including one TD3-like actor delay implementation.

without a bandwidth constraint. For most other bandwidth constraints, CER decreases performance. We hypothesize that the on-policy behavior induced by CER biases the training process if the actor does not change every step. This hypothesis could explain the performance reduction for the other configurations.

This experiment concludes that the proposed architecture and transfer strategies perform best under high bandwidths from the cloud to the edge. However, agents can be trained within reasonable time even under highly constrained bandwidths. A side conclusion is that CER is only beneficial if the actor is updated on every interaction step. Otherwise, it is even detrimental, which is a significant limitation for the usage of CER.

VI. CONCLUSIONS AND FUTURE WORK

Training deep reinforcement learning agents on physical systems is challenging due to expensive data collection. Sim-to-real approaches train agents in simulations and directly deploy them to the physical system, suffering from performance losses induced by the *reality gap*. Sim-to-real approaches with continued training on the physical system

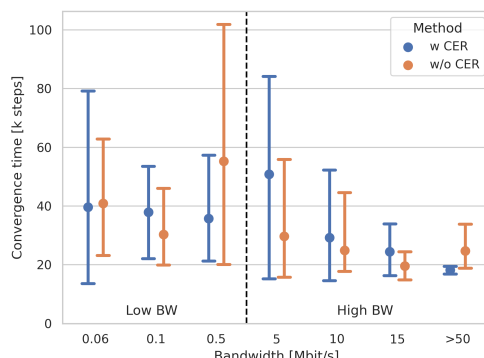


Fig. 6: Convergence time comparison of training instances with different bandwidth settings classified in low and high, trained with and without CER.

may be the best compromise of simulation and real-world training, which is why we adopt this strategy for this work.

In this work, we propose a distributed cloud-edge architecture to address the problem of training reinforcement learning agents on computationally constrained physical systems. We further propose sim-to-real transfer strategies with delayed optimization of the neural networks and introduce a one-step delay in simulation to force the agent to learn from its observation augmentation.

We evaluate the proposed methods with a linear inverted pendulum case study. We vary the *reality gap*, optimization delays, and cloud-edge bandwidth to analyze the performance of the proposed architecture. As expected, higher *reality gaps* lead to longer training time. However, it is still possible to train agents with high *reality gaps* repeatedly using the architecture. After the sim-to-real transfer, we identified a crucial replay memory prefill that stabilizes the learning process. The training is faster with higher bandwidths, but the architecture still allows training with very constrained bandwidths. An additional conclusion is that CER [25] is only beneficial if the actor can be updated every interaction step, and even detrimental if the updates happen

less frequently. A final observation is that with the cloud-edge architecture, the training process on the cloud is robust to crashes of the plant or edge.

Training data from physical systems often contains noise and disturbances. These faulty experiences can delay the entire training process. We will filter faulty experiences in future work to stabilize the training process further. Moreover, to reduce the sensitivity to the bandwidth, we will look into the usage of residual networks, where only smaller residual networks need to be trained and sent to the edge. Additionally, we will expand the architecture to other applications such as UAVs and use other state-of-the-art DRL algorithms.

ACKNOWLEDGMENT

Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

The authors would like to thank Daniele Bernardini, Andrea Bastoni, Alexander Züpke and Andres Rodrigo Zapata Rodriguez for helpful discussions.

REFERENCES

- [1] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," in *Proceedings of Robotics: Science and Systems*, (Freiburg/Breisgau, Germany), June 2019.
- [2] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.
- [3] H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar, "Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 3651–3657, IEEE, 2019.
- [4] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.
- [5] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, 2021.
- [6] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.
- [7] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, and V. Vanhoucke, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on Robot Learning*, pp. 651–673, PMLR, 2018.
- [8] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3389–3396, IEEE, 2017.
- [9] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.
- [10] Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani, "Data efficient reinforcement learning for legged robots," in *Conference on Robot Learning*, pp. 1–10, PMLR, 2020.
- [11] M. Veerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *arXiv preprint arXiv:1707.08817*, 2017.
- [12] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6023–6029, IEEE, 2019.
- [13] M. Neunert, T. Boaventura, and J. Buchli, "Why off-the-shelf physics simulators fail in evaluating feedback controller performance—a case study for quadrupedal robots," in *Advances in Cooperative Robotics*, pp. 464–472, World Scientific, 2017.
- [14] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Proceedings of Robotics: Science and Systems*, (Pittsburgh, Pennsylvania), June 2018.
- [15] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3803–3810, IEEE, 2018.
- [16] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," in *Proceedings of Robotics: Science and Systems*, (Cambridge, Massachusetts), July 2017.
- [17] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.
- [18] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12627–12637, 2019.
- [19] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.
- [20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.
- [21] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*, pp. 1587–1596, PMLR, 2018.
- [22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [23] L.-J. Lin, "Reinforcement learning for robots using neural networks," tech. rep., Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.
- [24] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.
- [25] S. Zhang and R. S. Sutton, "A deeper look at experience replay," *arXiv preprint arXiv:1712.01275*, 2017.
- [26] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [27] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*, vol. 24. Springer Science & Business Media, 2011.
- [28] D. Seto and L. Sha, "A case study on analytical analysis of the inverted pendulum real-time control system," tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1999.
- [29] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.
- [30] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.
- [31] "Quanser courseware and resources." <https://www.quanser.com/solution/control-systems/>.
- [32] J. Carlson, *Redis in action*. Simon and Schuster, 2013.
- [33] P. F. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *CoRR*, vol. abs/1610.03518, 2016.

5.2 Learning to Generate All Feasible Actions

Reference

M. Theile, D. Bernardini, R. Trumpp, C. Piazza, M. Caccamo, and A. L. Sangiovanni-Vincentelli, “Learning to generate all feasible actions,” *IEEE Access*, vol. 12, pp. 40668–40681, 2024

DOI: <https://doi.org/10.1109/ACCESS.2024.3376739>

Abstract

Modern cyber-physical systems are becoming increasingly complex to model, thus motivating data-driven techniques such as reinforcement learning (RL) to find appropriate control agents. However, most systems are subject to hard constraints such as safety or operational bounds. Typically, to learn to satisfy these constraints, the agent must violate them systematically, which is computationally prohibitive in most systems. Recent efforts aim to utilize feasibility models that assess whether a proposed action is feasible to avoid applying the agent’s infeasible action proposals to the system. However, these efforts focus on guaranteeing constraint satisfaction rather than the agent’s learning efficiency. To improve the learning process, we introduce action mapping, a novel approach that divides the learning process into two steps: first learn feasibility and subsequently, the objective by mapping actions into the sets of feasible actions. This paper focuses on the feasibility part by learning to generate all feasible actions through self-supervised querying of the feasibility model. We train the agent by formulating the problem as a distribution matching problem and deriving gradient estimators for different divergences. Through an illustrative example, a robotic path planning scenario, and a robotic grasping simulation, we demonstrate the agent’s proficiency in generating actions across disconnected feasible action sets. By addressing the feasibility step, this paper makes it possible to focus future work on the objective part of action mapping, paving the way for an RL framework that is both safe and efficient.

Contributions to this paper

- Conceptualization of the action mapping framework
- Shared derivation of the various gradient estimators
- Implementation of the algorithms and case studies
- Evaluation of the algorithm
- Majority of paper writing

Copyright

Creative Commons License – CC BY 4.0 DEED

<https://creativecommons.org/licenses/by/4.0/>

See Appendix A.8 for the reuse statement. The following shows the accepted version.

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.3376739

Learning to Generate All Feasible Actions

MIRCO THEILE^{1,2}, (Student Member, IEEE), DANIELE BERNARDINI^{1,3}, (Member, IEEE),
RAPHAEL TRUMPP¹, (Student Member, IEEE), CRISTINA PIAZZA³, (Senior Member, IEEE),
MARCO CACCAMO¹, (Fellow, IEEE), ALBERTO L. SANGIOVANNI-VINCENTELLI², (Fellow, IEEE)

¹TUM School of Engineering and Design, Technical University of Munich (e-mail: {mirco.theile,daniele.bernardini,raphael.trumpp,mcaccamo}@tum.de)

²Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley (e-mail: alberto@berkeley.edu)

³TUM School of Computation, Information and Technology, Technical University of Munich (e-mail: cristina.piazza@tum.de)

Corresponding author: Mirco Theile (e-mail: mirco.theile@tum.de).

Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

ABSTRACT Modern cyber-physical systems are becoming increasingly complex to model, thus motivating data-driven techniques such as reinforcement learning (RL) to find appropriate control agents. However, most systems are subject to hard constraints such as safety or operational bounds. Typically, to learn to satisfy these constraints, the agent must violate them systematically, which is computationally prohibitive in most systems. Recent efforts aim to utilize feasibility models that assess whether a proposed action is feasible to avoid applying the agent's infeasible action proposals to the system. However, these efforts focus on guaranteeing constraint satisfaction rather than the agent's learning efficiency. To improve the learning process, we introduce *action mapping*, a novel approach that divides the learning process into two steps: first learn feasibility and subsequently, the objective by mapping actions into the sets of feasible actions. This paper focuses on the feasibility part by *learning to generate all feasible actions* through self-supervised querying of the feasibility model. We train the agent by formulating the problem as a distribution matching problem and deriving gradient estimators for different divergences. Through an illustrative example, a robotic path planning scenario, and a robotic grasping simulation, we demonstrate the agent's proficiency in generating actions across disconnected feasible action sets. By addressing the feasibility step, this paper makes it possible to focus future work on the objective part of action mapping, paving the way for an RL framework that is both safe and efficient.

INDEX TERMS action mapping, feasibility, generative neural network, self-supervised learning

I. INTRODUCTION

Cyber-physical systems are becoming increasingly complex, with applications ranging from autonomous vehicles in chaotic urban environments to robotic assistants for support in everyday tasks. Most of these applications require the development of complex control systems. Traditionally, these systems were modeled in detail, and control strategies were derived using model-based techniques. However, the increasing complexity of these systems limits the applicability of model-based techniques, thus making data-driven techniques appealing. While data-driven techniques such as reinforcement learning (RL) improved significantly in recent years, they still lack guarantees that they meet all system constraints, i.e., only providing *feasible* control commands.

A popular idea is deriving only the feasibility-relevant part of the system to ensure feasibility while using learning techniques to optimize the underlying objective. The feasibility

model only delineates whether a suggested control command in a given situation is feasible, i.e., the control command does not violate any constraints and does not lead to a state from which a future constraint violation is inevitable. Given this feasibility model, the subsequent challenge is integrating it within a learning framework in which a policy aims to optimize an objective function subject to feasibility constraints. The commonly applied techniques are *action rejection*, *resampling*, and *action projection*.

Action rejection is a traditional approach, e.g., applied in the Simplex architecture [1], which can be summarized as follows. If the policy's proposed action is feasible according to the feasibility model, it is applied to the system. Otherwise, a backup policy is used, which generates a feasible action, usually independent of the objective. While this is the simplest method to implement, and the timing requirements are predictable, the drawback is that the policy needs to learn the

feasibility model explicitly to avoid its action being rejected and replaced with the sub-optimal backup action.

As a straightforward augmentation of the action rejection scheme, action resampling can be applied when training a stochastic policy. Instead of directly switching to the safe action, if the proposed action is infeasible, the policy can be resampled, and the newly generated action can be tested [2]. This process can be repeated until either a feasible action is proposed or a timeout is reached, at which point the safe action of the feasibility controller is applied to the system. While this method may decrease the rejection rate of the policy's actions, it adds computational costs. Additionally, most learning methods train agents that output a reparameterization of a single Gaussian. Resampling from this Gaussian may not offer a feasible action if it is too narrow or poorly aligned with the set of feasible actions. Moreover, the learning agent must still explicitly learn to avoid proposing infeasible actions.

A more nuanced method is action projection [3], which replaces a proposed infeasible action with a feasible action closest to the proposed action. This projection is typically formulated as an optimization problem that must be solved online. The supposed advantage of this method over action rejection is that the replacement action is better than the safe action, which was derived independently of the objective. However, only because the projected action is *close in the action space* does not mean it is also *close in performance*. Additionally, the online optimization requirement may not be computationally feasible, especially for complex systems. From a learning perspective, the projection can either be penalized or ignored. If penalized, the agent again needs to learn explicitly to avoid infeasible actions, but it could receive guidance from the projection distance. If the agent does not penalize infeasible actions, the agent is not required to learn the feasibility model. However, the projection to the closest feasible action will map all infeasible actions to the borders of the feasible action sets. Learning algorithms that require action densities or policy gradients must be adapted to handle the resulting high action density on the borders.

In all three approaches, the learning agent that aims to find an optimum of the objective subject to the feasibility constraints is not aided by the feasibility model; it is solely made safe. The agent must still violate the constraints systematically during interactions with the environment, albeit without actually applying infeasible actions to the system, to learn to satisfy them in the future. We introduce a different approach that allows the learning agent to benefit explicitly from the model-based feasibility model. We call the approach *action mapping*. The idea is to learn the feasibility and the objective consecutively. First, a *feasibility policy* is trained to generate all feasible actions for a given state. Using this feasibility policy, an *objective policy* can learn to choose the optimal action from the feasible ones, given an objective. Note that the optimization problem in the feasible actions could be solved with various methods, including, but not limited to, learning, which can all benefit from the guarantee of constraint satisfaction.

This methodology promises multiple potential advantages. First, the feasibility policy can be trained directly on the feasibility model, requiring no interactions with the environment. Afterward, the objective policy learns to choose among feasible actions, which could significantly reduce the number of interactions with the environment. The combined agent, i.e., feasibility plus objective policy, still needs to exhaustively violate constraints. However, it can learn constraint satisfaction offline from the feasibility model without interactions with the environment. Second, the feasibility policy can be reused if multiple objectives are subject to the same constraints. Third, any knowledge of the environment that can be extracted from the feasibility model can potentially be utilized in the objective policy through parameter sharing between both policies. Lastly, once deployed, it requires precisely one pass of the feasibility policy and the objective policy per step if the feasibility policy has no support in the infeasible action space.

Given these potential advantages, the pivotal question is: How do we train the feasibility policy? This paper endeavors to answer this very question. To this end, we derive the objective of the feasibility policy as a distribution matching problem in which the target is a uniform distribution over the feasible action space. The uniform distribution is chosen since the feasibility policy is agnostic to the objective and should thus not be biased toward specific actions. We further present a methodology for estimating the gradient of different divergence measures to train a feasibility policy toward the target distribution. To evaluate our proposed methodology, we perform three experiments. The first is an illustrative example with an analytical and highly parallelizable feasibility function that shows the input and output of the feasibility policy. The second example illustrates how the feasibility policy can learn to generate feasible trajectory segments for robotic path planning problems, providing a closer tie to reinforcement learning. The third experiment showcases a simple robotic grasping example where feasibility is defined as grasping poses that lead to a successful grasp. This experiment shows how a feasibility policy can be learned for systems without a feasibility model that can be efficiently parallelized.

The contributions of this work are the following:

- Conceptualization of *action mapping* as a framework for safe and efficient reinforcement learning;
- Formulation of a distribution matching problem to train the feasibility policy towards generating all feasible actions;
- Derivation of gradient estimators for different divergence measures utilizing kernel density estimates, resampling, and importance sampling;
- Evaluation of the proposed approach in an illustrative 2D example, a qualitative example for spline-based path planning, and a quantitative planar robotic grasping example.

The remainder of this paper is structured as follows. Section II discusses related work. Section III describes the action mapping motivation and the formulation as a distribution

matching problem, followed by the gradient estimation in Section IV. Section V provides an illustrative example to visualize the feasibility policy and Section VI provides an additional example that showcases how action mapping could be used in robotic path planning problems. Sections VII and VIII introduce and discuss the robotic grasping experiments.

II. RELATED WORK

In discrete action spaces, the equivalent of action mapping is action masking, for which the feasibility of each action is evaluated, and the agent chooses the best action among the feasible ones. In [4], the action masking concept is termed *shielding*, in which the shield is based on linear temporal logic. The authors in [5] investigate the consequences of action masking for policy gradient deep reinforcement learning (DRL) algorithms. Applications in various domains show significant performance improvements, e.g., in autonomous driving [6], unmanned aerial vehicle (UAV) path planning [7], and vehicle routing [8].

For continuous action spaces, a straightforward masking approach is not yet available. As discussed before, the approaches can be grouped into *action rejection*, *resampling* [9], and *action projection* [10], [11], [12]. The safety model can be based on control barrier functions [13], Lyapunov functions [14], or variants thereof. Cheng et al. [3] use action projection and train a second model on the previous interventions to reduce the need for future interventions. Zhong et al. [15] derive a *safe-visor* that rejects infeasible actions proposed by the agent and replaces it with a safe action.

The distribution matching problem is similar to posterior sampling, a long-standing problem in statistics. State-of-the-art methods in Bayesian statistics rely on Markov Chain Monte Carlo (MCMC) algorithms [16], [17], eliminating the need to normalize the distribution, which is often an intractable problem [18]. Variational Inference (VI) relies instead on fitting the posterior with a family of parametric probability distributions that can be sampled [19], [20]. Neural samplers offer another alternative by approximating the posterior with a generative neural network [21], [22].

Normalizing Flows (NFs) infer the probability density function (pdf) for each sample using invertible mappings [23], [24], [25]. While NFs do not require density estimates, they have been shown to require a prohibitive number of layers to effectively match a target distribution in more than one dimension [26]. However, the depth of such models can lead to challenges like vanishing or exploding gradients, which are even exacerbated by the inherent conditioning difficulties of NFs [27].

For robotic grasping, the authors in [28] propose using DRL to find optimal grasps through interaction with multiple real-world robots. If the goal is to find grasping poses explicitly to be used as the target of a classical controller, supervised learning techniques are often utilized [29]. To support various downstream tasks, it would be necessary to find all feasible grasps. To this end, the action space is typically discretized, and grasping success is estimated for each discrete action

through heat-maps. This can be learned using supervised [30], [31] or self-supervised [32] methods. [32] explicitly utilizes the structure given by spatial equivariances. We aim to find a solution that needs neither discretization nor the use of the structure, as these requirements are specific to grasping and also restrict applicability to planar picking in carefully crafted environments.

III. OPTIMIZATION PROBLEM

A. ACTION MAPPING

For a state space \mathcal{S} and an action space \mathcal{A} , the feasibility model can be expressed through the function

$$g : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{B}, \quad (1)$$

which delineates if a suggested action is feasible in a given state. Given g , the state-dependent set of feasible actions $\mathcal{A}_s^+ \subseteq \mathcal{A}$ contains all actions that are feasible for the state s , i.e., all actions for which $g(s, a) = 1$.

For action mapping, the feasibility policy is defined as

$$\pi_{\text{feasibility}} : \mathcal{S} \times \mathcal{Z} \rightarrow \mathcal{A}_s^+. \quad (2)$$

It learns a state-conditioned surjective map from a bounded latent space $\mathcal{Z} \subset \mathbb{R}^m$, with appropriate dimensionality m , into the set of feasible actions for that state. The latent space \mathcal{Z} can be thought of as an infinite set of *indices*. For each *index*, the feasibility policy has to output a different feasible action.

Given the task specifics, an objective policy can be defined that learns the optimal latent value as

$$\pi_{\text{objective}} : \mathcal{S} \rightarrow \mathcal{Z}. \quad (3)$$

This optimal *index* in the latent space can then be mapped to a feasible action using $\pi_{\text{feasibility}}$. Convolving the functions as $(\pi_{\text{feasibility}} \circ \pi_{\text{objective}}) : \mathcal{S} \rightarrow \mathcal{A}_s^+$, yields the action mapping policy

$$\pi(s) = \pi_{\text{feasibility}}(s, \pi_{\text{objective}}(s)). \quad (4)$$

In this work, we derive how to train the feasibility policy. Since this work only concerns the feasibility policy, the subscript is dropped in the following.

B. FEASIBILITY POLICY

To train the feasibility policy π_θ , we parameterize it with parameters θ and formulate a distribution matching problem. The goal is that π_θ maps every $z \in \mathcal{Z}$ to an $a \in \mathcal{A}_s^+$, without any bias toward any specific feasible actions. Therefore, by sampling uniformly in \mathcal{Z} , π_θ should generate a uniform distribution in \mathcal{A}_s^+ .

When sampling uniformly in \mathcal{Z} , π_θ becomes a generator with a conditional probability density function (pdf) $q_\theta(a|s)$. The target distribution is the uniform distribution in the feasible action space given as

$$p(a|s) = \frac{g(s, a)}{\int_{\mathcal{A}} g(s, a') da'}. \quad (5)$$

TABLE 1: Non-exhaustive list of f-divergences and the corresponding first derivative for gradient estimators.

	$f(t)$	$f'(t)$
JS	$\frac{1}{2} \left[(t+1) \log\left(\frac{2}{t+1}\right) + t \log(t) \right]$	$\frac{1}{2} \log\left(\frac{2t}{t+1}\right)$
FKL	$-\log(t)$	$-\frac{1}{t}$
RKL	$t \log(t)$	$\log(t) + 1$

The f-divergences are obtained by substituting the f functions above in (7) and setting $t = q_\theta/p$. The conventions for p, q , FKL and RKL assume that p is the target distribution, q is the model, and the FKL divergence is $\int p \log(p/q)$.

Given a divergence measure \mathcal{D} , the optimal parameters are the solution to the optimization problem

$$\operatorname{argmin}_{\theta \in \Theta} \int_{\mathcal{S}} \mathcal{D}(p(\cdot|s) || q_\theta(\cdot|s)) ds, \quad (6)$$

with Θ being the set of possible parameters. The following section details how to iteratively minimize the divergence.

IV. METHODOLOGY

The following derives the gradient w.r.t. θ to iteratively minimize the divergence for a given state. For simplicity of notation, we omit the state and action dependence of q_θ and p .

A. F-DIVERGENCE

As the divergence measure, we choose the f-divergence, a generalization of the Kullback-Leibler (KL) divergence ([33]). The f-divergence between two pdfs p and q_θ has the form

$$\mathcal{D}_f(p || q_\theta) = \int_{\mathcal{A}} p f\left(\frac{q_\theta}{p}\right) da, \quad (7)$$

where $f : (0, \infty) \rightarrow \mathbb{R}$ is a convex function. Different choices of f lead to well-known divergences as summarized in Table 1. The gradients of the f-divergence w.r.t. θ can be estimated commuting the derivative with the integral ([34]) and using the score function gradient estimator ([35]) as

$$\begin{aligned} \frac{\partial}{\partial \theta} \mathcal{D}_f &= \frac{\partial}{\partial \theta} \int_{\mathcal{A}} p f\left(\frac{q_\theta}{p}\right) da \\ &= \int_{\mathcal{A}} p f'\left(\frac{q_\theta}{p}\right) \frac{1}{p} \frac{\partial}{\partial \theta} q_\theta da \\ &= \int_{\mathcal{A}} q_\theta f'\left(\frac{q_\theta}{p}\right) \frac{\partial}{\partial \theta} \log q_\theta da, \end{aligned} \quad (8)$$

considering that p does not depend on θ . Since q_θ is normalized to 1 and thus $\partial_\theta \int_{\mathcal{A}} q da = \int_{\mathcal{A}} q \partial_\theta \log q da = 0$, a Lagrangian term λ can be added to the gradient:

$$\frac{\partial}{\partial \theta} \mathcal{D}_f = \int_{\mathcal{A}} q_\theta \left(f'\left(\frac{q_\theta}{p}\right) + \lambda \right) \frac{\partial}{\partial \theta} \log q_\theta da. \quad (9)$$

If the support of q_θ includes all of \mathcal{A} the above formula in (9) can be rewritten as the expectation on q_θ as

$$\frac{\partial}{\partial \theta} \mathcal{D}_f = \mathbb{E}_{q_\theta} \left[\left(f'\left(\frac{q_\theta}{p}\right) + \lambda \right) \frac{\partial}{\partial \theta} \log q_\theta \right]. \quad (10)$$

Alternatively, using a proposal distribution q' with full support in \mathcal{A} , the expectation in (10) can be reformulated as

$$\frac{\partial}{\partial \theta} \mathcal{D}_f = \mathbb{E}_{q'} \left[\frac{q_\theta}{q'} \left(f'\left(\frac{q_\theta}{p}\right) + \lambda \right) \frac{\partial}{\partial \theta} \log q_\theta \right]. \quad (11)$$

B. GRADIENT ESTIMATION

Given a sample $a \sim q_\theta$, it is not possible to directly evaluate $q_\theta(a)$ as it is not available in closed form. Therefore, q_θ needs to be estimated to compute the gradients of the f-divergence. Given N sampled actions $a_i \sim q_\theta$, q_θ can be approximated with a Kernel Density Estimation (KDE) by

$$q_\theta(a) \approx \hat{q}_{\theta, \sigma}(a) = \frac{1}{N} \sum_{a_i \sim q_\theta} k_\sigma(a - a_i), \quad (12)$$

where k_σ is a Gaussian kernel with a diagonal bandwidth matrix σ . The KDE enables the estimation of the expectation. Using (10), computing the expectation value as the average over the samples yields

$$\frac{\partial}{\partial \theta} \mathcal{D}_f \approx \frac{1}{N} \sum_{a_i \sim q_\theta} \left(f'\left(\frac{\hat{q}_{\theta, \sigma}}{p}\right) + \lambda \right) \frac{\partial}{\partial \theta} \log \hat{q}_{\theta, \sigma}. \quad (13)$$

The gradient estimator in (13) did not converge in our experiments. While a systematic investigation of the convergence issue was not completed, we suspect two primary reasons. First, the support q_θ usually does not cover the whole action space \mathcal{A} , which is necessary for the expectation formulation in (10). Second, evaluating $q_\theta(a_i)$ based on a KDE, which uses a_j as supports, has a bias for $j = i$.

Adding Gaussian noise to the samples gives full support in \mathcal{A} and reduces the bias at the support points of the KDE, which led to convergence in the experiments. The new samples are given by $a_j^* = a_i + \epsilon$ for $m_i \leq j < m(i+1)$ and $\epsilon \sim \mathcal{N}(0, \sigma')$, where m indicates the number of samples drawn for each original sample. This is equivalent to sampling from a KDE with a_i as supports and σ' as bandwidth. Using importance sampling in (11), the gradient in (13) after resampling can be rewritten as follows

$$\begin{aligned} \frac{\partial}{\partial \theta} \mathcal{D}_f &\approx \\ \frac{1}{M} \sum_{a_j^* \sim \hat{q}_{\theta, \sigma'}} \frac{\hat{q}_{\theta, \sigma}}{\hat{q}_{\theta, \sigma'}} \left(f'\left(\frac{\hat{q}_{\theta, \sigma}}{p}\right) + \lambda \right) \frac{\partial}{\partial \theta} \log \hat{q}_{\theta, \sigma}, \end{aligned} \quad (14)$$

with $M = mN$. Additionally, equation (14) requires an estimate of p , which in turn requires an estimate of the volume in (5)

$$\int_{\mathcal{A}} g(a) da \approx \frac{1}{M} \sum_{a_j^*} \frac{g(a_j^*)}{\hat{q}_{\theta, \sigma'}(a_j^*)}. \quad (15)$$

This volume estimation in (15) is similar to self-normalized importance sampling ([36]) but uses the proposal distribution. The bandwidth σ' of the proposal distribution is a hyper-parameter. Setting $\sigma' = c \sigma$, experiments show that in most cases $c > 1$ helps convergence. Intuitively, a larger bandwidth enables the exploration of nearby modes in the action space. Specific estimators for the different f-divergences can be obtained by substituting f' from Table 1 into (14). A summary of the gradient estimators used in this work is given in Table 2.

TABLE 2: Gradient estimators of various losses and choice of Lagrangian multiplier λ .

Loss	Actor Gradient Estimator	λ
JS	$\frac{1}{2M} \sum_{a_j^*} \frac{\hat{q}_{\theta,\sigma}}{\hat{q}_{\theta,\sigma'}} \log \left(\frac{2\hat{q}_{\theta,\sigma}}{p+\hat{q}_{\theta,\sigma}} \right) \frac{\partial}{\partial \theta} \log \hat{q}_{\theta,\sigma}$	0
FKL	$-\frac{1}{M} \sum_{a_j^*} \frac{p}{\hat{q}_{\theta,\sigma'}} \frac{\partial}{\partial \theta} \log \hat{q}_{\theta,\sigma}$	0
RKL	$\frac{1}{M} \sum_{a_j^*} \frac{\hat{q}_{\theta,\sigma}}{\hat{q}_{\theta,\sigma'}} \log \left(\frac{\hat{q}_{\theta,\sigma}}{p} \right) \frac{\partial}{\partial \theta} \log \hat{q}_{\theta,\sigma}$	-1
GAN	$\frac{1}{N} \sum_{a_i} \frac{\partial}{\partial a} \log(1 - \xi_\phi) \frac{\partial}{\partial \theta} a_i$	-
ME	$\frac{1}{N} \sum_{a_i} \frac{\partial}{\partial \theta} \log \hat{q}_{\theta,\sigma} - \frac{\partial}{\partial a} \log \xi_\phi \frac{\partial}{\partial \theta} a_i$	-

C. TRAINING PROCESS

Algorithm 1 shows a training loop when training a feasibility policy directly on the feasibility model using a Jensen-Shannon (JS) loss. The training iterates as follows: A batch of random states is sampled, and the actor generates N actions a_i per state. For each action a_i , m values are sampled from a normal distribution $\mathcal{N}(0, \sigma')$ and added to the action values to create the M action samples a_j^* . Using the actions a_i as support of the KDE in (12), the densities $\hat{q}_{\theta,\sigma}(a_j^*)$ and $\hat{q}_{\theta,\sigma'}(a_j^*)$ are computed. Then the feasibility model g is evaluated on all samples a_j^* and the estimate of $p(a_j^*)$ is computed using (5) and importance sampling in (15). Finally, the gradient of θ can be computed according to (14). For a better understanding of the gradient, the trace of the gradient is highlighted in red throughout the algorithm.

Algorithm 1: Jensen-Shannon training loop

```

1 Initialize  $\theta$ 
2 for 1 to Training Steps do
3   for  $k = 1$  to  $K$  do
4      $s_k \leftarrow$  Sample from  $\mathcal{S}$ 
5      $z_i \leftarrow$  Sample uniformly in  $\mathcal{Z}, \forall i \in [1, N]$ 
6      $a_i \leftarrow \pi_\theta(s_k, z_i), \forall i \in [1, N]$ 
7      $\epsilon_j \sim \mathcal{N}(0, \sigma'), \forall j \in [1, M]$ 
8      $a_j^* \leftarrow \text{stop\_gradient}(a_{[j/m]}) + \epsilon_j, \forall j \in [1, M]$ 
           // Resample from KDE
9      $\hat{q}_j \leftarrow \frac{1}{N} \sum_{i=1}^N k_\sigma(a_j^* - a_i), \forall j \in [1, M]$ 
           // Evaluate KDE on samples
10     $\hat{q}'_j \leftarrow \frac{1}{N} \sum_{i=1}^N k_{\sigma'}(a_j^* - a_i), \forall j \in [1, M]$ 
           // Evaluate proposal pdf
11     $\hat{r}_j \leftarrow g(s_k, a_j^*), \forall j \in [1, M]$  // Evaluate
           feasibility model on samples
12     $\hat{V} \leftarrow \frac{1}{M} \sum_{j=1}^M \frac{\hat{r}_j}{\hat{q}'_j}$  // MC integration
           with importance sampling
13     $\hat{p}_j \leftarrow \frac{\hat{r}_j}{\hat{V}}, \forall j \in [1, M]$ 
14     $g_k \leftarrow \frac{1}{2M} \sum_{j=1}^M \frac{\hat{q}_j}{\hat{q}_j + \hat{p}_j} \log \left( \frac{2\hat{q}_j}{\hat{q}_j + \hat{p}_j} \right) \nabla_\theta \log(\hat{q}_j)$ 
           // gradient trace
15  end
16   $\theta \leftarrow \theta - \alpha_\theta \frac{1}{K} \sum_{k=1}^K g_k$ 
17 end

```

Intuitively, the gradient in (14) attracts support actions a_i towards sample actions a_j^* where $p(a_j^*) > \hat{q}_{\theta,\sigma}(a_j^*)$ and repulses support actions from samples where $p(a_j^*) < \hat{q}_{\theta,\sigma}(a_j^*)$. The different f-divergences place different weights on attraction and repulsion. FKL only attracts support actions towards samples with high p , while RKL repulses strongly from samples with $p = 0$, and JS attracts and repulses with lower magnitude.

D. ACTOR-CRITIC

Algorithm 1 assumes that the training can be performed directly on the feasibility model. However, multiple actions must be evaluated for the same state to train the actor. This is possible if g is available in closed form or effectively simulated. In some scenarios, g can be a real experiment that does not allow reproducibility of states. To mitigate this problem, an auxiliary neural network $\xi_\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ with parameters ϕ can be trained to imitate the environment g . The policy can then be trained to match the distribution of feasible actions according to this auxiliary neural network. We refer to π_θ and ξ_ϕ as actor and critic, respectively.

The actor and critic can be trained simultaneously. The critic is trained on data from a replay memory collected through interactions between the actor and the environment, with each training batch containing half feasible actions and half infeasible actions to stabilize training. To further improve the training efficiency of the critic, when the actor interacts with the environment, it suggests multiple actions, which the critic evaluates. The action with the highest uncertainty, i.e., the action with $\xi \approx 0.5$ is selected as it contains the most information for the critic. We call this process *maximum uncertainty sampling*. During evaluation, to improve the precision of the actor, the critic can be evaluated on proposed actions, and actions with low values can be rejected. This action optimization can increase precision but may reduce recall or the ability to find all the disconnected sets of feasible actions.

V. ILLUSTRATIVE EXAMPLE

This section provides illustrative examples to elucidate the feasibility policy and demonstrates the potential for direct training on a parallelizable feasibility model across multiple actions for a given state. Hyperparameters, their ranges, and training and inference times are summarized in Table 3.

A. PROBLEM

Consider three circles with given radii and center points as the state s . The feasibility model g deems any point \mathbf{a} a feasible action if it falls within at least one circle and lies inside a unit square, described as follows: $s = (\mathbf{c}_k, r_k)_{k \in \{1,2,3\}}$ where (\mathbf{c}_k, r_k) are the center points and radii of the circles, and $\mathbf{a} \in \mathbb{R}^2$ represents a coordinate. The feasibility model is thus expressed by

$$g(s, \mathbf{a}) = (\mathbf{0} \leq \mathbf{a} \leq \mathbf{1}) \wedge \bigvee_{k=1}^3 (|\mathbf{a} - \mathbf{c}_k| < r_k). \quad (16)$$

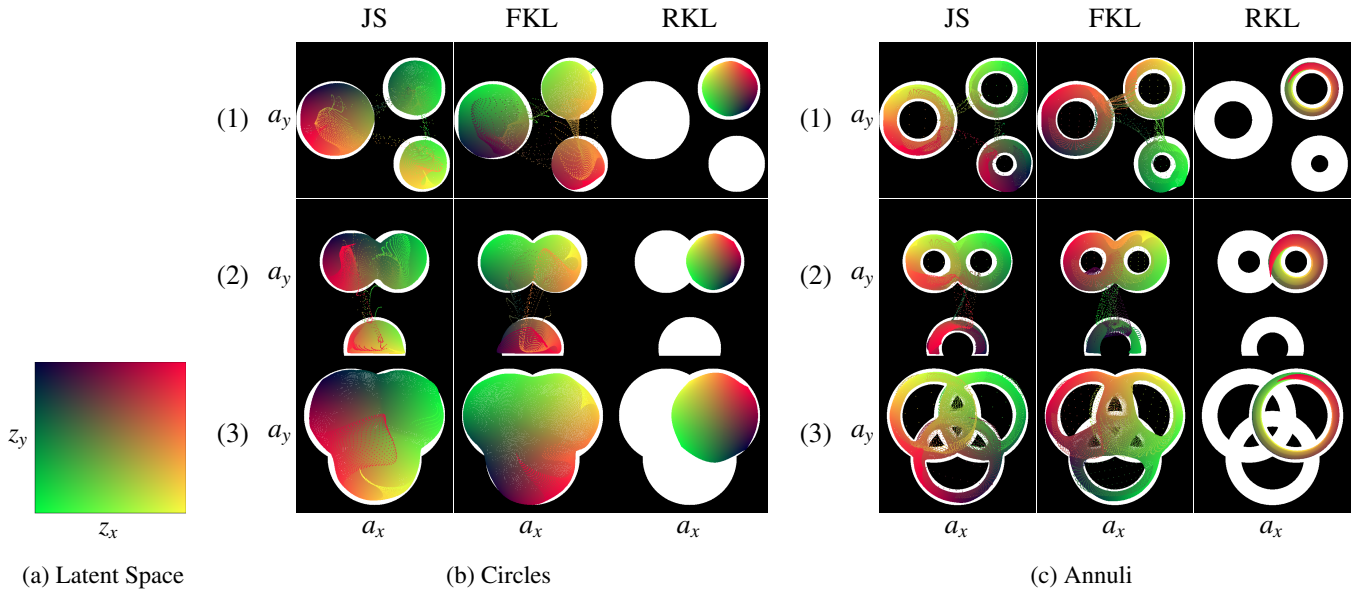


FIGURE 1: Illustrative example showing two feasibility models, which specify feasible regions as the union of three random circles (b) or annuli (c). Three states (1)-(3) are shown for each example, solved with the JS, FKL, and RKL divergence, with feasible and infeasible action space in white and black, respectively. The colored points are actions generated by the feasibility policy when using the corresponding latent space value $(z_x, z_y) \in \mathcal{Z}$ in (a).

In the extended example, each circle includes an inner radius, forming annular regions.

B. RESULTS

Figure 1 illustrates the outcomes of applying three distinct divergences, JS, FKL, and RKL, to the circle and annulus scenarios, depicted in subfigures (b) and (c), respectively. Actions are generated from a grid of 256^2 latent values shown in subfigure (a), where each color corresponds to a specific latent value. Three states, marked as (1), (2), and (3), represent various configurations: disconnected shapes, partially connected shapes, and fully connected shapes. The figure visually underscores the different outcomes using the divergences: the RKL approach tends to focus on singular modes, even failing to span overlapping regions, as seen in the third row of both (b) and (c). On the contrary, both FKL and JS exhibit a more expansive coverage, approaching the borders of the feasible space, indicated by the white regions, with the JS divergence showing a reduced density within the infeasible space, represented by the black regions, as compared to FKL. This phenomenon is particularly evident in the first and second states for the circle and annulus examples, which can be attributed to the repulsive gradient present in JS divergence that is absent in the FKL divergence.

These visualizations show that a feasibility policy can be trained to navigate complex distributions beyond the Gaussian reparameterization commonly found in the literature. They further elucidate the importance of enabling the FKL and JS divergences to address disconnected feasible sets effectively. Ultimately, these examples offer an intuitive comprehension of the aim: for the feasibility policy to generate all feasible

actions by learning to map the latent space into diverse shapes conditioned on the state.

VI. FEASIBLE TRAJECTORY SEGMENTS EXAMPLE

When solving problems in robotic path planning with reinforcement learning, a standard action space is the direction and velocity target of the robot. However, in tasks that span a long time horizon, it can be beneficial to reduce the number of actions by bundling multiple actions in parametric trajectory segments, often splines, to be followed. Another benefit of generating splines is that these can be checked for collisions with obstacles and other system constraints, such as maximum curvature. This application example shows how learning all feasible actions could be used in this context.

A. PROBLEM

Consider a stationary agent at the center of an environment with known obstacles. In this example, the objective is to find all quadratic splines that fulfill the following conditions

- 1) does not intersect with any obstacle;
- 2) longer than a minimum length;
- 3) shorter than a maximum length;
- 4) its maximal curvature is less than a threshold.

Figure 2a shows an example scenario with randomly generated obstacles (gray) and example splines. For each constraint, the figure shows an example that violates it, additionally providing examples of feasible splines. The splines are parameterized through the endpoint and an intermediary point that bends the spline, yielding a 4D action space. The feasibility model checks for any constraint violation numerically along the spline. The agent observes the obstacles as a black and white

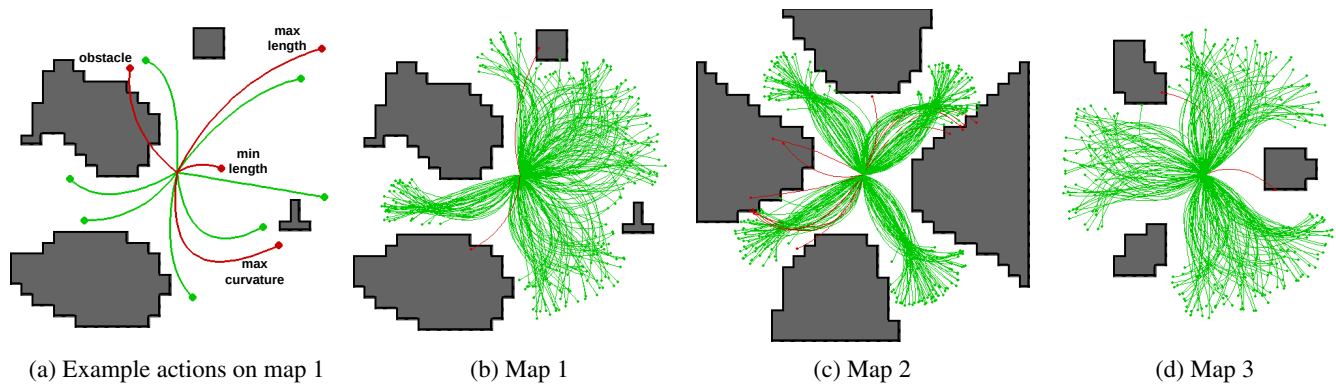


FIGURE 2: Quadratic spline action space application showing three different maps: a randomly generated map in (a) and (b) and two handcrafted maps in (c) and (d). In (a), example splines are shown with green indicating a feasible spline and red indicating an infeasible one. An example for each constraint violation is given. In (b)-(d), the agent generates 256 actions that are displayed with the color depending on the feasibility of each proposed action.

image with size 31×31 . It is trained with the JS loss on randomly generated obstacle maps and evaluated on maps not seen during training. The parameters for training, and training and inference times are given in Table 3.

B. RESULTS

Figure 2 shows three example obstacle maps and action samples from the agent. In Figure 2b, the agent provides 256 splines for the randomly generated map, among which 254 are feasible. On the right side of the map, with only two smaller obstacles, the agent produces a wide range of splines that avoid the two obstacles, with a larger margin toward the bottom obstacle. The left side of the map, with larger obstacles and only a smaller gap for feasible paths, shows that the agent also produced a group of splines. Given the minimum length constraint on the splines, the splines going to the left are disconnected from the splines on the right, considering the parameter space. The two infeasible splines generated by the agent are likely to be on the transition boundary between these disconnected sets of feasible splines.

Map 2 in Figure 2c shows a situation that contains four disconnected sets of feasible splines, one in each diagonal direction. The agent produces feasible splines in each direction, though generating more infeasible splines. This is likely due to the difficulty of generating four relatively small disconnected sets separated by large volumes of infeasible action space. Map 3 in Figure 2d shows a simpler problem with three small obstacles resulting in three disconnected sets of feasible splines. In this example, the agent again generated 254 feasible splines in all three sets with only two splines when transitioning between sets. Overall, the agent can generate splines in all disconnected sets, largely avoiding generating infeasible splines.

This example shows how action mapping could be applied to motion or path planning problems when they are solved with reinforcement learning. It can be clearly seen that the feasibility policy learned to generate splines representative of all feasible options with only a few infeasible splines. Therefore, an objective policy should be greatly aided if it

only needs to choose among the splines that the feasibility policy can generate. In our future work, we plan to investigate action mapping using splines as action space in a reinforcement learning-based path planning problem.

VII. ROBOTIC GRASPING SETUP

Besides the illustrative examples, the proposed method was tested in a simplified robotic grasping simulation, where we compare different f-divergences with other approaches and investigate how the proposed approach reacts to distortions in the observation.

A. GRASPING SIMULATION

Our grasping simulator generates four shapes (H, 8, Spoon, T) for training and a Box shape for testing. The shape position, orientation, color, and geometry parameters are randomly sampled, producing various observations. The observation space is a 128×128 pixel RGB image. We assume a vertical configuration of a parallel gripper with three degrees of freedom x , y , and α and assume that the object is an extrusion of the 2D shape in the observation. The action space is constrained to the center 78×78 pixel region to avoid undefined behavior at the border of the RGB image. The angle of the grasp is in $[0, \pi)$ as the gripper is symmetrical; thus, a complete revolution is unnecessary.

The success of a grasp is only determined by the relative position and alignment of the gripper to the outline of the object, as seen from a camera positioned above the experiment. Given the alignment of the gripper, i.e., x , y , and α and a simulated picture of the object from a fixed camera, we developed an algorithm that provides a success/failure outcome in a deterministic and reproducible manner. Given the maximum aperture of the parallel gripper l and the width of the gripper claws w , the simulation analyzes the cropped image content of dimensions $l \times w$ between the gripper claws before the claws close on the object. The simulation checks if the object is sufficiently present, equidistant from the claws, and aligned within parameterized margins. Figure 3 shows

TABLE 3: List of parameters for all experiments.

	Sec. V	Sec. VI	Sec. VII	Description
N	128	256	128	Support size
M	256	256	256	Resampling size
σ_{xy}	0.01	0.1	0.025	KDE bw. x, y
σ_{sc}	-	-	0.4 (RKL: 0.1)	KDE bw. $\sin \alpha, \cos \alpha$
c	2.0	2.0	3.0	Sampling bw. $\sigma' = c\sigma$
U	-	-	64	Max Uncertainty Proposals
$ \mathcal{M} $	-	-	320,000	Replay memory size
K	16	16	16	Actor batch size
L	-	-	32	Critic batch size
lr_a	$5 * 10^{-5}$	$5 * 10^{-5}$	$5 * 10^{-5}$	Learning rate actor
lr_c	-	-	$5 * 10^{-5}$	Learning rate critic
c	$[0.0, 1.0]^2$	-	-	Center point range
r	$[0.1, 0.3]$	-	-	Radius range circles
r_o	$[0.2, 0.3]$	-	-	Outer radius range annuli
r_i	$[0.3, 0.7] * r_o$	-	-	Inner radius range annuli
l_{\min}	-	0.5	-	Minimum spline length
l_{\max}	-	1.0	-	Maximum spline length
c_{\max}	-	8.0	-	Maximum curvature
T_t	8 h	4 h	48 h	Training time
I_1	2.2 ms	1.5 ms	4.0 ms	Inference time 1 action
I_{256}	2.2 ms	1.5 ms	8.3 ms	Inference time 256 actions
I_{4096}	9.0 ms	4.0 ms	92.0 ms	Inference time 4096 actions

The training and inference times were measured on an NVIDIA A100 GPU. Inference times for multiple actions measure generating multiple actions for one problem. RKL is sensitive to large KDE bandwidths and benefits from a smaller bandwidth for $\sin \alpha, \cos \alpha$.

successful grasping poses and the respective gripper content for the objects that are trained on.

In the primary experiment, we test the algorithm under aligned observation and action spaces. In a second study, we investigate if distortions of the observation affect the performance. The distortions investigated are a rotation, projection, and rotation + projection as shown in Figure 4. These distortions correspond to different camera perspectives. The applied distortion is only on the observation and does not change the mechanics of the experiment.

B. NEURAL NETWORK DESIGN

The neural network that was used for the actor and critic in the robotic experiment is illustrated in Figure 5. The neural network design was guided by simplicity and inspired by Generative Adversarial Networks (GANs). Features that rely on domain-specific knowledge are avoided to evaluate better the learning method presented in the paper. The actor and critic share the residual feature extraction network ([37]). The hyperparameters for training and training and inference times are summarized in Table 3.

As a peculiarity of the network and the loss, the actor’s inferred action has four components, $[x, y, r \sin \alpha, r \cos \alpha]$, with $r \in [0, \sqrt{2}]$. The angle can be extracted trivially with the arctan of the ratio of the third and fourth action components. As the scale factor r does not change the angle, the critic receives the normalized action $[x, y, \sin \alpha, \cos \alpha]$ as input. To avoid the actor from reaching the singularity at $r = 0$ and the distribution q being spread along the radius, $g(s, a)$ and $\xi(s, a)$ are scaled with an unnormalized Gaussian on the radius, centered at 0.5 with the standard deviation of σ_{sc} .

C. COMPARISON

In the primary experiment, we are comparing different f -divergences with each other and with two other approaches. The first is a maximum entropy (ME) RL algorithm similar to Soft Actor-Critic (SAC) in [38], which trains the actor to minimize

$$\min_{\theta} \mathbb{E}_{s \sim \mathcal{M}, z \sim \mathcal{Z}} [\log q_{\theta}(\pi_{\theta}(s, z)|s) - \xi_{\phi}(s, \pi_{\theta}(s, z))], \quad (17)$$

with \mathcal{M} being the replay memory. The critic is trained as described in Section IV-D. Instead of using the reparameterization trick with a known distribution to estimate the entropy, we use the KDE. The other approach is an implementation of a conditional GAN ([39]) with a growing dataset. The min-max optimization problem is given through

$$\min_{\theta} \max_{\phi} \mathbb{E}_{s, a \sim \mathcal{M}_p} [\log(\xi_{\phi}(s, a)) - \log(1 - \xi_{\phi}(s, \pi_{\theta}(s, z)))] , \quad (18)$$

with a positive replay memory \mathcal{M}_p only containing feasible actions. An asterisk is added (e.g., JS*) when using action optimization, rejecting 10% of the proposed actions with the lowest critic value.

In the secondary evaluation, we compare with a common approach in the literature ([32]) that uses spatial equivariance. The domain-specific approach utilizes fully convolutional networks to output a probability of success for each action of a *discretized* action space. As in [32], the observation is fed into the neural network multiple times with different rotations. The neural network then only needs to output a one-channel image containing the probability of success of each discretized x, y action for the given rotation of the image. This approach thus uses translation equivariance by using a convolutional neural network (CNN) and rotation equivariance. In the experiments, we denote it as the heat-map approach (H).

The approach is implemented using fully convolutional networks with an hourglass structure, adopting the beginning of the Resnet in Figure 5 and adding the same structure in reverse order with nearest-neighbor upsampling. The approach predicts grasping success for 78x78 pixels with 16 rotation angles, trained on a cross-entropy loss on the grasping outcome sampled from the replay buffer. The replay buffer is also filled with imitation learning examples, and maximum uncertainty sampling is applied. For evaluation, the success estimate of each discretized action is used as its probability to be sampled. To increase accuracy, an inverted temperature factor increases the difference between higher and lower score actions. Specifically, the actions are sampled according to

$$q(a|s) = \frac{\exp(\beta \log \xi(s, a))}{\sum_{\forall a \in \mathcal{A}_d} \exp(\beta \log \xi(s, a))}, \quad (19)$$

with ξ being the fully convolutional network with s as input and as output shape the discretized action space \mathcal{A}_d . The inverted temperature was set to $\beta = 100$ for H and $\beta = 1000$ for H^* .

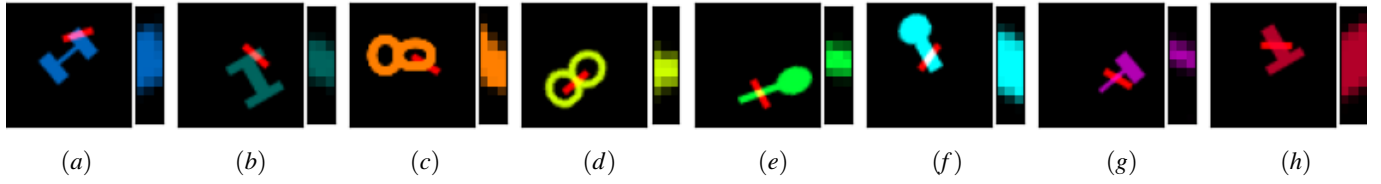


FIGURE 3: Feasible gripper positions (red) for different variations of the shapes (*H-shape* (a+b), *8-shape* (c+d), *Spoon* (e+f), and *T-shape* (g+h)) used in training, with a detailed view of the area between the gripper to the right of each figure.

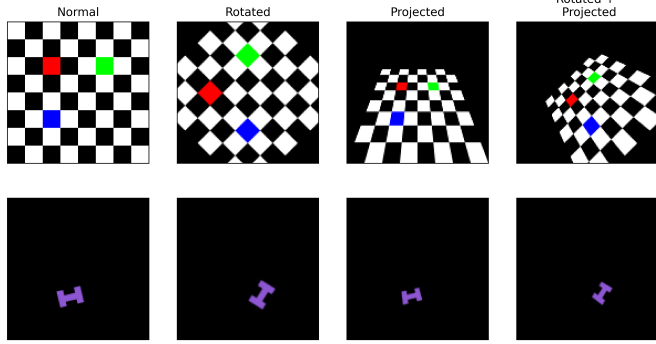


FIGURE 4: Different distortions are applied, showing a colored chess board for illustration and an example shape under all distortions.

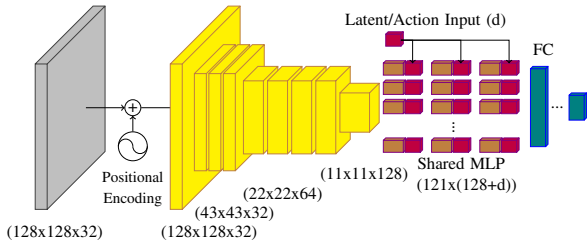


FIGURE 5: Before processing, the image is embedded (in gray) and augmented with positional encoding, resulting in 32 total channels. After positional encoding, a convolutional layer with stride 3, followed by 7 residual blocks (in yellow) with a bottleneck, preprocesses the state. The output is processed by 3 layers of "pixel-wise" shared MLPs (in brown), with the features being concatenated with a latent input (in purple) of length d . The latent input is a random sample from \mathcal{Z} for the actor and the action to be evaluated for the critic. Four (for the actor) or three (for the critic) fully connected layers (in blue) output the action and the feasibility estimate, respectively.

VIII. ROBOTIC GRASPING RESULTS

A. TOP-DOWN OBSERVATION

For each configuration, three agents were trained for 1 million interaction steps with the environment, taking approximately 48 hours per agent on a single NVIDIA 40GB A100 GPU. At the start of the training, 80k examples, including positives and negatives, for randomly generated shapes were added to the replay memory to bootstrap the critic and discriminator learning.

The training architecture is implemented in TensorFlow[40] with the parameters in Table 3.

Figure 6 shows the problem, the ground truth feasible picking positions, the critic estimate, and a heat-map of the actor's proposed actions. All figures are projections taking the maximum over the dimension that is not shown. In the problem visualization in Figure 6a, five feasible picks are shown in different colors, which correspond to the markers in Figure 6b. These markers highlight the complex multimodality of the problem. While it appears that, e.g., red and purple are in the same mode in the x - y projection, it is visible in the x - α projection that they are not directly connected. Figure 6c shows that the critic has an approximate understanding of the feasible regions of the action space, showing five modes clearly in the x - y projection. The actor distribution in Figure 6d also shows all five modes, while the output is significantly sharper in the centers of the modes. This is due to the use of the KDEs and the choice of bandwidth σ .

In the qualitative comparison in Figure 7, the actor distributions of the different algorithms are shown for three different shapes. While the *H* and *8* shapes were trained on, the *Box* shape has not been seen during training. The different subfigures show the action heat-maps of all implemented algorithms, showing only the x - y projections. The *H*-row shows that Jensen-Shannon (JS) and Forward Kullback-Leibler (FKL) learned all five modes, with JS having the fewest samples in the connecting area. Against the expectation from the illustrative examples, Reverse Kullback-Leibler (RKL) also learned all modes. The most probable reason is that the actor learns to match the critic's distribution, changing simultaneously from a rough estimate of one feasibility region to the refined shape of individual modes. If the actor learns the entire distribution of the critic early on, when the critic learns to distinguish different modes, the actor's distribution has support in all modes and is thus trapped in each mode. The GAN implementation shows four very unbalanced modes. Additionally, the modes are single points, which correspond to the automatically generated imitation examples, showing that the GAN approach can only imitate but cannot find other feasible actions. The ME implementation collapses in a single mode. The *8*-row and the *Box*-row show a similar pattern with the most pronounced spread of the action distributions in JS, FKL, and RKL and mostly collapsed action regions in the other approaches.

Each algorithm's accuracy and shares of modes on all shapes were evaluated to quantify the capability of generating actions

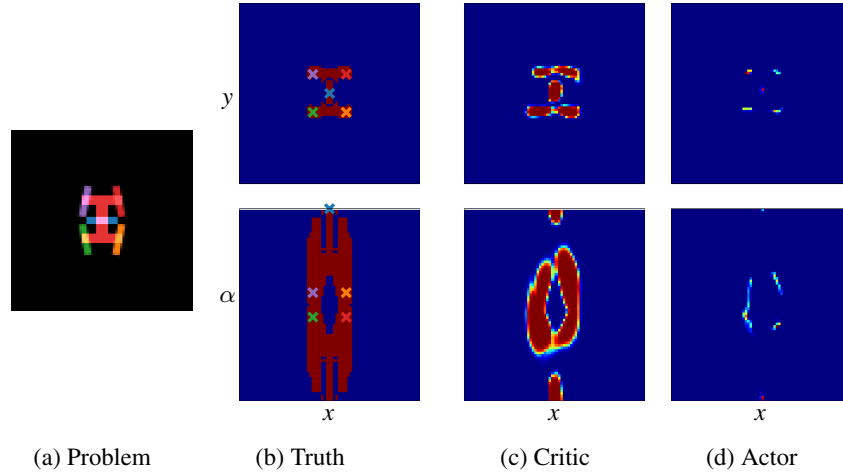


FIGURE 6: Critic classification and actor distribution trained with JS compared with the ground truth. Five example grasps are shown in the problem and their associated locations in the ground truth. The figures show projections onto the x - y plane (top row) and the x - α plane (bottom row).

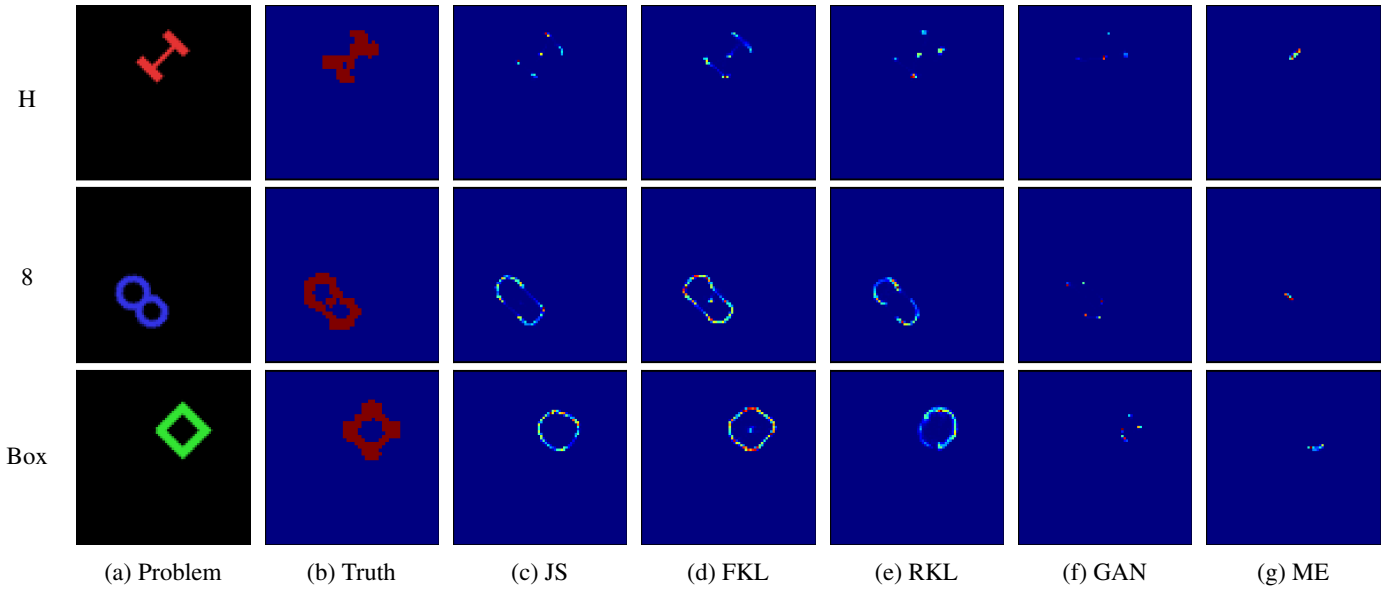


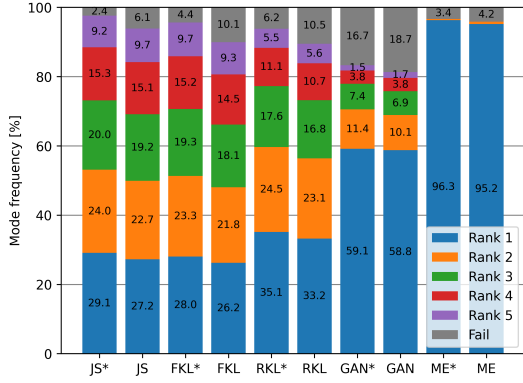
FIGURE 7: Qualitative comparison of the implemented algorithms, showing action heat-maps on three different states, with the last state never been observed during training.

in all disconnected sets of feasible actions. 1024 random states were generated for each shape that differed in pose, color, and geometry. For each state, 1024 actions were sampled from the different actors. The actions were then evaluated, and the mode of each action was recorded. The modes were then ranked and averaged over all the states of that shape by frequency. By averaging the ranks instead of the modes, the last rank shows the average ratio of the least frequent mode for each state.

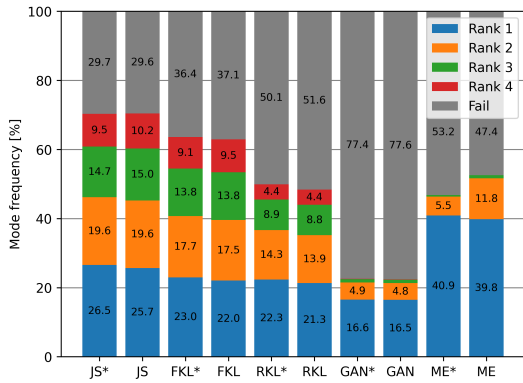
Figure 8 shows the shares of each rank for the H and Box shapes for all the algorithms. This figure presents the multimodal capabilities of the proposed approaches. For the H shape, JS and FKL have the most balanced distribution over the grasping modes. The GAN approach sometimes generates actions in all the modes but primarily focuses the actions in a primary mode. The ME approach almost exclusively generates

actions in one mode. The comparison on the Box shape shows that the generalization capability of the JS and FKL algorithms outperform the other approaches, which could indicate that explicitly learning all feasible actions improves generalization. The generalization capability of the GAN implementation is significantly lower than the others, as seen on the Box shape, indicating that that approach overfitted on the imitation examples.

To quantify the overall performance, Table 4 shows the precision (feasible actions generated over total actions generated) for each shape and the last ranked mode for the H , T , and Box shapes. The table shows that ME has solid performance on all shapes trained on but has lower generalization performance and fails to find the different modes. The GAN algorithm shows some actions in the last ranked modes, but it is signifi-



(a) H Shape



(b) Box Shape

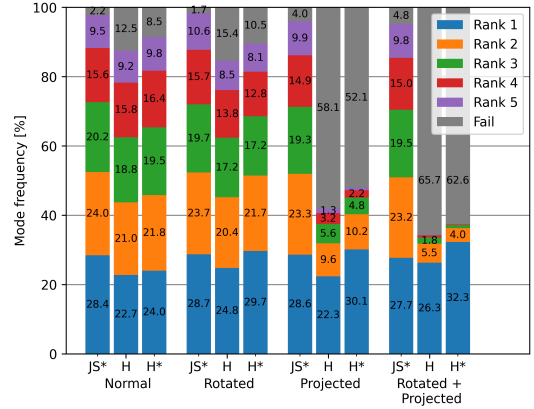
FIGURE 8: Gripping rank comparison, with the ratio of picks for each ranked mode or failure in %.

cantly weaker overall. The best approach is JS with the highest precision and similar shares in the last ranked mode as FKL. As discussed before, action optimization improves precision but reduces recall, slightly decreasing the least ranked mode for most approaches. The maximum deviations in the superscript show that all approaches learn reliably, with the GAN having the highest performance deviations among runs.

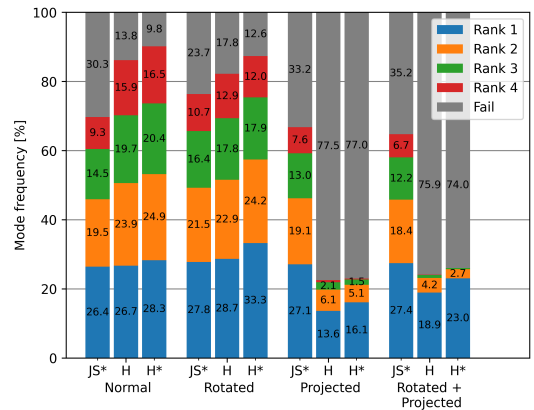
B. OBSERVATION VARIATION EXPERIMENTS

For each observation distortion, we trained one agent using the JS loss and one agent using the heat-map approach, each for 10^6 training steps. The results are shown in Figure 9 and Table 5, which compare the performance of the proposed Jensen-Shannon (JS) approach with the heat-map (H) approach. As expected, the domain-specific heat-map approach performs well on the original problem. In that scenario, no scene understanding is required, and only local features need to be considered to estimate grasping success. Therefore, the approach is expected to generalize well to unseen shapes, as seen for the Box-Shape, since the grasping success depends only on gripper alignment. It only needs to learn to imitate the grasping success heuristic shown in Figure 3.

Rotating the observation does not seem to impact its performance. However, under projection and projection + rotation, the heat-map approach fails to learn to grasp reliably. Our



(a) H Shape



(b) Box Shape

FIGURE 9: Gripping rank comparison, with the ratio of picks for each ranked mode or failure in %.

proposed approach learns well under all distortions. In general, the performance of our proposed approach does not depend on the distortion as it does not explicitly use the spatial structure. Its design does not depend on the specifics of the experiment at all. It can, therefore, learn independently of the distortion applied as long as the object is still fully observable.

IX. DISCUSSION

This paper introduced the concept of action mapping, in which an optimization process can be learned sequentially by first learning feasibility and then learning the objective. In this paper, we focused on the former part by learning to generate all feasible actions. We showed that by formulating a distribution matching problem and deriving a gradient estimator for general f-divergences, we train a feasibility policy that can function as a map between a latent space and the feasible action space. An illustrative example, a robotic path planning example, and experiments for robotic grasping show that our approach allows the feasibility policy to generate actions in all disconnected sets of feasible actions, a challenging task for state-of-the-art approaches. Enabling FKL and JS through our gradient estimator was instrumental.

Our experiments, detailed in Table 3, reveal that training

TABLE 4: Grasping score and mode comparison.

	JS*	JS	FKL*	FKL	RKL*	RKL	GAN*	GAN	ME*	ME	
Score	H	97.6 ^{0.0}	93.9 ^{0.1}	95.6 ^{0.5}	89.9 ^{0.6}	93.8 ^{1.6}	89.5 ^{1.3}	83.3 ^{5.9}	81.3 ^{6.1}	96.6 ^{0.3}	95.8 ^{0.5}
	T	98.2 ^{0.6}	96.2 ^{0.6}	97.1 ^{0.4}	93.5 ^{0.1}	96.1 ^{1.4}	93.2 ^{1.3}	84.8 ^{5.7}	82.8 ^{5.3}	96.7 ^{0.1}	95.8 ^{0.7}
	8	93.0 ^{1.2}	88.4 ^{1.4}	89.5 ^{1.0}	84.5 ^{0.7}	87.3 ^{4.3}	83.7 ^{4.6}	58.9 ^{7.8}	57.3 ^{7.9}	86.6 ^{1.1}	87.2 ^{2.6}
	Spoon	98.8 ^{0.5}	98.5 ^{0.5}	97.4 ^{0.6}	94.2 ^{1.4}	98.2 ^{0.5}	96.9 ^{0.9}	86.5 ^{6.4}	86.2 ^{6.9}	96.7 ^{0.5}	96.6 ^{0.7}
	Box	70.3 ^{4.3}	70.4 ^{4.0}	63.6 ^{2.4}	62.9 ^{2.1}	49.9 ^{13.5}	48.4 ^{12.7}	22.6 ^{3.3}	22.4 ^{4.0}	46.8 ^{1.0}	52.6 ^{16.0}
Avg	91.6 ^{0.8}	89.5 ^{0.7}	88.6 ^{0.3}	85.0 ^{0.1}	85.1 ^{3.9}	82.3 ^{3.7}	67.2 ^{5.2}	66.0 ^{5.4}	84.7 ^{0.1}	85.6 ^{3.6}	
Mode	H	9.2 ^{0.2}	9.7 ^{0.3}	9.7 ^{0.8}	9.3 ^{0.5}	5.5 ^{0.3}	5.6 ^{0.3}	1.5 ^{2.2}	1.7 ^{2.3}	0.0 ^{0.0}	0.0 ^{0.0}
	T	13.8 ^{0.8}	14.4 ^{1.1}	17.1 ^{0.6}	17.3 ^{0.2}	9.1 ^{3.0}	9.4 ^{3.0}	2.0 ^{1.3}	2.0 ^{1.3}	0.0 ^{0.0}	0.0 ^{0.0}
	Box	9.5 ^{1.2}	10.2 ^{0.9}	9.1 ^{0.7}	9.5 ^{0.7}	4.4 ^{2.2}	4.4 ^{2.0}	0.1 ^{0.1}	0.1 ^{0.1}	0.0 ^{0.0}	0.0 ^{0.0}

Comparison on all shapes with the mean of the grasping success ratio in % on top and the least ranked mode in % on the bottom, with the maximum deviations over the three runs in superscript.

TABLE 5: Grasping score and mode comparison under perspective distortions.

	Normal			Rotated			Projected			Rotated + Projected			
	JS*	H	H*	JS*	H	H*	JS*	H	H*	JS*	H	H*	
Score	H	97.8	87.5	91.5	98.3	84.6	89.5	96.0	41.9	47.9	95.2	34.3	37.4
	T	98.9	88.4	92.3	98.9	87.2	91.8	97.4	41.9	46.0	96.2	38.6	40.7
	8	91.6	84.8	89.4	94.9	80.9	86.4	90.3	24.5	28.0	86.6	17.2	19.0
	Spoon	99.4	89.0	93.0	98.9	88.0	92.2	98.3	43.5	46.1	97.1	38.0	40.9
	Box	69.7	86.2	90.2	76.3	82.2	87.4	66.8	22.5	23.0	64.8	24.1	26.0
Avg	91.5	87.2	91.3	93.5	84.6	89.4	89.8	34.9	38.2	88.0	30.5	32.8	
Mode	H	9.5	9.2	9.8	10.6	8.5	8.1	9.9	1.3	0.7	9.8	0.2	0.0
	T	13.8	17.9	18.5	12.8	16.5	15.5	8.2	1.6	0.9	8.8	0.4	0.2
	Box	9.3	15.9	16.5	10.7	12.9	12.0	7.6	0.7	0.3	6.7	0.1	0.0

time varies significantly across different setups, with no clear correlation to increases in dimensionality. Surprisingly, the 2D system described in Section V required more training time than the 4D system in Section VI. While our results do not show increased complexity with higher dimensions, we anticipate that scalability to higher-dimensional action spaces may still pose challenges. Nevertheless, adopting alternative non-parametric density estimators from existing literature could help mitigate these scalability concerns.

Given the proposed method for training the feasibility policy from a feasibility model, the following steps will focus on action mapping. We will test it in reinforcement learning scenarios for which a feasibility model is known. A potential problem could be that the rough transition between disconnected sets of feasible actions makes deterministic objective policies more challenging. An added regularizing loss on smoothness could improve the transition, all be it by likely reducing accuracy. Further, the approach is very sensitive to the KDE bandwidth. We may need to adapt it throughout training, learn it, or derive a better estimate based on the Jacobian of the network.

REFERENCES

- [1] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," in *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 99–107, IEEE, 2009.
- [2] H. Bharadhwaj, A. Kumar, N. Rhinehart, S. Levine, F. Shkurti, and A. Garg, "Conservative safety critics for exploration," in *International Conference on Learning Representations*, 2021.
- [3] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 3387–3395, 2019.
- [4] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.
- [5] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," in *The International FLAIRS Conference Proceedings*, vol. 35, 2022.
- [6] H. Krasowski, X. Wang, and M. Althoff, "Safe reinforcement learning for autonomous lane changing using set-based prediction," in *IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020.
- [7] M. Theile, H. Bayerlein, M. Caccamo, and A. L. Sangiovanni-Vincentelli, "Learning to recharge: Uav coverage path planning through deep reinforcement learning," 2023.
- [8] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," *Advances in Neural Information Processing Systems*, vol. 31, 2018.
- [9] J. Garcia and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.
- [10] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, "A general safety framework for learning-based control in uncertain robotic systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 7, pp. 2737–2752, 2019.
- [11] Z. Li, U. Kalabić, and T. Chu, "Safe reinforcement learning: Learning with supervision using a constraint-admissible set," in *2018 Annual American Control Conference (ACC)*, pp. 6390–6395, 2018.
- [12] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," *arXiv preprint arXiv:1801.08757*, 2018.
- [13] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European control conference (ECC)*, pp. 3420–3431, IEEE, 2019.
- [14] L. Sha *et al.*, "Using simplicity to control complexity,"
- [15] B. Zhong, A. Lavaei, H. Cao, M. Zamani, and M. Caccamo, "Safe-visor architecture for sandboxing (ai-based) unverified controllers in stochastic cyber-physical systems," *Nonlinear Analysis: Hybrid Systems*, vol. 43, p. 101110, 2021.

- [16] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, pp. 97–109, 04 1970.
- [17] A. E. Gelfand and A. F. Smith, "Sampling-based approaches to calculating marginal densities," *Journal of the American statistical association*, vol. 85, no. 410, pp. 398–409, 1990.
- [18] J. K. Kruschke, "Chapter 5 - bayes' rule," in *Doing Bayesian Data Analysis (Second Edition)* (J. K. Kruschke, ed.), pp. 99–120, Boston: Academic Press, second edition ed., 2015.
- [19] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul, "An introduction to variational methods for graphical models," *Machine Learning*, vol. 37, pp. 183–233, 01 1999.
- [20] M. Wainwright and M. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends in Machine Learning*, vol. 1, pp. 1–305, 01 2008.
- [21] S. Nowozin, B. Cseke, and R. Tomioka, "f-gan: Training generative neural samplers using variational divergence minimization," in *Advances in Neural Information Processing Systems* (D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds.), vol. 29, Curran Associates, Inc., 2016.
- [22] T. Hu, Z. Chen, H. Sun, J. Bai, M. Ye, and G. Cheng, "Stein neural sampler," *ArXiv*, vol. abs/1810.03545, 2018.
- [23] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *ICML*, 2015.
- [24] E. G. Tabak and C. V. Turner, "A family of nonparametric density estimation algorithms," *Communications on Pure and Applied Mathematics*, vol. 66, 2013.
- [25] E. G. Tabak and E. Vanden-Eijnden, "Density estimation by dual ascent of the log-likelihood," *Communications in Mathematical Sciences*, vol. 8, pp. 217–233, 2010.
- [26] Z. Kong and K. Chaudhuri, "The expressive power of a class of normalizing flow models," in *International conference on artificial intelligence and statistics*, pp. 3599–3609, PMLR, 2020.
- [27] F. Koehler, V. Mehta, and A. Risteski, "Representational aspects of depth and conditioning in normalizing flows," in *Proceedings of the 38th International Conference on Machine Learning* (M. Meila and T. Zhang, eds.), vol. 139 of *Proceedings of Machine Learning Research*, pp. 5628–5636, PMLR, 18–24 Jul 2021.
- [28] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on Robot Learning*, pp. 651–673, PMLR, 2018.
- [29] K. Kleeberger, R. Bormann, W. Kraus, and M. F. Huber, "A survey on learning-based robotic grasping," *Current Robotics Reports*, vol. 1, no. 4, pp. 239–249, 2020.
- [30] S. Kumra, S. Joshi, and F. Sahin, "Antipodal robotic grasping using generative residual convolutional neural network," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9626–9633, IEEE, 2020.
- [31] D. Morrison, P. Corke, and J. Leitner, "Learning robust, real-time, reactive robotic grasping," *The International journal of robotics research*, vol. 39, no. 2-3, pp. 183–201, 2020.
- [32] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1307–1319, 2020.
- [33] F. Liese and I. Vajda, "On divergences and informations in statistics and information theory," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4394–4412, 2006.
- [34] P. L'Ecuyer, "On the interchange of derivative and expectation for likelihood ratio derivative estimators," *Management Science*, vol. 41, no. 4, pp. 738–748, 1995.
- [35] J. Kleijn and R. Rubinstein, "Optimization and Sensitivity Analysis of Computer Simulation Models by the Score Function Method," Other publications TiSEM 958c9b9a-544f-48f3-a3d1-c, Tilburg University, School of Economics and Management, 1996.
- [36] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.
- [37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.
- [38] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.
- [39] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *ArXiv*, vol. abs/1411.1784, 2014.
- [40] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.



MIRCO THEILE received the M.Sc. degree in electrical engineering and information technology from Technical University of Munich, Germany, in 2018, where he is currently pursuing a Ph.D. degree. Currently, he is also a visiting researcher at the University of California in Berkeley, USA. His research interests extend to reinforcement learning in applications of cyber-physical systems, including UAVs, robotics, and real-time systems.



DANIELE BERNARDINI received his M.Sc. degree in Theoretical Physics at the University of Florence in 1997. After graduation, he spent 2 more years as a researcher at the Ludwig Maximilians University, Munich before transitioning to the industry, where he gained more than 20 years of experience in software development and data science. In 2021 he joined the Technical University of Munich as research group leader where he focuses on advancing perception for robotic manipulation. Since 2021 he is a co-founder and CEO of Cognivix, a startup specializing in automation solutions for industries requiring high variability and low volume production.



RAPHAEL TRUMPP graduated with a M.Sc. degree in mechanical engineering from the Technical University of Munich in 2021, where he is currently pursuing a Ph.D. in informatics. His research focuses on machine learning, especially combining deep reinforcement learning with classical control methods. He is interested in applying these to interactive multi-agent scenarios like autonomous racing and robotics.



CRISTINA PIAZZA received a B.Sc. in Biomedical Engineering, a M.S. in Automation and Robotics Engineering and a PhD degree in Robotics (summa cum laude, 2019) from the University of Pisa (Italy). She subsequently moved to Chicago (USA) where she worked as a postdoctoral researcher at Northwestern University. Since 2020, Prof. Piazza is tenure track assistant professor at Technical University of Munich



MARCO CACCAMO earned his Ph.D. in computer engineering from Scuola Superiore Sant'Anna (Italy) in 2002. Shortly after graduation, he joined University of Illinois at Urbana-Champaign as assistant professor in Computer Science and was promoted to full professor in 2014. Since 2018, Prof. Caccamo has been appointed to the chair of Cyber-Physical Systems in Production Engineering at Technical University of Munich, Germany. In 2003, he was awarded an NSF CAREER Award.

He is a recipient of the Alexander von Humboldt Professorship and he is IEEE Fellow.



ALBERTO L. SANGIOVANNI-VINCENTELLI is the Edgar L. and Harold H. Buttner Chair of Electrical Engineering and Computer Sciences at the University of California, Berkeley. He was a co-founder of Cadence and Synopsys, the two leading companies in the area of Electronic Design Automation. He is currently a Board member of 8 companies, including Cadence, and Chairman of the Board of Quantum Motion, Innatera, Phoelex, e4Life and Phononic Vibes. He is the recipient

of several academic honors, and research awards including the IEEE/RSE Wolfson James Clerk Maxwell Medal “for groundbreaking contributions that have had an exceptional impact on the development of electronics and electrical engineering or related fields”, the BBVA Frontiers of Knowledge Award in the Information and Communication Technologies category, the Kaufmann Award for seminal contributions to EDA, the IEEE Darlington Award, the EDAA lifetime Achievement Award, and four Honorary Doctorates from University of Aalborg, KTH, AGH and University of Rome, Tor Vergata. He is an author of over 1000 papers, 17 books and 3 patents in the area of design tools and methodologies, large scale systems, embedded systems, hybrid systems, and AI.

Chapter 6

Reinforcement Learning for Graph-based Task Scheduling

6.1 Latency-Aware Generation of Single-Rate DAGs from Multi-Rate Task Sets

Reference

M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, “Latency-aware generation of single-rate dags from multi-rate task sets,” in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 226–238, IEEE, 2020
DOI: <https://doi.org/10.1109/RTAS48715.2020.000-4>

Abstract

Modern automotive and avionics embedded systems integrate several functionalities that are subject to complex timing requirements. A typical application in these fields is composed of sensing, computation, and actuation. The ever-increasing complexity of heterogeneous sensors implies the adoption of multi-rate task models scheduled onto parallel platforms. Aspects like freshness of data or first reaction to an event are crucial for the performance of the system. The Directed Acyclic Graph (DAG) is a suitable model to express the complexity and the parallelism of these tasks. However, deriving age and reaction timing bounds is not trivial when DAG tasks have multiple rates. In this paper, a method is proposed to convert a multi-rate DAG task-set with timing constraints into a single-rate DAG that optimizes schedulability, age and reaction latency, by inserting suitable synchronization constructs. An experimental evaluation is presented for an autonomous driving benchmark, validating the proposed approach against state-of-the-art solutions.

Contributions to this paper

- Shared conceptualization of expressing timing constraints through dummy and synchronization nodes
- Shared implementation of the DAG generation methodology
- Share of paper writing

Copyright

© 2020 IEEE. Reprinted, with permission, from Micaela Verucchi, Mirco Theile, Marco Caccamo, and Marko Bertogna, “Latency-Aware Generation of Single-Rate DAGs from Multi-Rate Task Sets”, 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), April 2020.

See Appendix A.9 for the reuse statement. The following shows the accepted version.

Latency-Aware Generation of Single-Rate DAGs from Multi-Rate Task Sets

Micaela Verucchi^{*†}, Mirco Theile[†], Marco Caccamo[†] and Marko Bertogna^{*}

^{*}University of Modena and Reggio Emilia, Italy
 {micaela.verucchi, marko.bertogna}@unimore.it

[†]Technical University of Munich, Germany
 {mirco.theile, mcaccamo}@tum.de

Abstract—Modern automotive and avionics embedded systems integrate several functionalities that are subject to complex timing requirements. A typical application in these fields is composed of sensing, computation, and actuation. The ever increasing complexity of heterogeneous sensors implies the adoption of multi-rate task models scheduled onto parallel platforms. Aspects like freshness of data or first reaction to an event are crucial for the performance of the system. The Directed Acyclic Graph (DAG) is a suitable model to express the complexity and the parallelism of these tasks. However, deriving age and reaction timing bounds is not trivial when DAG tasks have multiple rates. In this paper, a method is proposed to convert a multi-rate DAG task-set with timing constraints into a single-rate DAG that optimizes schedulability, age and reaction latency, by inserting suitable synchronization constructs. An experimental evaluation is presented for an autonomous driving benchmark, validating the proposed approach against state-of-the-art solutions.

Index Terms—DAG, multi-rate, end-to-end latency, schedulability.

I. INTRODUCTION

Modern automotive and avionics real-time embedded systems are composed of applications including sensors, control algorithms and actuators to regulate the state of a system in its environment within given timing constraints. Task chains are commonly adopted to model a sequence of steps performed along the control path. Complex data dependencies may exist between task chains with different activation rates, making it very hard to find reliable upper bounds on the end-to-end latency of critical effect chains [1].

This problem is exacerbated by the adoption of even more complex task models based on Directed Acyclic Graphs (DAG) to capture the parallel activation of multiple jobs executing on heterogeneous multi-core platforms. A recent example in the automotive domain is given in the WATERS industrial challenge [2], focusing on the minimization of the end-to-end latency of critical effect chains of an autonomous driving system involving several sensors. The application is modeled in Figure 1, with three sensors providing input to multiple task chains. Nodes represent tasks with different activation periods, while edges represent the exchange of data between tasks, forming effect chains. Reaction to input stimuli and freshness of data are key factors to consider when deploying the application on a selected computing platform. *Data age* quantifies for how long an input data affects an output of a task chain, i.e., it is the maximum delay between

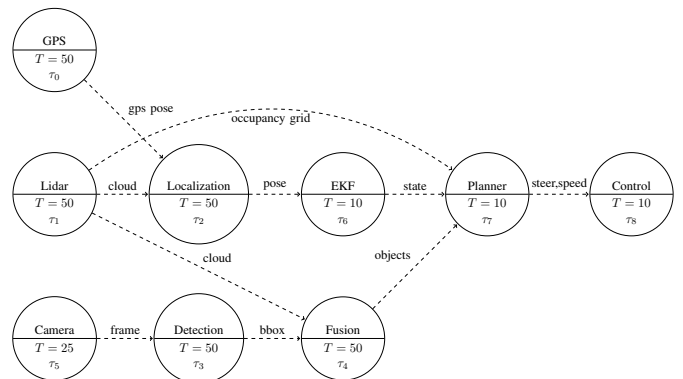


Fig. 1: An example of high-utilization automotive application with tasks at different periods.

a valid sensor input until the *last* output related to that input in the chain. *Data age* constraints are commonly found in control systems, where the age of the data can directly influence the quality of the control. In the considered application, key effect chains to optimize for *data age* are connected to the processing of camera frames and LiDAR point clouds: the older the input, the less precise is the localization of the ego vehicle and the detection of obstacles.

Another key metric to optimize is the *reaction latency*, a parameter that measures the reactivity of the system to a change in the input. It is defined as the maximum delay between a valid sensor input until the *first* output of the event chain that reflects such an input. It measures how much time it takes for a new event to propagate through a chain. In the considered autonomous driving example, key *reaction times* to optimize are the detection of an obstacle in the driving path, and the related actuation on the steering and breaking system to safely avoid it in due time.

The aim of this work is to consider such systems composed of DAG tasks having multiple rates with given constraints on age and reaction latencies. Starting from a high-level representation, a method is presented to create a single-rate DAG that fulfills the given restrictions, optimizing schedulability and end-to-end delays. To do so, a set of DAG candidates is generated and evaluated by a constrained cost function designed to pick the best DAG meeting the given requirements.

The paper is organized as follows: in the next section, an overview of the state-of-the-art is given before introducing the system model of the proposed approach. In Section IV, the mathematical basis of the following sections is derived. Section V gives a detailed explanation of the conversion from the multi-rate task set to the single-rate DAG. Section VI focuses on the requirements: end-to-end latency, schedulability and their evaluation. Finally, we conclude with an experimental part, in which the results of our framework are shown, comparing them with other existing methods.

II. RELATED WORK

A. End-to-end latency

A task chain is a sequence of communicating tasks in which every task receives data from its predecessor. In literature, two types of task chains can be found: periodic chains and event-driven chains [3]. In the former, each task is activated independently at a given rate, and it communicates with its successor by means of shared variables; in the latter, task executions are triggered by an event issued from a preceding task. The propagation delays of a task chain affect responsiveness, performance and stability of an application.

We hereafter focus on the periodic model, which is the most common in the automotive domain [1]. Di Natale et al. [4] proposed a method to evaluate the worst-case latency of mixed chains of real-time tasks and Controller Area Network (CAN) messages. Zeng et al. [5], [6] computed the probability distribution, via statistical analysis, of end-to-end latencies for CAN message chains.

Feiertag et al. [7] were the first to define *data age* and *reaction time* and to propose a framework to calculate end-to-end latencies in automotive systems, where each task operates according to the read-execute-write semantic, also known as the implicit communication model of AUTOSAR [8].

Becker et al. presented in [9] [10] a method to compute worst- and best-case *data age* for periodic tasks with implicit deadlines using implicit, explicit and Logical Execution Time (LET) communication models. The analysis is based on Read Interval (RI) and Data Interval (DI), which respectively are the interval in which a task can possibly read its input data in order to complete its execution before the deadline, and the interval for which the output data of a task can be available to the successor task in the chain. Multiple Data Propagation Trees (DPT) are constructed in order to compute the *data age*. A method is also described to constrain the maximum latency by inserting job-level dependencies. A tool, called MECHAniSer [11], is presented to compute latency values for a given task set. Regarding the LET model, Biondi et al. [12] and Martinez et al. [13] addressed the problem of computing end-to-end latency bounds on multi-cores, improving the results of Becker et al. in [10]. Our paper does not focus on the LET model, but it aims at deriving better latency bounds for the implicit model.

There exist other works that aim at selecting the best periods or deadlines to minimize data age in simpler task models. In [14], this is done on a single core platform, without

considering task chains. In [15], the authors propose a method to find the best period to bound data freshness of task chains, assuming the task set given in input be already schedulable. Adapting these solutions to our setting is not trivial, because we assume periods and deadline to be given.

B. Multi-rate DAG

In [16], Saito et al. present a framework developed for the Robot Operating System (ROS) to handle automotive applications with multi-rate tasks. The model assumes an event-driven data-flow system in which a node starts when the predecessor nodes are completed. In order to handle multi-rate tasks, a synchronization system is adopted consisting of two kinds of additional nodes: synch driver nodes and synch nodes. The synch driver node is used to adjust the publishing period of the sensors, buffering the data of the highest rate one, in order to have a node with a unique rate for all the sensors. Synch nodes are then inserted before the tasks to handle buffered data. In this way, a single-rate DAG is obtained and scheduled using a fixed-priority algorithm based on the HLBS scheduler [17].

Forget et al. [18] faced the same problem for autopilot applications, considering periodic tasks modeled as nodes in a DAG with two kinds of edges: simple and extended. Simple edges are precedence constraints between tasks having the same rate, while extended edges are data dependencies between tasks having different rates. To handle extended edges, a method is proposed to generate multiple conversions from extended edges through simple precedence constraints between jobs, selecting a permutation that guarantees Earliest Deadline First (EDF) schedulability.

Another conversion method from a multi-rate DAG to a single-rate one has been proposed by Saidi et al. in [19] for a similar DAG model. The output DAG has a period equal to the hyper-period of the input task set. The nodes are the job instances activated in a hyper-period for each task. Edges are precedence constraints between jobs, which are inserted based on the ratio between the periods of the communicating tasks. A multi-core heuristic is proposed to schedule the DAG, while minimizing a cost function related to task schedulability.

Converting the original task set to a DAG is a very convenient approach that allows seamlessly inserting explicit precedence constraints to control end-to-end latency. To our knowledge, most of the other methods in the literature perform similar conversions to impose such precedence constraints for limiting latency. While the work of Becker [9] may appear different, as it does not explicitly consider DAGs, it ends up implementing a similar approach by inserting precedence constraints between different jobs. In Section VII, we will highlight the differences between the presented methods and our approach.

III. SYSTEM MODEL

This work shows how to convert a *Multi-Rate Task set with Constraints* into a *Single-Rate Directed Acyclic Graph (DAG)*,

in order to analyze schedulability and end-to-end latency of task-chains.

A. Multi-Rate Task set with Constraints

The input to the proposed method is a task set Γ , modeling an application like the one in Figure 1, composed of N periodic tasks τ_x arriving at time $t = 0$. Each task τ_x is described by the tuple (WC_x, BC_x, T_x, D_x) , where:

- $WC_x \in \mathbb{R}$ is the Worst Case Execution Time (WCET) of the task;
- $BC_x \in \mathbb{R}$ is the Best Case Execution Time (BCET);
- $T_x \in \mathbb{N}$ is the period;
- $D_x \in \mathbb{R}$ represents the relative deadline.

The exchange of data between two tasks is modeled with as *data edge*, a directed (dashed) edge between the producer and the consumer of the data. Moreover, precedence constraints may be specified between two tasks (τ_x, τ_y) , stating that a job $\tau_{y,b}$ cannot start until all the jobs of τ_x released in τ_y 's period completed their execution. For this reason, precedence constraints can be inserted only between tasks having the same period, corresponding to job level precedence constraints.

To constrain the latency of data propagation in task-chains, upper bounds on *data age* and on *reaction time* can be given. The latency constraints evaluation is described in more detail in Section VI.

Our approach is based on a global non-preemptive list scheduling approach, as described in Section VI-B. Such a policy allows different instances of the same task to run on different cores, while preventing a job to be migrated during its execution, mitigating the preemption overhead.

B. Directed Acyclic Graph

The output of the proposed method is a *single-rate Directed Acyclic Graph (DAG)*. Such a model is based on the parallel DAG model proposed by Baruah in [20] to capture the parallelism of a task to be scheduled on a multi-core platform. In this model, tasks are represented as directed acyclic graphs, each with a unique source vertex and a unique sink vertex. Each vertex represents a sequential job, while edges represent precedence constraints between jobs.

In this work, we use a similar model with a semantic difference, i.e., a DAG represents a full application, with each vertex representing a task instance, which we call job. In detail, the DAG is specified by a 3-tuple (V, E, HP) where:

- V represents the set of nodes, namely the jobs of the tasks of Γ , and $n = |V|$;
- E is the set of edges describing job-level precedence constraints;
- HP is the period of the DAG, namely the hyper-period of the tasks involved: $HP = lcm_{\tau_x \in \Gamma} \{T_x\}$.

In this model, the communication between jobs utilizes buffers in shared memory, which can be accessed by all the cores. The time to write/read a shared buffer is included in the execution time of each task. We adopt the implicit communication model defined in AUTOSAR [8], solving mutual exclusion via double-buffering. Each task complies

with a read-execute-write semantic, i.e., it reads a private copy before the execution, and it writes a private copy at the end of the execution [1].

C. Notation

For the sake of clarity, a standardized set of indexing names is adopted throughout the entire paper, i.e., $\{i, j, k\}$ denote general nodes in a DAG (jobs, synchronization or dummy nodes), $\{x, y, z\}$ indicate tasks, and $\{a, b, c\}$ are used for jobs.

IV. BACKGROUND

This part describes the main algorithms used in the following sections. A DAG is represented as an adjacency matrix $\mathbf{T} \in \mathbb{B}^{n \times n}$, in which $T_{i,j} = 1$ iff there exists an edge $e(v_j, v_i)$ ¹. Given this Boolean formulation of the DAG, Boolean algebra can be applied. Therefore, the Boolean matrix product is defined as:

$$\mathbf{C} = \mathbf{A}\mathbf{B}, \quad \mathbf{A} \in \mathbb{B}^{n \times m}, \mathbf{B} \in \mathbb{B}^{m \times n}, \mathbf{C} \in \mathbb{B}^{n \times n} \quad (1)$$

for which the cells of \mathbf{C} evaluate to

$$c_{i,j} = \bigvee_{k=0}^{m-1} a_{i,k} \wedge b_{k,j} \quad (2)$$

Cell-wise Boolean operations are denoted as \wedge and \vee for *and* and *or*, respectively. Additionally, a maximum matrix multiplication is used in this work to combine Boolean matrices with real matrices. It is defined as

$$\mathbf{C} = \max\text{Product}(\mathbf{A}, \mathbf{B}), \quad (3)$$

$$\mathbf{A} \in \mathbb{B}^{n \times m}, \mathbf{B} \in \mathbb{R}^{m \times n}, \mathbf{C} \in \mathbb{R}^{n \times n}$$

where the cells of \mathbf{C} are calculated as

$$c_{i,j} = \max_{k \in \{0, \dots, m-1\}} \{a_{i,k} b_{k,j}\}. \quad (4)$$

A. Transitive Closure

The proposed scheduling method and the related end-to-end latency computation make use of the mathematical principles of graph theory [21]. One principle is the transitive closure [22] of a DAG, defined as

$$\mathbf{D} = \bigvee_{k=1}^n \mathbf{T}^k \quad (5)$$

where the exponentiation of a Boolean matrix is calculated through the Boolean matrix product defined in (1). The transitive closure of a DAG describes the set of descendants of each node, where $d_{i,j} = 1$ if there exists a path from v_j to v_i , i.e., v_i is a descendant of v_j . Consequently, v_j is an ascendant of v_i . The transpose of the descendants matrix, \mathbf{D}^T , therefore represents the ascendants matrix.

Computing the power of k of an adjacency matrix of a graph means calculating the nodes reachable through any k -step walk from every node v_i , which is a general result in graph theory

¹We chose the column-row approach over the commonly used row-column approach to perform state and value propagation, described later in this section, by left-multiplying the transition matrix to a column state vector.

(Lemma 2.5 in [21]). Instead of computing the descendants matrix via (5), we can adopt a simpler formulation. By introducing a self-loop to every node, the power of k of the adjacency matrix calculates not only the reachable nodes of any k -step walk, but it also includes the reachable nodes through all shorter walks. Therefore,

$$\mathbf{D} = (\mathbf{T} \vee \mathbf{I})^n \wedge \neg \mathbf{I} \quad (6)$$

where $\mathbf{I} \in \mathbb{B}^{n \times n}$ is the identity matrix, and $\neg \mathbf{I}$ is the Boolean complement of \mathbf{I} . Given that \mathbf{T} is an acyclic transition matrix, \mathbf{T}^k has no element on the main diagonal $\forall k \in \mathbb{N}_{>0}$. Therefore, the elements introduced on the main diagonal are set back to zero.

B. State and Value Propagation

To use the DAG matrix \mathbf{T} for the analysis of a DAG, two propagation methods are useful. The first is a Boolean state propagation and the second is a maximum value propagation. Let $\mathbf{x}_k \in \mathbb{B}^{n \times 1}$ denote a state describing which node of the DAG is visited at iteration k . Then, the state of the DAG in iteration $k+1$ can be calculated using the Boolean matrix multiplication as:

$$\mathbf{x}_{k+1} = \mathbf{T}\mathbf{x}_k \quad (7)$$

In this way \mathbf{x}_{k+1} will contain 1 for the nodes that are reached with one step-walk from the ones in state \mathbf{x}_k , 0 for the others.

Similarly, a value can be propagated through the DAG. Let $\mathbf{v}_k \in \mathbb{R}^{n \times 1}$ denote a value for each node of the DAG at iteration k . This value can be propagated through the paths of the DAG by using

$$\mathbf{v}_{k+1} = \text{maxProduct}(\mathbf{T}, \mathbf{v}_k), \quad (8)$$

where the vector \mathbf{v}_{k+1} describes the value \mathbf{v}_k in the next iteration.

In this work, we are interested in propagating execution times along the DAG. Given that in a DAG more paths can converge to the same node, we will propagate the maximum value among converging paths. In the case of propagating execution times through the DAG, we can define a value function \mathbf{v} as

$$\mathbf{v} = \text{maxProduct}(\mathbf{T}, \mathbf{v} + \mathbf{c}), \quad (9)$$

with \mathbf{c} being the execution time of each node (WC or BC). In this equation, the value of a node is equal to the maximum of its predecessors' values plus its execution time. The fixed-point \mathbf{v}^* solving Equation (9) can be found by iterating

$$\mathbf{v}_{k+1} = \text{maxProduct}(\mathbf{T}, \mathbf{v}_k + \mathbf{c}) \quad (10)$$

until it converges to \mathbf{v}^* when $\mathbf{v}_{k+1} = \mathbf{v}_k$. Convergence is guaranteed to happen after at most n iterations, because the graph is acyclic and, therefore, all its paths are composed of n or fewer nodes.

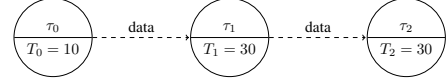


Fig. 2: The simple task set defined in Example 1.

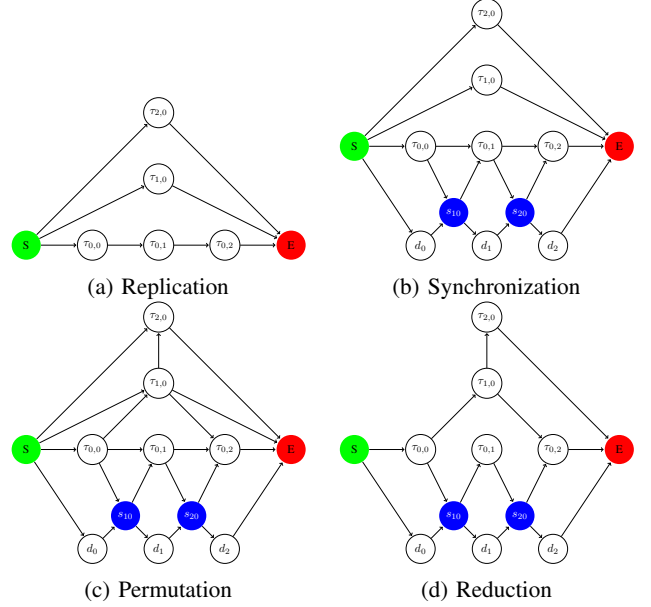


Fig. 3: The 4-Stage DAG generation depicted.

V. DAG GENERATION

In this section, we explain how to convert a task set of periodic tasks with constraints to a set of potential *single-rate* DAGs. The explanation and mathematical derivations are augmented with an example to illustrate the conversion.

Example 1. We consider an application modeled as a Multi-Rate task set $\Gamma = \{\tau_0 = (7, 5, 10, 10), \tau_1 = (13, 10, 30, 30), \tau_2 = (10, 8, 30, 30)\}$, with a constraint on the maximum data age of chain $\{\tau_0, \tau_1, \tau_2\}$ to be smaller than 50. The Multi-Rate task set is represented in Figure 2.

A set of DAGs is generated using a 4-Stage DAG Generation. The set is subsequently pruned to accelerate the analysis in the next sections.

A. 4-Stage DAG Generation

We aim at generating a set of DAGs that have the potential to meet all the constraints. The DAG generation can be split into four stages:

- 1) The respective jobs of the tasks are created.
- 2) The jobs are synchronized to meet their respective deadlines.
- 3) The job-level precedence edges are added to address the *data edges*.
- 4) The DAGs are simplified by removing redundant edges.

The four steps for the example are depicted in Figure 3. We hereafter detail each step.

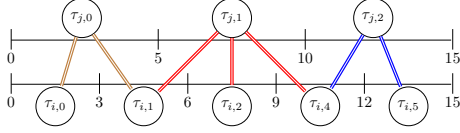


Fig. 4: Example of jobs of non-harmonic tasks limited to their super-period. Doubled arches indicate possible interaction between jobs.

1) *Replication*: Each task has to execute a number of jobs within one hyper-period. For a task τ_x , the number of jobs is $\frac{HP}{T_x}$. Since jobs are just instances of the same task, they should always run sequentially, therefore job-level precedence edges are added between successive jobs $\tau_{x,a}$ and $\tau_{x,a+1}$ where $a \in \{0, \dots, \frac{HP}{T_x} - 1\}$. Additionally, the start node of the DAG is connected to each first job of each task, and each last job is connected to the end node. The resulting DAG for the example is shown in Figure 3a. To synchronize the jobs in the following step, each job gets an offset and deadline. For $\tau_{x,a}$, the offset is aT_x and the deadline is $aT_x + D_x$.

2) *Synchronization*: To be sure that tasks instances maintain their original period and deadlines in the DAG, a synchronisation mechanism has to be applied. In this way, we can enforce a job to start after its offset and to finish before its deadline. To accomplish this, we add additional nodes for synchronization purposes, as in Figure 3b. Firstly, we add a synchronization node σ_t , with $WC = BC = 0$, for each unique value t in the list of offsets and deadlines of all jobs. Secondly, we add dummy nodes δ between each two consecutive synchronization nodes σ_t and $\sigma_{t'}$, with $WC = BC = t' - t$, i.e., the difference in the timestamps of the corresponding synchronization nodes. The source and sink of the DAG are synchronization nodes too, with a timestamp of 0 and HP , respectively.

To enforce the jobs to execute in a time-window within its offset and deadline, an edge to the job is added from the synchronization node of the corresponding offset, and another one from the job to the synchronization node corresponding to its deadline.

3) *Permutation*: The various instances of tasks with different periods may be scheduled in multiple ways. We would like to enforce a suitable execution order between such instances, in order to minimize the latency of a given set of task chains. Thus, we convert the original multi-rate task set into several single-rate DAGs, each representing a possible activation pattern of the considered tasks. To do so, we include additional precedence edges to the DAG obtained at the previous step.

Consider two tasks τ_x and τ_y with periods T_x and T_y , assuming $T_y \geq T_x$ without loss of generality. Let $SP_{x,y}$ be the super-period of tasks τ_x and τ_y , defined as the least common multiple of their periods, i.e., $SP_{x,y} = lcm(T_x, T_y)$. Note there are $\frac{HP}{SP_{x,y}} - 1$ super-periods in the hyper-period HP of the whole task set.

There exist multiple ways to insert precedence edges between jobs of τ_x and τ_y in each super-period of length $SP_{x,y}$.

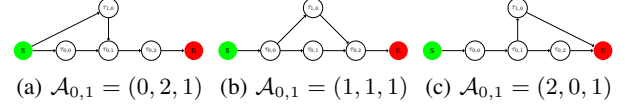


Fig. 5: Arrangement permutations with one parallel job of τ_x ; there are three permutations because $Q = 3$ and $\psi_0 = 1$, therefore $Q + 1 - \psi = 3$. The permutation (b) corresponds to the example in 3c.

Each possible edge assignment that complies with the multi-rate task specification is called “job arrangement”.

To find all the possible permutations, two cases must be considered: harmonic and non-harmonic periods. In the former case, there exists $q \in \mathbb{N}$ for which $q = \frac{T_y}{T_x}$ and $SP_{x,y} = T_y$. Therefore, finding all the permutations between one job of τ_y and q of τ_x allows finding all the job arrangements in their super-period. The non-harmonic case is slightly more complicated. For two non-harmonic tasks, $q \in \mathbb{N}$ can be computed as $q = \lceil \frac{T_y}{T_x} \rceil$, but $SP_{x,y} \neq T_y$. In this case, one job of τ_y can be arranged with q or $q + 1$ jobs of τ_x , because of the non-harmonicity. To better understand the problem, let us consider an example in which $T_x = 3$ and $T_y = 5$, as in Figure 4.

When periods are harmonic, a job of τ_x always interact with exactly one job of τ_y (and respectively, τ_y interacts with exactly q jobs of τ_x). However, for non-harmonic periods, a job of τ_x can interact with 1 or 2 (at most) jobs of τ_y , as shown in Figure 4. For this reason, in the non-harmonic scenario some jobs of τ_y will interact with q (in the example $\lceil \frac{5}{3} \rceil = 2$, as for $\tau_{y,0}$ and $\tau_{y,2}$) jobs of τ_x , while others with $q + 1$ (in this case 3, as for $\tau_{y,1}$). In general, a job $\tau_{y,a}$ can interact with all the jobs between $\tau_{x,b}$ and $\tau_{x,c}$, where b and c can be obtained as:

$$b \in \mathbb{N} \mid o(\tau_{x,b}) \leq o(\tau_{y,a}) \wedge o(\tau_{x,b}) + T_x > o(\tau_{y,a}) \quad (11)$$

$$c \in \mathbb{N} \mid o(\tau_{x,c}) < o(\tau_{y,a}) + T_y \wedge o(\tau_{x,c}) + T_x \geq o(\tau_{y,a}) + T_y \quad (12)$$

where $o(\tau_{x,b})$ stands for the offset of the job $\tau_{x,b}$.

Once the interacting job of τ_x and τ_y have been associated, this case can be traced back to the harmonic one.

Now, let us consider a job $\tau_{y,s}$ and all the possible arrangements with Q jobs of τ_x (which is either q or $q + 1$), denoted as $\mathcal{A}_{x,y}(s) = (pre_s, post_s, \psi_s)$ in the super-period $SP_{x,y}$. In this tuple, pre_s (resp. $post_s$) denotes the number of jobs of τ_x executing before (resp. after) each job of τ_y . ψ_s denotes the number of jobs of τ_x that can execute in parallel to the job of τ_y . This parameter is critical for the data update *variability*, that is defined as the difference between the maximum and minimum number of data updates. Since pre_s , $post_s$, and ψ_s comprise all the jobs of τ_x interacting with $\tau_{y,s}$, it follows that

$$pre_s + post_s + \psi_s = Q. \quad (13)$$

Three example arrangements for two tasks, τ_0 with $T_0 = 10$ and τ_1 with $T_1 = 30$, are shown in Figure 5. In all three arrangements, the job of τ_1 is parallel to one job of τ_0 ($\psi_0 =$

1). The number of permutations of arrangements with each $\tau_{y,s}$ can be calculated as:

$$perm(\mathcal{A}_{x,y}(s)) = \sum_{\psi=\{0\dots Q\}} (Q+1-\psi), \quad (14)$$

while the permutations can be found combining all the possible edges between the jobs of the two tasks.

For the harmonic case, this value is also the total number of permutations of a super-period:

$$perm_{SP_{x,y}} = perm(\mathcal{A}_{x,y}(s)). \quad (15)$$

On the other hand, for the non-harmonic case, the number of permutations for the super-period is obtained as:

$$perm_{SP_{x,y}} = \prod_{\forall \tau_{y,s} \in \{0\dots \frac{SP_{x,y}}{T_y}\}} perm(\mathcal{A}_{x,y}(s)). \quad (16)$$

Finally, considering all the super-periods contained in a hyper-period, the total number of permutations can be given by:

$$perm_{total} = \prod_{\forall x,y} perm_{SP_{x,y}}^{\frac{HP}{SP_{x,y}}}, \quad (17)$$

where $x \neq y$ and τ_x and τ_y are consecutive tasks in a given task chain. Each combination of arrangement permutations generates a new DAG that can be analyzed. Therefore, it is critical to keep the number of possible permutations as small as possible. A reduction of the exploration space is discussed in Section V-B.

Figure 3c shows one of the obtained DAG, whose simplified² adjacency matrix \mathbf{T} and transitive closure matrix \mathbf{D} (obtained with (6)) are the following:

$$\mathbf{T} = \begin{matrix} S & \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ \tau_{0,0} & 0 & 0 & 1 & 0 & 1 & 0 \\ \tau_{0,1} & 0 & 0 & 0 & 1 & 0 & 0 \\ \tau_{0,2} & 0 & 0 & 0 & 0 & 0 & 1 \\ \tau_{1,0} & 0 & 0 & 0 & 1 & 0 & 1 \\ \tau_{2,0} & 0 & 0 & 0 & 0 & 0 & 1 \\ E & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{D} = & \begin{matrix} S & \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ \tau_{0,0} & 0 & 0 & 1 & 1 & 1 & 1 \\ \tau_{0,1} & 0 & 0 & 0 & 1 & 0 & 0 \\ \tau_{0,2} & 0 & 0 & 0 & 0 & 0 & 1 \\ \tau_{1,0} & 0 & 0 & 0 & 1 & 1 & 1 \\ \tau_{2,0} & 0 & 0 & 0 & 0 & 0 & 1 \\ E & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix} \end{matrix}$$

4) *Reduction*: While constructing the DAGs, it is possible to end up generating redundant edges. There is a redundant edge between two nodes when there exist both a direct edge and a non-direct path. Redundant edges can be removed using a technique called transitive reduction, firstly proposed by Aho et al. in [23]. The transitive reduction of a DAG uniquely describes the sub-graph of this DAG with the fewest possible edges, while maintaining the same reachability relation.

The transitive reduction of a DAG can be calculated in different ways. Since in this work we need the transitive reduction as well as the transitive closure of the DAG, we compute the transitive reduction using

$$\mathbf{T}_r = \mathbf{T} \wedge \neg(\mathbf{T} \cdot \mathbf{D}), \quad (18)$$

where $(\mathbf{T} \cdot \mathbf{D})$ has 1 in (j, i) if the node j can reach the node i in more than one step, 0 otherwise. Applying equation (18)

²Without synchronization and dummy nodes, removed for a clearer representation, but used in the actual algorithm.

means removing direct edges $e(v_j, v_i)$ in \mathbf{T} that are redundant because a non-direct path already exists between node j and node i .

In Figure 3d, the obtained DAG with reduced edges is presented. For that example, the matrix $\mathbf{T} \cdot \mathbf{D}$ and \mathbf{T}_r (obtained with (18)) are the following³:

$$\mathbf{T} \cdot \mathbf{D} = \begin{matrix} S & \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ \tau_{0,0} & 0 & 0 & 0 & 1 & 0 & 1 \\ \tau_{0,1} & 0 & 0 & 0 & 0 & 0 & 1 \\ \tau_{0,2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \tau_{1,0} & 0 & 0 & 0 & 0 & 0 & 1 \\ \tau_{2,0} & 0 & 0 & 0 & 0 & 0 & 0 \\ E & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{T}_r = & \begin{matrix} S & \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \tau_{0,0} & 0 & 0 & 1 & 0 & 1 & 0 \\ \tau_{0,1} & 0 & 0 & 0 & 1 & 0 & 0 \\ \tau_{0,2} & 0 & 0 & 0 & 0 & 0 & 1 \\ \tau_{1,0} & 0 & 0 & 0 & 1 & 0 & 1 \\ \tau_{2,0} & 0 & 0 & 0 & 0 & 0 & 1 \\ E & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix} \end{matrix}$$

The interaction of the different *data edges* during the Permutation stage can result in DAGs that are inherently not schedulable. These DAGs can be removed to speed up the analysis. Then, two factors are further inspected: potential cycles in the generated DAG, and length of the longest chain.

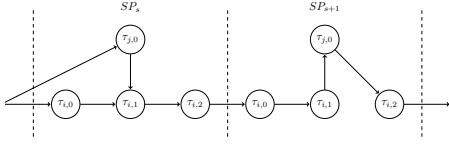
DAGs containing cycles need to be removed, as they are inconsistent with the task semantics and they could not be feasibly scheduled. Finding cycles in a graph is a common problem which can be solved with several approaches. In this work, we use state propagation, described in Section IV-B. We adopt a state vector \mathbf{x} , whose elements indicate whether a path exists (1) or not (0). Initially, $\mathbf{x}_0 = \mathbf{1}$ to consider the potential paths from all the nodes. Then, we apply state propagation in Equation (7), multiplying the state vector with the adjacency matrix \mathbf{T} . This means stepping from a node to its successor: if it has any, the resulting vector will have a 1 in the corresponding position, otherwise it will have a 0. Repeating this operation means going through all the possible paths. Since the graph is acyclic and it has n nodes, there should be no path with a length greater than n . In other words, the resulting vector should have all 0's after at most n steps, indicating that all the paths have ended, i.e., there are no more nodes to step into. If this is not the case, it means the DAG contains cycles, and it can be discarded.

Finally, the longest chain in the DAG corresponds to the chain with the longest execution time. This chain can be explicitly found by calculating the fixed-point of (9) with $\mathbf{c} = \mathbf{WC}$, the WCET of each node. If any value in $\mathbf{v}^* + \mathbf{WC}$ is bigger than the hyper-period HP , it means that there exists a path whose sum of WCETs exceeds the hyper-period, which makes the DAG not schedulable. Also these DAGs are discarded.

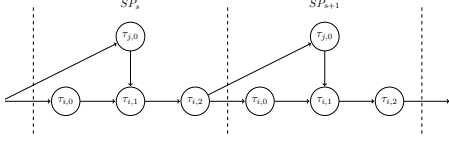
In the example, the vectors of \mathbf{WC} and \mathbf{v}^* are the following:

$$\mathbf{WC} = \begin{bmatrix} 0 \\ 7 \\ 7 \\ 7 \\ 10 \\ 13 \\ 0 \end{bmatrix} \quad \mathbf{v}^* = \begin{bmatrix} 30 \\ 23 \\ 7 \\ 0 \\ 10 \\ 0 \\ 0 \end{bmatrix}$$

³Given the previously mentioned simplification, consecutive jobs of the same task have precedence constraints between them, rather than having edges to and from synchronization nodes. In the example, there is an edge from $\tau_{0,0}$ to $\tau_{0,1}$ and one from $\tau_{0,1}$ to $\tau_{0,2}$.



(a) Heterogeneous arrangement example with $\mathcal{A}_{x,y}(s) = (0, 2, 1)$ and $\mathcal{A}_{x,y}(s+1) = (2, 1, 0)$



(b) Homogeneous arrangement example with $\mathcal{A}_{x,y}(s) = \mathcal{A}_{x,y}(s+1) = (0, 2, 1)$

Fig. 6: Examples showing heterogeneous and homogeneous arrangements.

B. Permutation Space Reduction

The worst-case number of permutations, and thus the total number of DAGs created, is given in (17), i.e., it is scaling exponentially with the size of the task set. Therefore, a reduction of the permutation space is essential to keep the approach computationally tractable for larger task sets. To reduce the permutation space, we inspect the inter-super-period-arrangement.

Given the previously adopted tasks τ_0 and τ_1 , Figure 6 shows two possible arrangements, omitting synchronization nodes for simplicity.

Let us consider a couple of harmonic tasks τ_x , a τ_y in their super-period $SP_{x,y}$. For a given parallelism ψ_s , relative to job $\tau_{y,s}$, the execution order of the parallel jobs is not defined. Therefore, a bounded number of jobs of τ_x , denoted as $prePar_s \in \{0, \dots, \psi_s\}$, can execute before $\tau_{y,s}$. Consequently $\psi_s - prePar_s$ jobs of τ_x will execute after $\tau_{y,s}$. The probability distribution of $prePar_s$ is not relevant since the only values that affect the latency variability are, by definition, the extremes, i.e.,

$$\max(prePar_s) = \psi_s \text{ and } \min(prePar_s) = 0 \quad (19)$$

Based on these definitions, the number of jobs of τ_x between two consecutive jobs $\tau_{y,s}$ and $\tau_{y,s+1}$ (in the following super-period) is given by

$$n_{s,s+1} = post_s + (\psi_s - prePar_s) + pre_{s+1} + prePar_{s+1} \quad (20)$$

The upper and lower bound of this value are given by

$$\max(n_{s,s+1}) = post_s + \psi_s + pre_{s+1} + \psi_{s+1} \quad (21)$$

$$\min(n_{s,s+1}) = post_s + pre_{s+1} \quad (22)$$

The *variability* of the data updates, i.e., the difference between the maximum and the minimum data updates in between, can be formalized as:

$$Var_{x,y} = \max_s \{ \max(n_{s,s+1}) \} - \min_t \{ \min(n_{t,t+1}) \}, \quad (23)$$

using t in the minimum to highlight that the maximum and minimum do not need to consider the same job of τ_y ,

and thus the same arrangement. However, in a homogeneous arrangement, $\mathcal{A}_{x,y}(s) = \mathcal{A}_{x,y}(s+1) = (pre, post, \psi)$ and $s = t$. Therefore,

$$Var_{x,y,hom} = 2\psi_s \quad (24)$$

Comparing to heterogeneous arrangements, in which $\mathcal{A}_{x,y}(s) \neq \mathcal{A}_{x,y}(s+1), \forall s$, two observations can be made. On the one hand, a higher value of ψ_s for a job $\tau_{y,s}$ increases schedulability of the related super-period, since it allows for more parallelism and shortens the longest path. Given that the schedulability of all the super-periods determines the schedulability of the hyper-period, the value of ψ_s is crucial. On the other hand, from an application side, the data update *variability* should be as low as possible to constrain end-to-end latency.

To reduce the permutation space while investigating all the permutations that optimize latency, we chose to sacrifice optimality w.r.t. schedulability. Homogeneous arrangements are better at this compromise. To show it, we prove that

$$Var_{x,y,heter} > Var_{x,y,hom} \quad (25)$$

Proposition: Given two tasks τ_x and τ_y with periods $gT_x = T_y$, $hT_y = HP$, $g, h \in \mathbb{N}_+$, a heterogeneous arrangement results in a strictly higher *variability* than a homogeneous arrangement.

Proof: In a heterogeneous arrangement, $\mathcal{A}_{x,y}(s) \neq \mathcal{A}_{x,y}(s+1)$, which means that $(pre_s, post_s, \psi_s) \neq (pre_{s+1}, post_{s+1}, \psi_{s+1})$. Let us define $\alpha_s \in \mathbb{Z}$ (resp. $\beta_s \in \mathbb{Z}$) as the difference between the jobs of τ_x that execute before (resp. after) $\tau_{y,s+1}$ and the jobs of τ_x that execute before (resp. after) $\tau_{y,s}$ ⁴

$$\alpha_s = pre_{s+1} - pre_s \quad (26)$$

$$\beta_s = post_{s+1} - post_s \quad (27)$$

Consequently, considering that $pre_s + post_s + \psi_s = pre_{s+1} + post_{s+1} + \psi_{s+1} = Q$, we derive

$$\psi_{s+1} = \psi_s - \alpha_s - \beta_s \quad (28)$$

With this definition, Equation (20) provides

$$\begin{aligned} n_{s,s+1} &= post_s + (\psi_s - prePar_s) + pre_{s+1} + prePar_{s+1} \\ n_{s,s+1} &= post_s + (\psi_s - prePar_s) + pre_s + \alpha_s + prePar_{s+1} \\ &= Q + \alpha_s - prePar_s + prePar_{s+1}. \end{aligned}$$

Then,

$$\max(n_{s,s+1}) = Q + \alpha_s + \psi_{s+1}$$

$$= Q + \psi_s - \beta_s$$

$$\min(n_{s,s+1}) = Q + \alpha_s - \psi_s$$

⁴Remember that there is only one job of $\tau_{y,s}$ in each super-period $SP_{x,y}$. Therefore, $\tau_{y,s+1}$ refers to the next super-period.

The *variability* in Equation (23) can then be simplified to

$$\begin{aligned} Var_{x,y,het} &= \max_s \{ \max(n_{s,s+1}) \} - \min_t \{ \min(n_{t,t+1}) \} \\ &= \max_s \{ \psi_s - \beta_s \} - \min_t \{ \alpha_t - \psi_t \} \\ &= \max_s \{ \psi_s - \beta_s \} + \max_t \{ \psi_t - \alpha_t \} \\ &= 2\psi_s + \max_s \{ -\beta_s \} + \max_t \{ -\alpha_t \}. \end{aligned}$$

As the full arrangement is the same in each hyper-period, the super-period arrangement is cyclic. Since α_s and β_s denote the change of the arrangement, the cyclicity of \mathcal{A} requires

$$\sum_{s \in \{0, \dots, \frac{HP}{T_y}\}} \alpha_s = \sum_{s \in \{0, \dots, \frac{HP}{T_y}\}} \beta_s = 0. \quad (29)$$

Therefore, $\exists \alpha_s < 0$ and $\exists \beta_s < 0$ such that

$$Var_{x,y,het} > 2\psi_s \quad (30)$$

Since $Var_{x,y,het} > 2\psi_s$, it then follows $Var_{x,y,het} > Var_{x,y,hom}$, proving the proposition. \square

We can therefore omit heterogeneous arrangements without affecting the resulting end-to-end latency, since no such arrangement can provide a better compromise with respect to *variability*. By discarding the heterogeneous arrangements in the permutations, the value of $perm_{total}$ in (17) can be reduced to

$$perm_{total} = \prod_{\forall x,y} perm_{SP_{x,y}}, \quad (31)$$

where $x \neq y$ and τ_x and τ_y are consecutive tasks in a given task chain, and the full hyper-period arrangement is defined by a unique super-period arrangement. This is valid both for harmonic and non-harmonic tasks.

C. Computational Complexity

The computational cost of the overall method can be summarized as $\mathcal{O}(perm_\psi \times perm_{\mathcal{A}} \times n^4)$. The first term $perm_\psi$ represents all the permutations for all the possible ψ values. From (31), it can be expressed as

$$perm_\psi = \prod_{(e_x, e_y) \in E} \frac{\max(T_x, T_y)}{\min(T_x, T_y)}. \quad (32)$$

The second term $perm_{\mathcal{A}}$ represents all the arrangement permutations for a fixed ψ . From (14), it can be expressed as

$$perm_{\mathcal{A}} = \prod_{(e_x, e_y) \in E} \left(\frac{\max(T_x, T_y)}{\min(T_x, T_y)} - \psi + 1 \right). \quad (33)$$

Lastly, $\mathcal{O}(n^4)$ is the maximum cost of all the math operations applied on the obtained DAGs, which are matrix-vector multiplication $\mathcal{O}(n^2)$, matrix multiplication $\mathcal{O}(n^3)$ and matrix exponentiation $\mathcal{O}(n^4)$. Let us define R as the maximum ratio between periods of the taskset, i.e., $R = \frac{\max(T_x)}{\min(T_y)} \forall x, y \in \{0 \dots N-1\}$. The computational cost of the method can then be expressed as:

$$\mathcal{O}(R^{|E|} R^{|E|} (RN)^4) = \mathcal{O}(R^{2|E|} (RN)^4). \quad (34)$$

job	EST	LST	EFT	LFT
$\tau_{0,0}$	0	0	5	7
$\tau_{0,1}$	10	13	15	20
$\tau_{0,2}$	20	23	25	30
$\tau_{1,0}$	5	7	15	20
$\tau_{2,0}$	15	20	23	30

TABLE I: Timing attribute for the for Example 1.

The complexity is thus exponential in the number of edges $|E|$. Such a high cost is mainly determined by the need to take into account all the permutations at once. However, this is also the reason why the proposed conversion method allows better controlling end-to-end latencies, jointly optimizing data and reaction times of all the task chains given in input. This is achieved by picking up the best configuration out of all the permutations generated by means of a cost function.

VI. END-TO-END LATENCY AND SCHEDULABILITY

In this section, a method to calculate an upper bound on *data age* and *reaction time* is proposed. As explained in the introduction, data age defines the maximum time a data produced by the first task of the chain can influence the last one. Reaction time is the maximum interval between the acquisition of a stimulus in the first task of a chain and the moment the first instance of the last task in the chain reacts to it.

We first define a set of additional timing attributes, that will be used to compute the end-to-end latency. The schedulability of the DAG is verified by deriving a static schedule. If more than one generated DAG meets the latency and schedulability constraints, we select the DAG that maximizes a weighted sum of the end-to-end latencies, taking into account all the tasks chains in input.

For each job, we define the following timing attributes: Earliest Finishing Time (EFT), Latest Finishing Time (LFT), Earliest Starting Time (EST) and Latest Starting Time (LST). The earliest a node can start is the maximum of all its predecessors' earliest finishing times. Similarly, the latest a node can finish is the minimum of its successors' latest starting times. These values can be iteratively calculated using the operators defined in Section IV, initializing $EST_j = 0$ and $EFT_j = HP$ for all nodes j :

$$\begin{aligned} EST_i &= \max_{\forall j} \{ (EST_j + BC_j) \mathbf{T}_{j,i} \} \\ LFT_i &= \min_{\forall j} \{ (LFT_j - WC_j) \mathbf{T}_{i,j} \} \\ EFT_i &= EST_i + BC_i \\ LST_i &= LFT_i - WC_i. \end{aligned}$$

Table I reports the timing attributes computed for Example 1.

A. Task Chain Propagation

In a DAG \mathcal{G} , a node j is defined to *react* to node i if there exists a direct or indirect edge from node i to node j . A node k *reacting* to node j also *reacts* to node i . Further, a node k *reacts* to the chain (i, j) if node j *reacts* to node i and node k *reacts* to node j .

Extending this definition to tasks and jobs:

- $\tau_{y,b}$ reacts to $\tau_{y,a}, \forall b > a$;
- Consequently, if $\tau_{y,a}$ reacts to $\tau_{x,c}$, it follows that $\tau_{y,b}$ reacts to $\tau_{x,c}$.

Given a task chain (τ_x, \dots, τ_z) , the *reactions* of jobs of task τ_z to each job of τ_x can be found. Consider a job $\tau_{x,a}$ of the first task in the chain. The *first* (resp. *last*) *reaction* to $\tau_{x,a}$ is defined as the first (resp. last) job of the last task τ_z that reacts to $\tau_{x,a}$. The *reaction time* (resp. *data age*) is then defined as the maximum interval between a stimulus in a job $\tau_{x,a}$ and the finishing time of the first (resp. last) reaction, taken over all instances $\tau_{x,a}$, for all $a \in [0, \frac{HP}{T_x}]$. Since the structure of the DAG repeats after each hyper-period, it is sufficient to consider only the first hyper-period.

Algorithm 1: findReactions

Input: $C = \{\tau_{start}, \dots, \tau_{end}\}$
Output: $1^{st}reactions, lastreactions$

```

1 forall  $a \in \{0, \dots, \frac{HP}{T_{start}} + 1\}$  do
2    $fr\_job = \tau_{start,a}$ ;
3    $lr\_job = \mathbf{null}$ ;
4   forall  $\tau_x \in C \setminus \tau_{start}$  do
5      $b = 0$ ;
6     while  $\tau_{x,b}$  does not react to  $fr\_job$  do
7        $b++$ ;
8        $fr\_job = \tau_{x,b}$ ;
9       if  $b > 0$  then
10         $lr\_job = \tau_{x,b-1}$ ;
11  if  $a \leq \frac{HP}{T_{start}}$  then
12     $1^{st}reactions.insert(\tau_{start,a}, fr\_job)$ ;
13  if  $(b \neq \mathbf{null})$  and  $(a > 0)$  and
     $(1^{st}reactions(\tau_{start,a-1}) \neq fr\_job)$  then
14     $lastreactions.insert(\tau_{start,a-1}, lr\_job)$ ;
15 return  $1^{st}reactions, lastreactions$ ;
```

A method to compute the *first* and *last reactions* is shown in Algorithm 1. The algorithm considers every job of the starting task of the chain in one hyper-period, plus an additional job (to cover the last reactions). The first reacting job (fr_job) is set to $\tau_{start,a}$. Then, for each task in the chain, we find the first job $\tau_{x,b}$ that reacts to fr_job , and we use it to update fr_job . This can happen either in the same hyper-period of fr_job , or in the next one. The preceding job $\tau_{x,b-1}$ is instead used to update lr_job , which keeps track of the last reaction to $\tau_{start,a-1}$. Once the whole chain has been considered, $1^{st}reactions$ and $lastreactions$ are updated. The latter is updated only if b is not null and if the first reaction to $\tau_{start,a}$ is different from the first reaction to $\tau_{start,a-1}$. *Reaction time* and *data age* can then be simply derived as

$$RT = \max_{\tau_{x,a} \in 1^{st}reactions} \{LFT_{1^{st}reactions} - EST_{\tau_{x,a}}\} \quad (35)$$

$$DA = \max_{\tau_{x,a} \in lastreactions} \{LFT_{lastreaction} - EST_{\tau_{x,a}}\}, \quad (36)$$

i.e., reaction time (resp. data age) is the difference between the first (resp. last) moment some data is used by a job of the last task in the chain (LFT) and the first moment the same data is read from the job of the first task in the chain (EST). Since the schedule repeats identically after each hyper-period, it is sufficient to consider all the jobs of the first task in the first hyper-period.

We hereafter prove that Algorithm 1 correctly finds the first and last reactions. The algorithm considers all the jobs of the starting task in the chain (line 1). For each of the starting jobs, it iterates over all the other tasks in the chain, always looking for the first and last reacting job (lines 4-14). Let us consider two consecutive tasks in the chain τ_x and τ_y and only one job a of τ_x .

- To find the maximum reaction time, the jobs of τ_y that are said to react to $\tau_{x,a}$ are those that are definitely executing after $\tau_{x,a}$, i.e., they belong to $\tau_{x,a}$'s descendants, or their *EST* is greater than the *LFT* of $\tau_{x,a}$. Since the DAG is schedulable, a reacting job can always be found (and the loop at line 6 is not infinite) either in the same hyper-period of $\tau_{x,a}$, or in the next one. Once a job $\tau_{x,b}$ is found to react to $\tau_{x,a}$, it becomes the starting job to find the first reaction between τ_y and the next task in the chain.
- The maximum data age of $\tau_{x,a}$ is strictly related to the first reaction to $\tau_{x,a+1}$. Indeed, the first reaction to $\tau_{x,a+1}$ assures that the data from $\tau_{x,a}$ are no longer used: the last time they were used was by the job preceding the one that surely reacts to $\tau_{x,a+1}$. Thus, when finding the first reaction $\tau_{x,a+1}$, the last reaction of $\tau_{x,a}$ can be found (line 14).

In the example DAG in Figure 3d, *data age* is 30, while *reaction time* is 50. The chains leading to these values are $\{\tau_{0,0}, \tau_{1,0}, \tau_{2,0}\}$ for *data age* and $\{\tau_{0,1}, \tau'_{1,0}, \tau'_{2,0}\}$ for *reaction time*, where a prime indicates that the job is in the next hyper-period.

B. Schedulability

To build a feasible schedule for a given number of cores, we apply a list-scheduling heuristic for non-preemptive DAG, very similar to the Heterogeneous Earliest Finishing Time (HEFT) algorithm presented in [24]. We decided to use a (node-level) limited preemptive scheduling for (i) avoiding job-level migrations, (ii) reducing cache-related preemption delays, and (iii) minimizing the input-output delay and jitter [25].

The list-scheduling algorithm is summarized in Algorithm 2. Jobs are sorted in increasing LFT order (line 3). Given p homogeneous processors, a job is scheduled at time t only if it is ready and a processor is available. A job enters the ready queue (line 8) at time t only if (i) its *EST* is greater than or equal than t , (ii) all its predecessors in the DAG have been executed, and (iii) its *LFT* is the smallest between all the remaining jobs' *LFT*. The ready queue is sorted in increasing LFT order (line 9). A ready job is scheduled if a processor is available and if its execution time, starting from the current t , does not exceed its *LFT* (lines 13,16,17). If

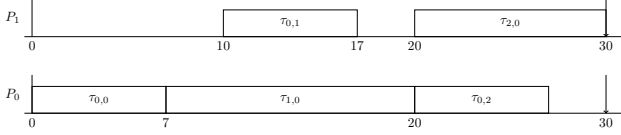


Fig. 7: Schedule produced for the DAG in Figure 3d with 2 cores

this last condition is not met, the algorithm declares the DAG not schedulable (lines 13,14). An example of the schedule obtained for Example 1 is shown in Figure 7.

Algorithm 2: isDAGSchedulable

Input: $V, pred, p, EST, LFT, WC$

Output: *true* if the DAG is schedulable on p processors, *false* otherwise

Data: Ready queue of jobs $rq = \{\}$, procExec vector

```

1  $pq_i = \{\}, \forall i = 1, \dots, p;$ 
2  $nodes = \{v_0, \dots, v_{n-1}\}, n = |V|;$ 
3  $sort(nodes)$  sort by ascending value of
  LFT
4 for  $t = 0, 1, \dots, HP$  do
5   forall  $node \in nodes$  do
6     if  $EST[node] > t$  and all  $pred[node]$  have
       finished and  $LFT[node] \leq$  all other nodes
       LFTs then
7        $nodes = nodes \setminus node;$ 
8        $rq.push(node);$ 
9    $sort(rq)$  sort by ascending value of
     LFT
10  for  $i = 1, \dots, p$  do
11    if  $rq \neq \{\}$  and  $procExec_i == 0$  then
12       $readyJob = rq.pop();$ 
13      if  $t + WC[readyJob] > LFT[readyJob]$ 
14        then
15          return false;
16        else
17           $procExec_i = WC[readyJob];$ 
18           $pq_i.push(readyjob);$ 
19    if  $procExec_i > 0$  then
20       $procExec_i = procExec_i - 1;$ 
21 return true;

```

VII. EVALUATION

To evaluate our approach, we first use simulation to validate end-to-end latency bounds as well as schedulability and then compare the proposed method with the state-of-the-art using a realistic automotive benchmark.

A. Evaluation via Simulation

To validate that the DAGs generated with the method presented in this paper comply with the constraints, we developed

Taskset Γ	
$\tau_i = (WC_i, BC_i, P_i, D_i)$	Task
$\tau_0 = (7, 5, 50, 50)$	GPS
$\tau_1 = (12, 10, 50, 50)$	Lidar
$\tau_2 = (28, 22, 50, 50)$	Localization
$\tau_3 = (28, 25, 50, 50)$	Detection
$\tau_4 = (25, 18.9, 50, 50)$	Fusion
$\tau_5 = (2, 1.8, 25, 25)$	Camera
$\tau_6 = (6.5, 3, 10, 10)$	EKF
$\tau_7 = (5, 3.2, 10, 10)$	Planner
$\tau_8 = (4.5, 1.8, 10, 10)$	Control
Task chains	
chain $\{\tau_{start}, \dots, \tau_{end}\}$	(Age, Reaction)
$\{\tau_5, \tau_3, \tau_4\}$	(120, 120)
$\{\tau_0, \tau_2, \tau_6, \tau_7, \tau_8\}$	(120, 150)
$\{\tau_1, \tau_2, \tau_6, \tau_7, \tau_8\}$	(120, 150)
$\{\tau_5, \tau_3, \tau_4, \tau_7, \tau_8\}$	(150, 150)
Scheduling constraints	
6 processors	

TABLE II: Periodic taskset and constraints used for the simulation, referring to the application of Figure 1.

a simulation tool. The tool uses the DAG to schedule the individual tasks, which tracks the data propagation through the task chains under analysis. The execution time of each task is identically and independently sampled from the BC to WC interval. The schedule is generated according to EDF, and the deadline is set equal to LFT.

We simulated the best DAG, in terms of schedulability and end-to-end latency, produced for the application introduced in Figure 1. The task set specification and constraint are detailed in Table II. The latency computed for the given chains is reported in Table III.

Using the simulation tool, the DAG is simulated for 10^9 ms, which leads to the following results. Two distributions of *reaction time* and *data age* of two task chains are shown in Figure 8 and 9. In Figure 8 the *reaction time* plot is showing two distributions, one for each camera frame. In the DAG, the camera jobs are serialized to the detection job, leading to only one distribution for the *data age*, because the detection job always receives the freshest camera frame. A similar distribution for the *reaction time* can be seen for the task chain in Figure 9, as the task chain, is extended with the planner and control task. The *data age*, however, shows several distributions. This is due to the higher rate of the planner and control task with respect to the fusion task. Nevertheless, the *data age* of the data corresponding to each control output is always based on the freshest camera frame, which can be seen by comparing the distribution shapes. The simulation showed that all the calculated upper bounds for *data age* and *reaction time* for the four task chains are not exceeded.

B. Evaluation via Benchmark

To further analyze the performance of the proposed method the detailed automotive benchmark proposed by BOSCH for the WATERS challenge in 2015 [26] has been adopted. Multi-rate periodic task sets and cause-effect chains are randomly generated while conforming with the char-

chain $\{\tau_x, \dots, \tau_y\}$	(Age, Reaction)
$\{\tau_5, \tau_3, \tau_4\}$	(75, 98.2)
$\{\tau_0, \tau_2, \tau_6, \tau_7, \tau_8\}$	(105, 65)
$\{\tau_1, \tau_2, \tau_6, \tau_7, \tau_8\}$	(105, 65)
$\{\tau_5, \tau_3, \tau_4, \tau_7, \tau_8\}$	(125, 108.2)

TABLE III: Maximum *data age* and *reaction time* for task chains of the best DAG produced for the task set described by Table II

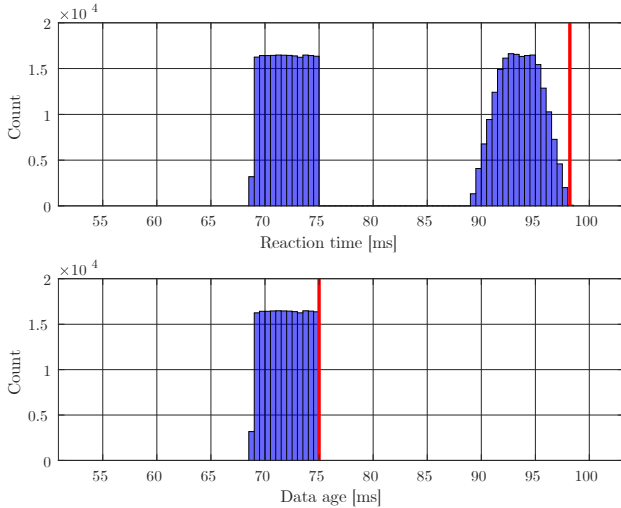


Fig. 8: *Reaction time* and *data age* of the chain {Camera, Detection, Fusion}, or $\{\tau_5, \tau_3, \tau_4\}$, evaluated in simulation with the red lines showing the calculated maximum.

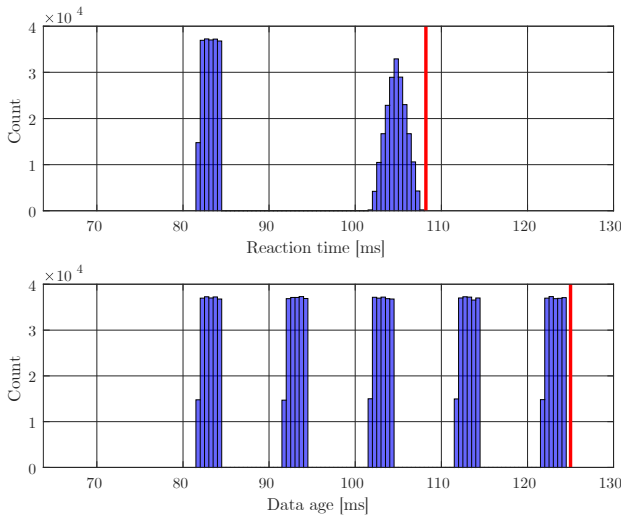


Fig. 9: *Reaction time* and *data age* of the chain {Camera, Detection, Fusion, Planner, Control}, or $\{\tau_5, \tau_3, \tau_4, \tau_7, \tau_8\}$, evaluated in simulation with the red lines showing the calculated maximum.

	permutation	admissible (%)	schedulable (%)
min	0.00	0.00	0.00
avg	1.830.48	62.10	61.60
max	18.148.00	100.00	100.00

TABLE IV: Statistics about DAG permutations, admissible and schedulable DAGs on 1000 different task set.

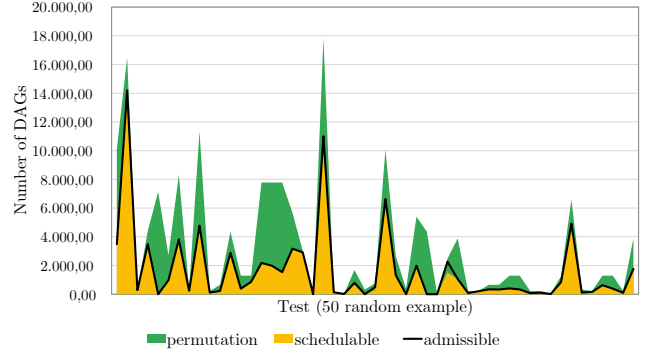


Fig. 10: Statistic about produced DAG on 50 randomly chosen task set of the 1000 analyzed.

acterization. Task periods are selected with given distribution, out of the periods found in automotive applications [1, 5, 10, 20, 50, 100, 200, 1000]ms. Cause-effect chains are generated to include tasks of either 1, 2, or 3 different period wherein tasks of the same period can appear 2 to 5 times. To obtain a higher utilization, the individual task execution times are generated based on UUniFast [27]. For the experiments 1000 task set composed of 5 tasks and 15 chains have been taken into account, with a utilization equal to 1.5, considering 2 cores available.

Table IV reports some statistics about the DAG obtained from the 1000 multi-rate periodic task sets. From the initial generated permutations the 40% is on average removed due to cycles or a non-schedulable longest chain. However, between the admissible generated DAGs⁵, almost the totality is also schedulable on 2 cores. Figure 10 shows 50 randomly selected examples in which the number of permutations, admissible DAGs and schedulable DAGs are compared.

C. Comparison with state-of-the-art

1) *Qualitative*: In [19], Saidi et al. present a method to convert a parallel multi-rate task set with precedence and data edges into a single-rate DAG. However, the end-to-end latency is not considered, and only one possible DAG is generated. Therefore, no guarantee is given on *reaction time* or *data age*. Moreover, there are no synchronization methods to force task instances to execute within their periods, potentially leading to a wrong implementation of the system.

In [18], Foget et al. present another conversion method, producing different DAGs. However, neither this work takes into account latency. Different solutions are created just for

⁵DAGs that have a correct structure (i.e., no cycles and no path greater than the hyper-period) but that may still be unschedulable.

	Forget [18]	Saidi [19]	Becker [9]	this paper
schedulable task set(%)	46.9	21.8	90.5 ⁶	90.5
1st lowest data age (%)	45.89	17.85	77.81	96.82
2nd lowest data age (%)	2.79	0.09	13.55	3.15
3rd lowest data age (%)	3.03	1.46	6.38	0.03
4th lowest data age (%)	0.00	4.58	2.25	0.00

TABLE V: Schedulability and *data age* results on 1000 task set compliant to [26] of 5 tasks and 15 chains, with utilization equals to 1.5.

	Forget [18]	Saidi [19]	Becker [9]	this paper
min [ms]	0.002	0.002	0.078	0.002
avg [ms]	0.571	0.022	3.001	21.410
max [ms]	4.422	0.116	16.614	433.033

TABLE VI: Execution times in milliseconds on an Intel(R) Core(TM) i7-7700HQ CPU @2.80GHz.

schedulability reasons, picking the version that makes the DAG schedulable with EDF.

Focusing on data latency, the most related approach is the one introduced by Becker et al. in [9], where the focus is on the computation of *data age*. In their work, they can compute *data age* for a given chain of periodic tasks, given a communication model (i.e. implicit, explicit or LET). The method allows generating job-level dependencies to meet latency requirements. However, *data age* is the only parameter under their analysis. Moreover, they can optimize end-to-end latency for only a single chain. Once job-level dependencies are inserted, all the other chains are affected. Finally, their work assumes the input task set is already schedulable. Our work has several improvements over their approach, i.e., (i) the model is more general and can jointly optimize the latency of multiple chains, (ii) we consider not only *data age*, but also *reaction time*, and (iii) our task allocation and scheduling algorithms also consider the schedulability of the system.

2) *Quantitative*: To show that our method dominates the state-of-the-art, we implemented the solutions proposed in [19], [18], [9], and tested them on the previously presented automotive benchmark by BOSCH. Table V shows the results for the 1000 task sets considered, and all the 15000 task chains, while Table VI offers a comparison of the running times of the considered methods for larger task sets, i.e., composed of 10 tasks with 15 task chains.

The proposed method not only dominates the others in term of schedulability, but also in terms of *data age*. Given that [18] and [19] do not propose a method to compute end-to-end latency, we adopted our algorithm for this scope. Considering *data age*, our method produces a DAG that leads to the lowest end-to-end latency bound in 96.82% of cases. There are some cases in which Becker [9] method obtains a tighter latency, since it optimizes a single chain. However, the limitation of that approach is that it is not able to optimize all the given chains for a task set, while our method optimizes them all. Therefore, we are willing to sacrifice the latency of some chains for a more balanced improvement of all chains.

⁶Since no method is proposed in [9] to check schedulability, we applied our method to derive the schedulable task sets.

On the other hand, when optimizing a single chain, our method allows finding a better solution than with the method presented in [9]. As an example, consider the task chain in Example 1. Using the method by Becker et al., a minimum *data age* of 40 can be achieved, inserting a precedence constraint between $\tau_{1,0}$ and $\tau_{2,0}$. Instead, our method allows achieving a *data age* of 30, picking a DAG with additional precedence constraints.

As can be expected, the improved performance of the proposed algorithm are obtained by paying a somewhat higher computational cost. Table VI shows that our method is on average about 7 times slower than [9]. We believe such a slowdown is acceptable for an offline analysis performed at system design time, as it allows obtaining the best solution for even complex task systems within a reasonable time.

VIII. CONCLUSIONS

This paper presented a detailed method that allows converting a multi-rate task set into a single-rate DAG which meets schedulability and timing requirements. To the best of our knowledge, this approach is the most general and complete w.r.t. the methods available in the literature. The transformation process maps the whole application into a DAG, using precedence constraints for synchronizing jobs to comply with task activation periods. Multiple DAGs are generated in four stages and a pruning process is applied to exclude the ones that are inherently not feasible. The set of feasible DAGs is narrowed down using a further analysis that considers *data age* and *reaction time* bounds on specific task chains, as well as the schedulability of the system on the considered homogeneous multi-core platform. The best DAG is selected based on the weighted sum of end-to-end latencies. Most of the operations performed are based on a matrix representation of the DAG. The conversion method and a simulation tool have been implemented and made available⁷. The efficiency of the proposed approach over existing methods has been extensively validated on real experimental benchmarks. In future works, we plan to extend this model to heterogeneous platforms. Moreover, we plan to integrate the proposed approach considering predictable execution models to solve the contention problems on the memory hierarchy [28].

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union’s Horizon 2020 Programme under the CLASS Project (<https://class-project.eu/>), grant agreement n° 780622. Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

⁷<https://github.com/mive93/multi-rate-DAG>

REFERENCES

- [1] A. Hamann, D. Dasari, S. Kramer, M. Pressler, F. Wurst, and D. Ziegenbein, “Waters industrial challenge 2017,” 2017.
- [2] A. Hamann, F. Dasari, Dakshina Wurst, I. Sañudo, N. Capodiecì, P. Burgio, and M. Bertogna, “Waters industrial challenge 2019,” 2017.
- [3] A. Vincentelli, P. Giusto, C. Pinello, W. Zheng, and M. Natale, “Optimizing end-to-end latencies by adaptation of the activation events in distributed automotive systems,” in *13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS’07)*. IEEE, 2007, pp. 293–302.
- [4] M. D. Natale, H. Zeng, P. Giusto, and A. Ghosal, *Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice*. Springer Publishing Company, Incorporated, 2012.
- [5] H. Zeng, M. D. Natale, P. Giusto, and A. L. Sangiovanni-Vincentelli, “Statistical analysis of controller area network message response times,” in *IEEE Fourth International Symposium on Industrial Embedded Systems, SIES 2009, Ecole Polytechnique Federale de Lausanne, Switzerland, July 8-10, 2009*, 2009, pp. 1–10.
- [6] H. Zeng, “Probabilistic timing analysis of distributed real-time automotive systems,” Ph.D. dissertation, EECS Department, University of California, Berkeley, Dec 2008.
- [7] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, “A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics,” in *IEEE Real-Time Systems Symposium: 30/11/2009-03/12/2009*. IEEE Communications Society, 2009.
- [8] “Autosar - specification of timing extensions,” Tech. Rep., 2014.
- [9] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, “Synthesizing job-level dependencies for automotive multi-rate effect chains,” in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2016, pp. 159–169.
- [10] —, “End-to-end timing analysis of cause-effect chains in automotive embedded systems,” *Journal of Systems Architecture*, vol. 80, pp. 104–113, 2017.
- [11] —, “Mechaniser-a timing analysis and synthesis tool for multi-rate effect chains with job-level dependencies,” in *7th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems WATERS*, vol. 16, no. 05, 2016.
- [12] A. Biondi and M. D. Natale, “Achieving Predictable Multicore Execution of Automotive Applications Using the LET Paradigm,” in *Proceedings of the 24th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2018)*, Porto, Portugal, April 2018.
- [13] J. Martinez, I. Sañudo, P. Burgio, and M. Bertogna, “End-to-end latency characterization of implicit and let communication models,” in *Proc. of the 8th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, ser. WATERS, 2017.
- [14] M. Xiong, S. Han, K.-Y. Lam, and D. Chen, “Deferrable scheduling for maintaining real-time data freshness: Algorithms, analysis, and results,” *IEEE Transactions on Computers*, vol. 57, no. 7, pp. 952–964, 2008.
- [15] D. Golomb, D. Gangadharan, S. Chen, O. Sokolsky, and I. Lee, “Data freshness over-engineering: Formulation and results,” in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2018, pp. 174–183.
- [16] Y. Saito, F. Sato, T. Azumi, S. Kato, and N. Nishio, “Rosch: Real-time scheduling framework for ros,” in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 52–58.
- [17] Y. Suzuki, T. Azumi, S. Kato *et al.*, “Hlbs: Heterogeneous laxity-based scheduling algorithm for dag-based real-time computing,” in *2016 IEEE 4th International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*. IEEE, 2016, pp. 83–88.
- [18] J. Forget, F. Boniol, E. Grolleau, D. Lesens, and C. Pagetti, “Scheduling dependent periodic tasks without synchronization mechanisms,” in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2010, pp. 301–310.
- [19] S. E. Saidi, N. Pernet, and Y. Sorel, “Automatic parallelization of multi-rate fmi-based co-simulation on multi-core,” in *Proceedings of the Symposium on Theory of Modeling & Simulation*. Society for Computer Simulation International, 2017, p. 5.
- [20] S. K. Baruah, D. Chen, S. Gorinsky, and A. K. Mok, “Generalized multiframe tasks,” *Real-Time Systems*, vol. 17, pp. 5–22, 1999.
- [21] N. Biggs, N. L. Biggs, and B. Norman, *Algebraic graph theory*. Cambridge university press, 1993, vol. 67.
- [22] P. Purdom, “A transitive closure algorithm,” *BIT Numerical Mathematics*, vol. 10, no. 1, pp. 76–94, 1970.
- [23] A. V. Aho, M. R. Garey, and J. D. Ullman, “The transitive reduction of a directed graph,” *SIAM Journal on Computing*, vol. 1, no. 2, pp. 131–137, 1972.
- [24] H. Topcuoglu, S. Hariri, and M.-y. Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.
- [25] G. Buttazzo and A. Cervin, “Comparative assessment and evaluation of jitter control methods,” in *Proceedings of the 15th conference on Real-Time and Network Systems*, 2007, pp. 163–172.
- [26] S. Kramer, D. Ziegenbein, and A. Hamann, “Real world automotive benchmarks for free,” in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.
- [27] E. Bini and G. C. Buttazzo, “Measuring the performance of schedulability tests,” *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.
- [28] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, “A predictable execution model for cots-based embedded systems,” in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2011, pp. 269–279.

6.2 Edge Generation Scheduling for DAG Tasks Using Deep Reinforcement Learning

Reference

B. Sun, M. Theile, Z. Qin, D. Bernardini, D. Roy, A. Bastoni, and M. Caccamo, “Edge generation scheduling for dag tasks using deep reinforcement learning,” *IEEE Transactions on Computers*, 2024

DOI: <https://doi.org/10.1109/TC.2024.3350243>

Abstract

Directed acyclic graph (DAG) tasks are currently adopted in the real-time domain to model complex applications from the automotive, avionics, and industrial domains that implement their functionalities through chains of intercommunicating tasks. This paper studies the problem of scheduling real-time DAG tasks by presenting a novel schedulability test based on the concept of trivial schedulability. Using this schedulability test, we propose a new DAG scheduling framework (edge generation scheduling—EGS) that attempts to minimize the DAG width by iteratively generating edges while guaranteeing the deadline constraint. We study how to efficiently solve the problem of generating edges by developing a deep reinforcement learning algorithm combined with a graph representation neural network to learn an efficient edge generation policy for EGS. We evaluate the effectiveness of the proposed algorithm by comparing it with state-of-the-art DAG scheduling heuristics and an optimal mixed-integer linear programming baseline. Experimental results show that the proposed algorithm outperforms the state-of-the-art by requiring fewer processors to schedule the same DAG tasks.

Contributions to this paper

- Conceptualization of trivial schedulability
- Shared conceptualization of the edge generation framework
- Share of the implementation of edge generation scheduling for DAG scheduling
- Share of paper writing

Copyright

© 2024 IEEE. Reprinted, with permission, from Binqi Sun, Mirco Theile, Ziyuan Qin, Daniele Bernardini, Debayan Roy, Andrea Bastoni, and Marco Caccamo, “Edge Generation Scheduling for DAG Tasks Using Deep Reinforcement Learning”, *IEEE Transactions on Computers* (Volume: 73, Issue: 4), April 2024.

See Appendix A.10 for the reuse statement. The following shows the accepted version.

Edge Generation Scheduling for DAG Tasks using Deep Reinforcement Learning

Binqi Sun, *Student Member, IEEE*, Mirco Theile, *Student Member, IEEE*, Ziyuan Qin, Daniele Bernardini, *Member, IEEE*, Debayan Roy, Andrea Bastoni, *Member, IEEE*, and Marco Caccamo, *Fellow, IEEE*

Abstract—Directed acyclic graph (DAG) tasks are currently adopted in the real-time domain to model complex applications from the automotive, avionics, and industrial domains that implement their functionalities through chains of intercommunicating tasks. This paper studies the problem of scheduling real-time DAG tasks by presenting a novel schedulability test based on the concept of *trivial schedulability*. Using this schedulability test, we propose a new DAG scheduling framework (*edge generation scheduling—EGS*) that attempts to minimize the DAG width by iteratively generating edges while guaranteeing the deadline constraint. We study how to efficiently solve the problem of generating edges by developing a deep reinforcement learning algorithm combined with a graph representation neural network to learn an efficient edge generation policy for EGS. We evaluate the effectiveness of the proposed algorithm by comparing it with state-of-the-art DAG scheduling heuristics and an optimal mixed-integer linear programming baseline. Experimental results show that the proposed algorithm outperforms the state-of-the-art by requiring fewer processors to schedule the same DAG tasks. The code is available at <https://github.com/binqi-sun/egs>.

Index Terms—DAG scheduling, real-time, edge generation, deep reinforcement learning



1 INTRODUCTION

Current real-time applications in the automotive, avionics, and industrial domains realize their functionalities through complex chains of intercommunicating tasks. For example, [1], [2] present recent driving assistance and autonomous driving applications where data is processed through multiple periodically-activated steps, from sensor data acquisition (e.g., Lidar and cameras) to actuators (e.g., brakes and steering wheel). Such applications—including their execution and precedence-constraint requirements—are modeled using directed acyclic graph (DAG) tasks with different periods that can be reduced to a *single DAG* using techniques such as [3].

Reasoning on real-time properties of DAG tasks has proved challenging. Testing the schedulability of DAG tasks is NP-hard in the strong sense [4], and many works have focused on devising heuristics for sufficient schedulability tests (see Section 2 for relevant related works and refer to [5]–[7] for a comprehensive survey), while exact results (e.g., [8]–[10]) can only be obtained for simple DAG tasks.

This work focuses on the setup where a single periodic non-preemptive DAG task is executed on a multicore platform with identical processors. Drawing from graph theory,

we develop a novel schedulability test based on the key observation that a DAG whose *width* is not greater than the number of available processors and whose *length* is less than or equal to the deadline of the DAG is schedulable. We classify such a DAG as a *trivially schedulable* DAG and show that any DAG is schedulable if and only if it can be converted into a trivially schedulable DAG by adding edges. In addition, we show that a trivially schedulable DAG task can be dispatched via *global* and *partitioned* strategies. While global dispatching strategies usually require prioritized queues for ready jobs, we show that prioritization is not needed when dispatching a trivially schedulable DAG task because a ready job is guaranteed to have an idle processor available for execution. For partitioned dispatching strategies, the paths covering a DAG can simply be assigned to processors in the order of the precedence constraints.

To test whether a DAG task is schedulable, we then focus on the problem of adding appropriate edges to convert it into a trivially schedulable DAG task without violating its original constraints. To this end, we propose the *Edge Generation Scheduling* (EGS) framework that attempts to make a DAG task trivially schedulable by iteratively adding appropriately chosen edges. If EGS succeeds in reducing the width to the number of processors while maintaining the length less than or equal to the deadline, the original DAG task is guaranteed to be schedulable.

The EGS framework shifts the complexity of solving the DAG scheduling problem to the problem of selecting the *best* edges to add to a DAG to make it trivially schedulable. We exploit topological and temporal graph properties to limit the search space for the edges to add and propose a deep reinforcement learning (DRL) approach to learn an edge generation policy. In particular, we use the DRL algorithm Proximal Policy Optimization (PPO) [11] and the graph

- Binqi Sun, Mirco Theile, Ziyuan Qin, Daniele Bernardini, Andrea Bastoni, and Marco Caccamo are with TUM School of Engineering and Design, Technical University of Munich, 85748 Munich, Germany. E-mail: {binqi.sun, mirco.theile, ziyuan.qin, danielle.bernardini, andrea.bastoni, mcaccamo}@tum.de.
- Mirco Theile is also with the Department of Electrical Engineering and Computer Sciences, University of California Berkeley, Berkeley, CA 94720, USA.
- Debayan Roy was formerly with TUM School of Engineering and Design, Technical University of Munich, 85748 Munich, Germany. E-mail: debayan.roy.tum@gmail.com

Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

representation neural network architecture Graphormer [12] that is well suited for solving this class of problems.

Combining the proposed EGS framework and the edge generation policy learned by the developed DRL, we derive a concrete DAG scheduling algorithm called EGS-PPO and evaluate it against state-of-the-art DAG scheduling heuristics. Our results show that EGS-PPO consistently outperforms the other approaches by requiring fewer processors to schedule the same DAGs. Additionally, EGS with a random edge generation policy can achieve results similar to the state-of-the-art, highlighting the significance of the EGS framework. We also compare against an optimal mixed-integer linear programming (MILP) baseline for small DAG tasks. EGS-PPO outperforms the other approaches, achieving three to five times smaller optimality gaps.

In summary, in this paper, we:

- 1) Present a new schedulability test (trivial schedulability) for DAG tasks based on observations from the graph domain;
- 2) Propose a novel DAG scheduling framework (EGS) that minimizes processor usage by iteratively generating edges;
- 3) Formulate the edge generation problem as a Markov decision process (MDP) and develop a deep reinforcement learning (DRL) agent to learn an effective edge generation policy for EGS;
- 4) Evaluate the effectiveness of the proposed EGS framework and DRL algorithm by comparing with exact solutions and state-of-the-art DAG scheduling algorithms through extensive experiments on synthetic DAG tasks.

The remainder of the paper is organized as follows. Section 2 reviews the literature on DAG scheduling, and Section 3 describes the system model and introduces the employed concepts of graph theory. Section 4 and Section 5 present our schedulability test and the EGS scheduling framework. A DRL algorithm is developed in Section 6 to learn an efficient edge generation policy for EGS. Section 7 discusses our experimental evaluation, and Section 8 presents future research directions and conclusions.

2 RELATED WORK

2.1 Real-time DAG scheduling

The periodic computation in many cyber-physical systems (CPS) domains, such as automotive, avionics, and manufacturing, is often modeled as a DAG task [2], [13]. Many applications in these domains are part of safety-critical control loops (*e.g.*, brake, speed, and steering control) and hence, have stringent timing requirements (*i.e.*, they are required to meet their deadlines [14], [15]). These requirements led to a body of work performing timing and scheduling analyses of a variety of DAG-based software models, ranging from a single DAG modeling one task [16]–[21], DAGs for multiple tasks with different periods [22]–[27], and more recently, to conditional DAGs [28]–[30], heterogeneous DAGs [31]–[34], and DAGs with mutually exclusive vertices [35]. Given the techniques proposed in [3] to reduce a DAG task set to a single DAG, this work focuses on single DAGs for establishing a new scheduling strategy.

We note that our method can be trivially extended to the scheduling of multiple DAGs with federated scheduling architectures following the approach in [24]. For brevity, the following focuses on real-time DAG scheduling closely related to this work. We refer the readers to [5]–[7] for a comprehensive survey.

The real-time DAG scheduling literature has mainly performed analysis to (i) derive schedulability tests, (ii) bound the response times, and (iii) put forward scheduling strategies to improve schedulability. Baruah et al. [16] first proposed a schedulability test for a single DAG task with constrained deadlines and the global earliest deadline first (EDF) scheduling policy. The test is mainly based on the task’s deadline and period, the length of the longest task chain, and the volume (*i.e.*, the sum of the WCETs) of the DAG. Later, Bonifaci et al. [22] extended the test to a global deadline monotonic (DM) scheduling policy, arbitrary deadlines, and a set of DAG tasks. Furthermore, Baruah et al. [23] improved the schedulability test for constrained deadlines by exploiting the concept of work functions.

One of the earliest works in the DAG response time analysis provided a bound—popularly known as Graham’s bound—for the response time of a task based on the longest path in and the volume of the DAG [17], which is valid for any work-conserving scheduling policy on homogeneous multicore platforms. Recently, He et al. [21] demonstrated the pessimism in Graham’s bound and proposed a tighter bound considering multiple long paths instead of the longest one. Melani et al. [29] also extended Graham’s bound to systems with multiple DAG tasks by considering inter-task interference. Global earliest-deadline-first (EDF) and fixed-priority (FP) scheduling policies were studied in that work. Further, in [25], the bounds were made tighter for two-level FP scheduling, where a DM scheme was followed at the task level, while subtasks were assigned priorities based on the topological order. In recent years, He et al. [18] proposed prioritizing subtasks in the longest paths to reduce the response time and improve schedulability, while at the task level, they still applied DM. Later, Zhao et al. [19] improved the priority assignment strategy at the subtask level by considering dependencies between subtasks and parallelization opportunities. Different from the above approaches, He et al. [20] relaxed the constraint that priority assignment must comply with the topological order of the DAG and proposed a new priority assignment policy, leading to smaller response time bounds.

Note that some of the works above performed analyses for given scheduling policies [16], [17], [21]–[23], [29], while others propose techniques to determine schedule configuration (*e.g.*, priorities) to improve the schedulability [18]–[20]. Our work follows the latter direction, *i.e.*, we determine a static ordering of sub-tasks that will make the task schedulable. In our experiments (Section 7), we show that our proposed approach outperforms the most recent works [18]–[20] in generating feasible schedules for DAG tasks.

2.2 DRL for DAG scheduling

Deep reinforcement learning (DRL) has been applied to various combinatorial optimization problems, including scheduling tasks. In recent years, several studies have applied DRL to DAG scheduling. Mao et al. [36] proposed a

DRL-based DAG scheduler for scheduling data processing jobs in the Spark cluster. Their model takes the cluster’s state information as input and learns to select the next DAG node to be executed via a graph convolution network (GCN) and a policy gradient method. Sun et al. [37] proposed a DRL approach to solve a coflow scheduling problem in distributed computing. It also uses a graph neural network (GNN) in combination with a policy gradient method. However, different from our work, it learns to schedule the edges of a DAG job representing communication stages (*i.e.*, coflows) instead of the DAG nodes representing computation stages.

More recently, Lee et al. [38] proposed a DRL-based DAG task scheduler, which employs a GCN to process a complex interdependent task structure and minimize the makespan of a DAG task. The scheduler assigns priorities to each sub-task to be used in list scheduling. Similar to [38], Joen et al. [39] developed a learning-based scheduler to assign priorities for list scheduling. The difference is that they proposed a one-shot neural network encoder to sample priorities instead of using an episodic reinforcement learning approach. In contrast to these works, our proposed method remains in the graph domain and adds edges to make the DAG task trivially schedulable. Since the code or implementation details of [38] and [39] have not been released, we cannot compare our method with theirs in the experimental evaluation.

3 SYSTEM MODEL AND PRELIMINARIES

3.1 Task model

We consider a DAG task running on M identical processors. The DAG task is characterized by $(\mathcal{G}, D \leq T)$, where \mathcal{G} is a graph defining the set of sub-tasks, T denotes the task period defined as the inter-arrival time of two consecutive jobs (*i.e.*, task instances), and D denotes the task deadline by which all the active sub-jobs must finish their execution. Without loss of generality, we consider constrained deadline, which means the deadline is smaller or equal to the task period (*i.e.*, $D \leq T$). The task graph \mathcal{G} is defined by $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = (v_i)$ is a set of n nodes representing n sub-tasks, and $\mathcal{E} = (e_{ij})$ is a set of directed edges representing the precedence constraints between the sub-tasks. Each sub-task v_i is a non-preemptive sequential computing workload, and its worst-case execution time (WCET) is denoted as C_i . For any two nodes v_i and v_j connected by a directed edge e_{ij} , v_j can start execution only if v_i has finished its execution. Node v_i is called a *predecessor* of v_j , and v_j is a *successor* of v_i . The predecessors and successors of a node v_i are formally defined as $pre(v_i) = \{v_j \in \mathcal{V} | e_{ji} \in \mathcal{E}\}$ and $suc(v_i) = \{v_j \in \mathcal{V} | e_{ij} \in \mathcal{E}\}$, respectively. Moreover, the nodes that are either directly or transitively predecessors (*resp.*, successors) of node v_i are defined as the *ancestors* (*resp.*, *descendants*) of node v_i , denoted by $anc(v_i)$ (*resp.*, $des(v_i)$). Furthermore, a node with no ancestor (*resp.*, descendant) is referred to as the *source* (*resp.*, *sink*) node of the DAG. Without loss of generality, we assume only one source node and one sink node exist in a DAG. A DAG with multiple source (sink) nodes can be easily supported by adding dummy nodes with zero WCET.

Example 1. Consider a DAG task (\mathcal{G}, D) consisting of 7 nodes and 8 edges. The DAG \mathcal{G} is shown in Fig. 1. The number

below each node denotes its WCET. The task deadline is set as $D = 8$. Take node v_6 as an example, the predecessors and successors of node v_6 are $pre(v_6) = \{v_2, v_3, v_4\}$ and $suc(v_6) = \{v_7\}$; the ancestors and descendants of node v_6 are $anc(v_6) = \{v_1, v_2, v_3, v_4\}$ and $des(v_6) = \{v_7\}$, respectively.

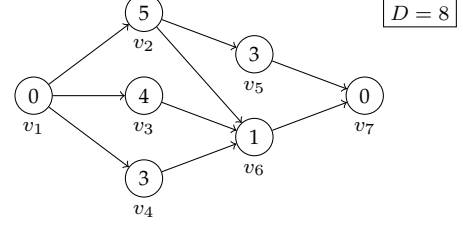


Fig. 1: Example of a DAG task.

3.2 Scheduling model

A DAG task is considered *schedulable* if all the sub-tasks can finish their execution no later than the deadline. At runtime, a sub-job is *ready* once the job is released and its predecessors have finished their execution. We consider two different strategies for dispatching ready sub-jobs to processors: (i) a *global* strategy, where sub-jobs are dynamically dispatched on the available processors and (ii) a *partitioned* strategy, where the assignment of nodes to processors is predetermined offline.

The global dispatching strategy can be implemented by maintaining a prioritized queue to store the ready sub-jobs waiting for execution. Once a sub-job is ready, it goes into the waiting queue, and when a processor becomes idle, the highest priority sub-job in the queue is assigned to the processor for execution. Note that since the node-to-processor mapping is not fixed in the global dispatching strategy, different sub-jobs of the same node can execute on different processors. A partitioned strategy can be implemented by maintaining a list for each processor to store the nodes to be executed and their relative execution order.

3.3 Boolean algebra in graph theory

In graph theory, Boolean matrices are widely-used to represent graph structures. Thus, Boolean algebra can be applied. Here, we introduce some basic Boolean matrix operators and show how they are used in graph operations.

3.3.1 Adjacency matrix and transitive closure

An *adjacency matrix* $\mathbf{A} \in \mathbb{B}^{n \times n}$ is a binary square matrix used to represent the connectivity relations in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $[\mathbf{A}]_{ij} = 1$ if and only if there exists an edge $e_{ij} \in \mathcal{E}$ between node v_i and node v_j . The *transitive closure* of a DAG is defined to represent the reachability relation between the nodes. It can be represented as a binary matrix $\mathbf{T}_c \in \mathbb{B}^{n \times n}$, where $[\mathbf{T}_c]_{ij} = 1$ if and only if node v_i is an ancestor of node v_j .

The adjacency matrix of the transitive closure can be calculated based on the adjacency matrix of the original graph by the Floyd-Warshall algorithm [40], which requires a time complexity of $\mathcal{O}(n^3)$.

3.3.2 Matrix multiplication

We introduce two matrix multiplication methods in Boolean algebra: *Boolean matrix multiplication* and *max-plus matrix multiplication*. The matrix multiplication is used to determine ancestors and dependents of nodes, while the max-plus matrix multiplication is used to compute execution time bounds.

Given two Boolean matrices $\mathbf{A}, \mathbf{B} \in \mathbb{B}^{n \times n}$, the Boolean matrix multiplication $\mathbb{B}^{n \times n} \times \mathbb{B}^{n \times n} \mapsto \mathbb{B}^{n \times n}$ is defined as:

$$[\mathbf{AB}]_{ij} = \bigvee_{k=1}^n ([\mathbf{A}]_{ik} \wedge [\mathbf{B}]_{kj}), \quad \forall i, j = 1, \dots, n \quad (1)$$

where $[\mathbf{AB}]_{ij}$ denotes the (i, j) -th element in the Boolean matrix multiplication product \mathbf{AB} ; \vee and \wedge denote the *or* and *and* operators, respectively.

Given a Boolean matrix $\mathbf{A} \in \mathbb{B}^{n \times n}$ and a real vector $\mathbf{b} \in \mathbb{R}^n$, the max-plus matrix multiplication $\mathbb{B}^{n \times n} \times \mathbb{R}^n \mapsto \mathbb{R}^n$ is defined as:

$$[\mathbf{A} \otimes \mathbf{b}]_i = \max_{k=1, \dots, n} ([\mathbf{A}]_{ik} \cdot [\mathbf{b}]_k), \quad \forall i = 1, \dots, n \quad (2)$$

where $[\mathbf{A} \otimes \mathbf{b}]_i$ is the i -th element in the max-plus matrix multiplication product $\mathbf{A} \otimes \mathbf{b}$.

3.4 Length of a DAG task

3.4.1 Path and DAG length

The length of a DAG is a lower bound on the total execution time of the DAG task. It is later used as a core element of the EGS scheduler. Formally, it is defined through the length of the critical path. A *path* $p = \{v_{p_1}, \dots, v_{p_m}\}$ is a sequence of nodes that are connected by a sequence of edges in the same direction (i.e., $e_{p_k p_{k+1}} \in \mathcal{E}, \forall k = 1, 2, \dots, m-1$). The length of path p is defined as the sum WCET of the nodes included in the path: $L(p) = \sum_{k=1}^m C_{p_k}$. A *complete path* of a DAG is a path that includes the source node and sink node of the DAG. The longest complete path is defined as the *critical path* p^* , and the length of the critical path is defined as the length of the DAG. More formally, we have:

Definition 3.1 (DAG length). The length of a DAG task (\mathcal{G}, D) equals the length of the longest path in the DAG. $L(\mathcal{G}) = \max_{p \in P} L(p)$, where P is the set of paths in \mathcal{G} .

The length of a DAG can be computed within time complexity $\mathcal{O}(n^2)$.

3.4.2 Node-level timing attributes

For each node, we define four timing attributes related to the DAG length: earliest starting time (EST), earliest finishing time (EFT), latest starting time (LST), and latest finishing time (LFT). The EST means the earliest time a node can start its execution, which equals the maximum of its predecessors' EFT. Similarly, the LFT represents the latest time a node can finish its execution while meeting the deadline, i.e., the minimum of its successors' LST. These timing attributes are used to restrict the action space of the reinforcement

learning agent, and are part of the node features to aid the agent's learning. They are defined through:

$$\begin{aligned} \mathbf{t}^{\text{EFT}} &= \mathbf{t}^{\text{EST}} + \mathbf{C} \\ \mathbf{t}^{\text{EST}} &= \mathbf{A} \otimes \mathbf{t}^{\text{EFT}} \\ \mathbf{t}^{\text{LST}} &= \mathbf{t}^{\text{LFT}} - \mathbf{C} \\ \mathbf{t}^{\text{LFT}} &= \mathbf{A}^T \otimes \mathbf{t}^{\text{LST}} \end{aligned} \quad (3)$$

where $\mathbf{t}^{\text{EST}}, \mathbf{t}^{\text{EFT}}, \mathbf{t}^{\text{LST}}, \mathbf{t}^{\text{LFT}} \in \mathbb{R}^n$ are vectors denoting the EST, EFT, LST, and LFT of the nodes, respectively; $\mathbf{C} \in \mathbb{R}^n$ is a vector denoting the WCET of each node.

Equations (3) can be solved by fixed-point iteration. First, we initialize $t_i^{\text{EST}} = 0, t_i^{\text{LFT}} = D, \forall i = 1, \dots, n$. Then, at each iteration $k = 1, \dots, n$, we update the values of each timing attribute according to (3) until they converge (i.e., no value is updated from iteration k to iteration $k+1$). It can be guaranteed that the values will converge within n iterations since the critical path of the DAG is composed of at most n nodes [3]. Thus, the time complexity of computing the node-level timing attributes is $\mathcal{O}(n^3)$.

Example 2. The critical path of the DAG in Example 1 is $\{v_1, v_2, v_5, v_7\}$, and the length of the DAG is $C_1 + C_2 + C_5 + C_7 = 8$. The EFT and LST of each node are shown in Fig. 2.

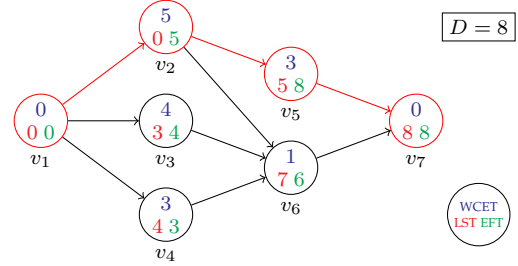


Fig. 2: Example of DAG length and node-level timing attributes. The nodes and edges of the critical path are marked in red. The numbers inside each node represent the node's WCET, LST, and EFT with corresponding colors.

3.5 Width of a DAG task

3.5.1 Antichain and DAG width

The width of a DAG task indicates the maximum number of nodes that can be run in parallel. It can be computed using the critical antichain of the DAG. An antichain $q = \{v_{q_1}, \dots, v_{q_m}\}$ in a DAG is a set of nodes that are pair-wise non-reachable (i.e., $v_i \notin \text{anc}(v_j), \forall i, j \in \{q_1, \dots, q_m\}$). The size of an antichain is defined as the number of nodes in the antichain. The maximum-size antichain is called the *critical antichain*, and the size of the critical antichain is defined as the width of the DAG. By Dilworth's theorem [41], the width of \mathcal{G} also equals the minimum number of paths needed to cover all the nodes of DAG \mathcal{G} .

Definition 3.2 (DAG width). The width of a DAG task (\mathcal{G}, D) is defined as the size of the maximum-size antichain in DAG \mathcal{G} . It is equivalent to the minimum number of paths needed to cover all the nodes in DAG \mathcal{G} .

There have been several methods proposed for DAG width computation in the literature. The most well-known

ones are minimum path cover algorithms, where the problem is reduced to either the *maximum matching* in bipartite graphs with time complexity $\mathcal{O}(\sqrt{nm}^*)$ [42] or the *minimum flows* in directed graphs with time complexity $\mathcal{O}(nm)$ [43], where m and m^* denote the number of edges in graph \mathcal{G} and the transitive closure of graph \mathcal{G} , respectively.

3.5.2 Node-level parallelism attributes

For each node $v_i \in V$, we define three attributes related to the DAG width: lateral width (LW), in-width (IW), and out-width (OW). The LW of node v_i means the maximum number of pair-wise non-reachable nodes with which node v_i can run in parallel. It equals the width of the DAG derived by removing node v_i and all its ancestors and descendants from DAG \mathcal{G} . The IW (*resp.*, OW) of node v_i denotes the maximum number of pair-wise non-reachable nodes among the ancestors (*resp.*, descendants) of node v_i and the nodes that are parallel to v_i . As with the node-level timing attributes, the parallelism attributes are used to restrict action space and are part of the node features intended to improve the agent's understanding of the DAG. Similar to the LW, the IW (*resp.*, OW) of node v_i can be calculated as the width of the DAG after removing node v_i and all its descendants (*resp.*, ancestors) from DAG \mathcal{G} :

$$\begin{aligned} m_i^{\text{LW}} &= W(\mathcal{G} \setminus \mathcal{V}'), \mathcal{V}' = \text{anc}(v_i) \cup \text{des}(v_i) \cup \{v_i\} \\ m_i^{\text{IW}} &= W(\mathcal{G} \setminus \mathcal{V}'), \mathcal{V}' = \text{des}(v_i) \cup \{v_i\} \\ m_i^{\text{OW}} &= W(\mathcal{G} \setminus \mathcal{V}'), \mathcal{V}' = \text{anc}(v_i) \cup \{v_i\}, \forall i = 1, \dots, n \end{aligned} \quad (4)$$

where m_i^{LW} , m_i^{IW} , and m_i^{OW} denote the LW, IW, and OW of node v_i , respectively; $\mathcal{G} \setminus \mathcal{V}'$ denotes the graph with the nodes in node set \mathcal{V}' and all their connected edges removed from graph \mathcal{G} . The time complexity of computing each node-level parallelism attribute is the same as the complexity of computing the graph width.

Example 3. The critical antichains of the DAG in Example 1 are $\{v_2, v_3, v_4\}$ and $\{v_3, v_4, v_5\}$, and the DAG width is 3. The LW, IW and OW of each node are illustrated in Fig. 3.

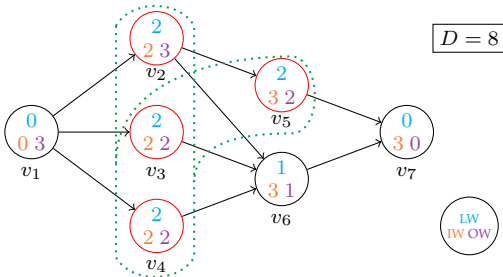


Fig. 3: Example of DAG width and node-level parallelism attributes. The nodes belonging to the critical antichains are marked red, and a dotted line surrounds each antichain. The numbers inside each node represent the node's LW, IW, and OW with corresponding colors.

4 SCHEDULABILITY TEST

We present an exact schedulability test for a DAG task based on the concept of *trivial schedulability* defined by the length and width of the DAG:

Definition 4.1 (Trivial schedulability). A DAG task (\mathcal{G}, D) is *trivially schedulable* on M processors if it satisfies the following two conditions:

- 1) the length of \mathcal{G} is no larger than the task deadline: $L(\mathcal{G}) \leq D$;
- 2) the width of \mathcal{G} is no larger than the number of processors: $W(\mathcal{G}) \leq M$.

Based on Definition 4.1, we can derive several important properties of a trivially schedulable DAG task, which are later used to prove our schedulability test.

First, we show that a trivially schedulable DAG is guaranteed to be schedulable under a global dispatching strategy by formulating the following Lemmas 4.1 - 4.3.

Lemma 4.1. *If a DAG task (\mathcal{G}, D) is trivially schedulable on M processors, then at most M sub-jobs are active at the same time.*

Proof. We prove the lemma by contradiction. Suppose there are $M + 1$ active sub-jobs at the same time. Since two sub-jobs can be active at the same time only if they do not have precedence constraints, there must be $M + 1$ nodes that are pair-wise non-reachable. Thus, they constitute an antichain of size $M + 1$. By Definition 3.2, the width of \mathcal{G} is thus at least $M + 1$, which contradicts the width constraint $W(\mathcal{G}) \leq M$ in Definition 4.1. \square

Lemma 4.2. *If a DAG task (\mathcal{G}, D) is trivially schedulable on M processors, each sub-job $v_i \in \mathcal{V}$ can start execution at its ready time using any global work-conserving dispatching strategies.*

Proof. We prove the lemma by contradiction. Suppose a sub-job of node v_i cannot start its execution at its ready time r_i . Since we consider a global work-conserving dispatching strategy, all M processors must be busy executing other sub-jobs at time r_i . Thus, we know that at least $M + 1$ sub-jobs (including v_i) are active at time r_i , which contradicts Lemma 4.1. \square

By Lemma 4.2, we know that a trivially schedulable DAG task will not have any ready sub-jobs waiting for processors to become idle. Therefore, it is not necessary to use a prioritized queue under a global dispatching strategy if the DAG task is trivially schedulable.

Additionally, Lemma 4.2 allows us to derive the schedulability of a trivially schedulable DAG task under a global work-conserving dispatching strategy:

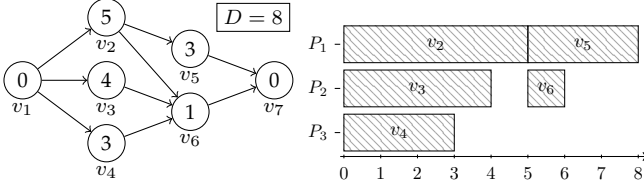
Lemma 4.3. *If a DAG task (\mathcal{G}, D) is trivially schedulable on M processors, then it is schedulable on M processors using any global work-conserving dispatching strategies.*

Proof. By Lemma 4.2, we know each sub-job can find at least an idle processor to start its execution at its ready time. Thus, the response time of the sink node is given by the length of the critical path in \mathcal{G} . By Definition 4.1, we know $L(\mathcal{G}) \leq D$. Therefore, the sink node finishes its execution no later than the task deadline. Since the sink node is a descendant of all other nodes, all the nodes finish their execution by the task deadline. \square

Second, we show that a trivially schedulable DAG task is also schedulable using the partitioned dispatching strategy in Algorithm 1.

Algorithm 1: Partitioned dispatching strategy

-
- Input:** (\mathcal{G}, D) : a trivially schedulable DAG task on M processors;
- Output:** a partitioned schedule;
- 1 Split \mathcal{G} into M paths $\{p_1, \dots, p_M\}$ using a minimum path cover algorithm (e.g., [42]);
 - 2 **for** $k \leftarrow 1$ to M **do**
 - 3 Assign the nodes in p_k to processor k and specify their execution order according to the precedence constraints in \mathcal{G} ;
-

Fig. 4: Example of the trivial schedule with $M = 3$.

Lemma 4.4. *If a DAG task (\mathcal{G}, D) is trivially schedulable on M processors, then it is schedulable on M processors using the partitioned dispatching strategy in Algorithm 1.*

Proof. By Definition 4.1, since (\mathcal{G}, D) is trivially schedulable on M processors, we know $L(\mathcal{G}) \leq D$ and $W(\mathcal{G}) \leq M$. Since $W(\mathcal{G}) \leq M$, there exist M paths $\{p_1, \dots, p_M\}$ that can cover all the nodes in \mathcal{G} (line 1, Algorithm 1). Since $L(\mathcal{G}) \leq D$, the length of each path in $\{p_1, \dots, p_M\}$ is smaller than or equal to $L(\mathcal{G}) \leq D$. Therefore, we can construct a feasible schedule of task (\mathcal{G}, D) by assigning each path in $\{p_1, \dots, p_M\}$ to a unique processor, where the execution order is determined according to the precedence constraints (lines 2-3, Algorithm 1). \square

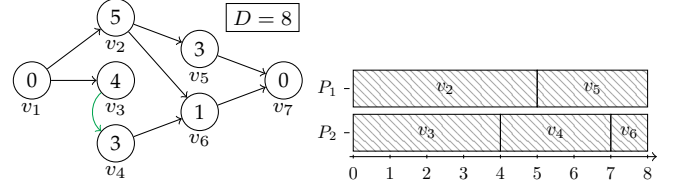
Lemma 4.4 not only proves the schedulability of a trivially schedulable DAG task, but also provides a way to generate a feasible assignment of nodes to processors that can be used by the partitioned dispatching strategy. We use the following example to show the generation process.

Example 4. *The trivial schedule of the DAG task in Fig. 4 (left) on $M = 3$ processors is illustrated in Fig. 4 (right). The DAG is split into 3 paths: $p_1 = \{v_1, v_2, v_5, v_7\}$, $p_2 = \{v_3, v_6\}$, and $p_3 = \{v_4\}$, which are mapped to P_1, P_2 , and P_3 , respectively.*

Now, we use Lemma 4.3 and 4.4 to derive the following exact schedulability test that can be used in conjunction with global and partitioned dispatching strategies.

Theorem 4.5 (Schedulability test). *A DAG task (\mathcal{G}, D) is schedulable on M processors if and only if there exists a trivially schedulable DAG task (\mathcal{G}', D) , where $\mathcal{G}' \supseteq_E \mathcal{G}$ (i.e., $\mathcal{V}' = \mathcal{V}$ and $\mathcal{E}' \supseteq \mathcal{E}$ in the transitive closures of \mathcal{G} and \mathcal{G}').*

Proof. **Sufficiency.** Suppose we have a graph $\mathcal{G}' \supseteq_E \mathcal{G}$, and task (\mathcal{G}', D) is trivially schedulable on M processors. By Lemma 4.3 (resp., Lemma 4.4), we know that task (\mathcal{G}', D) is schedulable using a global (resp., partitioned) dispatching strategy. Since task (\mathcal{G}', D) has the same nodes and deadline with task (\mathcal{G}, D) , and all the precedence constraints in \mathcal{G}

Fig. 5: Example of the trivial schedule with $M = 2$.

are included in \mathcal{G}' (i.e., $\mathcal{E}' \supseteq \mathcal{E}$), a feasible schedule of task (\mathcal{G}', D) is also a feasible schedule of task (\mathcal{G}, D) . Thus, task (\mathcal{G}, D) is schedulable on M processors.

Necessity. We prove the necessity by showing that a trivially schedulable DAG task $(\mathcal{G}' \supseteq_E \mathcal{G}, D)$ exists if task (\mathcal{G}, D) is schedulable on M processors. Suppose \mathcal{S} is a static schedule of (\mathcal{G}, D) on M processors. The static schedule \mathcal{S} specifies the processor allocation $p(v_i) \in \{1, \dots, M\}$ and execution starting time $s(v_i), \forall v_i \in \mathcal{V}$. Using the set of edges describing the precedence per processor k as

$$\hat{\mathcal{E}}_k = \{e_{ij}, \forall i, j \mid p(v_i) = p(v_j) = k \wedge s(v_i) \leq s(v_j)\}$$

and $\hat{\mathcal{E}} = \hat{\mathcal{E}}_1 \cup \dots \cup \hat{\mathcal{E}}_M$, we can construct the graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E} \cup \hat{\mathcal{E}})$. By construction, the nodes in \mathcal{G}' can be covered by M paths, each of which contains the nodes assigned to a processor k , whose precedence is described by $\hat{\mathcal{E}}_k$, yielding $W(\mathcal{G}') \leq M$. In addition, since \mathcal{S} respects all precedence constraints in \mathcal{E} and $\hat{\mathcal{E}}$, it is a feasible schedule of task (\mathcal{G}', D) , thus $L(\mathcal{G}') \leq D$. Since $W(\mathcal{G}') \leq M$ and $L(\mathcal{G}') \leq D$, $\mathcal{G}' \supseteq_E \mathcal{G}$ is trivially schedulable on M processors. \square

From Theorem 4.5, we know that a feasible schedule of task (\mathcal{G}, D) on fewer processors than the current DAG width can be obtained by generating a graph \mathcal{G}' that has additional edges to \mathcal{G} and satisfies both the length and width constraints. Example 5 shows a DAG generated by adding edge $e_{3,4}$ to the DAG task in Example 4. As illustrated in Fig. 5, the resulting DAG is trivially schedulable on $M = 2$ processors (compared to $M = 3$ in Example 4). Based on this observation, we propose a new DAG scheduling algorithm called Edge Generation Scheduling (EGS) to minimize processor usage in Section 5.

Example 5. *The trivial schedule of the DAG task in Fig. 5 (left) on $M = 2$ processors is illustrated in Fig. 5 (right). The DAG is split into 2 paths: $p_1 = \{v_1, v_2, v_5, v_7\}$ and $p_2 = \{v_3, v_4, v_6\}$, which are mapped to P_1 and P_2 , respectively.*

5 EDGE GENERATION SCHEDULING

We propose the edge generation scheduling (EGS) framework based on trivial schedulability. Consider a common DAG scheduling problem: is a given DAG task (\mathcal{G}, D) schedulable on M processors? For EGS, we reformulate the question: what is the minimum number of processors needed to schedule the DAG task (\mathcal{G}, D) ? Using trivial schedulability, the question forms the optimization problem:

$$\begin{aligned} & \underset{\mathcal{G}' \supseteq_E \mathcal{G}}{\text{minimize}} && W(\mathcal{G}') && (5) \\ & \text{subject to} && L(\mathcal{G}') \leq D \end{aligned}$$

i.e., finding a graph $\mathcal{G}' \supseteq_E \mathcal{G}$ that has minimal width while maintaining the length constraint. We aim to find this \mathcal{G}' by iteratively adding edges until no edges can be added without violating the length constraint or until a lower bound on the width is reached.

The rest of the section describes which edges can be added in Section 5.1 and how to compute the lower bound in Section 5.2. Section 5.3 shows the algorithm and complexity of EGS, and Section 5.4 shows an example to highlight the challenge in the edge selection choice.

5.1 Eligible edges

We define edge masks as Boolean matrices, where each entry is active if the corresponding edge can be added to the current DAG and inactive otherwise. Four different edge masks are developed for different purposes: *redundancy mask*, *cycle mask*, *length mask*, and *width mask*.

- **Redundancy mask \mathbf{M}_r .** The redundancy mask avoids adding redundant edges already existing in the current graph. The redundancy mask can be calculated as the reverse of the graph's transitive closure:

$$\mathbf{M}_r = \neg \mathbf{T}_c \quad (6)$$

- **Cycle mask \mathbf{M}_c .** The cycle mask avoids introducing cycles when adding edges to the graph. It can be computed as

$$\mathbf{M}_c = \neg((\mathbf{T}_c)^T \vee \mathbf{I}), \quad (7)$$

where $(\mathbf{T}_c)^T$ is the transpose of the transitive closure and \mathbf{I} denotes the Boolean identity matrix.

- **Length mask \mathbf{M}_l .** The length mask avoids adding edges that would violate the length constraint. The length mask can be derived by comparing the EFT of the predecessor node v_i and the LST of the successor node v_j of the candidate edge e_{ij} :

$$[\mathbf{M}_l]_{ij} = \mathbb{1}_{t_i^{\text{EFT}} \leq t_j^{\text{LST}}}, \quad \forall i, j = 1, \dots, n \quad (8)$$

where $\mathbb{1}_{t_i^{\text{EFT}} \leq t_j^{\text{LST}}}$ is an indicator function, returning 1 if $t_i^{\text{EFT}} \leq t_j^{\text{LST}}$ and 0 otherwise. The correctness of the length mask is proved by Theorem 5.1.

- **Width mask \mathbf{M}_w .** The width mask restricts edges to be generated between the nodes with the largest lateral width, which is a necessary condition to reduce DAG width as shown in Theorem 5.2.

$$[\mathbf{M}_w]_{ij} = \mathbb{1}_{m_i^{\text{LW}} = m_j^{\text{LW}} = W(\mathcal{G}) - 1}, \quad \forall i, j = 1, \dots, n, \quad (9)$$

where m_i^{LW} denotes the lateral width of node v_i .

Theorem 5.1 (Length constraint). *Given a DAG task (\mathcal{G}, D) with $L(\mathcal{G}) \leq D$ and DAG $\mathcal{G}' = (\mathcal{V}, \mathcal{E} \cup \{e_{ij}\})$, with $v_j \notin \text{des}(v_i)$ (redundant) and $v_j \notin \text{anc}(v_i) \cup \{v_i\}$ (cycle), $L(\mathcal{G}') \leq D$ if and only if $t_i^{\text{EFT}} \leq t_j^{\text{LST}}$ in task (\mathcal{G}, D) .*

Proof. Adding an edge e_{ij} to \mathcal{G} to create \mathcal{G}' does not change t_i^{EFT} and t_j^{LST} as it does not alter the ancestors and descendants of v_i and v_j , respectively. By definition, t_i^{EFT} equals the length of the longest path between the source node and v_i (denoted by p_i^l , $L(p_i^l) = t_i^{\text{EFT}}$). Conversely, t_j^{LST} equals the difference of D minus the length of the longest path between v_j and the sink node (denoted by p_j^r , $L(p_j^r) = D - t_j^{\text{LST}}$). Since

e_{ij} connects v_i and v_j in \mathcal{G}' , there exists a path $\hat{p}_{ij} = p_i^l \cup p_j^r$ with $L(\hat{p}_{ij}) = t_i^{\text{EFT}} + D - t_j^{\text{LST}}$, which is the longest path among all the paths that go through v_i and v_j . Therefore,

$$L(\mathcal{G}') \geq L(\hat{p}_{ij}) = t_i^{\text{EFT}} + D - t_j^{\text{LST}}.$$

The inequality is tight when \hat{p}_{ij} is a critical path of \mathcal{G}' .

Sufficiency. Denote \mathcal{P} as set of paths of \mathcal{G} and \mathcal{P}^* as the set of critical paths of \mathcal{G}' . There are two cases to consider:

- $\hat{p}_{ij} \in \mathcal{P}^*$. The above inequality is tight, i.e., $L(\mathcal{G}') = t_i^{\text{EFT}} + D - t_j^{\text{LST}}$. It follows $t_i^{\text{EFT}} \leq t_j^{\text{LST}} \implies L(\mathcal{G}') \leq D$.
- $\hat{p}_{ij} \notin \mathcal{P}^*$. Since \hat{p}_{ij} is the longest path among all the paths that go through v_i and v_j , it follows that either $v_i \notin p^*$ or $v_j \notin p^*$, $\forall p^* \in \mathcal{P}^*$. Therefore, $\mathcal{P}^* \subseteq \mathcal{P}$, $L(\mathcal{G}') = L(\mathcal{G}) \leq D$.

Necessity. $L(\mathcal{G}') \leq D \wedge L(\mathcal{G}') \geq t_i^{\text{EFT}} + D - t_j^{\text{LST}} \implies t_i^{\text{EFT}} + D - t_j^{\text{LST}} \leq D \implies t_i^{\text{EFT}} \leq t_j^{\text{LST}}$. \square

Theorem 5.2 (Width reduction). *For any DAG $\mathcal{G}' \supseteq_E \mathcal{G}$, $W(\mathcal{G}') < W(\mathcal{G})$ only if the transitive closure of \mathcal{G}' has at least one edge between the nodes with the largest lateral width in \mathcal{G} .*

Proof. We prove the theorem by contradiction. Suppose there exists a DAG $\mathcal{G}' \supseteq_E \mathcal{G}$ whose width is lower than \mathcal{G} and has no edge between the nodes with the largest LW in \mathcal{G} . We denote the critical antichain of DAG \mathcal{G} as q (i.e., $|q| = W(\mathcal{G})$). Since \mathcal{G}' has no edge between the nodes with the largest LW, there is no edge between the nodes belonging to the critical antichain q . Thus, q is an antichain of \mathcal{G}' . By the definition of the DAG width, we know that the width of \mathcal{G}' is equal to or larger than $|q| = W(\mathcal{G})$, which contradicts our assumption. \square

Finally, the complete edge mask is obtained by combining all the above edge masks:

$$\mathbf{M} = \mathbf{M}_r \wedge \mathbf{M}_c \wedge \mathbf{M}_l \wedge \mathbf{M}_w \quad (10)$$

Example 6. *The valid and invalid edges of the DAG task in Example 1 are illustrated in Fig. 6.*

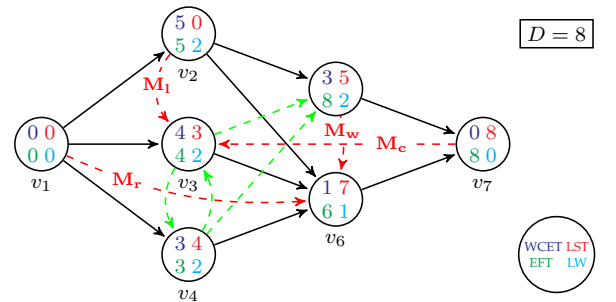


Fig. 6: Example of the valid and invalid edges, shown with the green and red dashed edges, respectively. For ease of presentation, we only show one invalid action masked out by each type of action mask (\mathbf{M}_r , \mathbf{M}_c , \mathbf{M}_l or \mathbf{M}_w). The numbers inside each node represent the node's WCET, LST, EFT, and LW with corresponding colors.

5.2 Lower bound of task parallelism

Theorem 5.3 (Lower bound). *Given a DAG task (\mathcal{G}, D) , a set of sub-tasks $\hat{\mathcal{V}} \subseteq \mathcal{V}$ are not schedulable on fewer than $\text{LB}(\hat{\mathcal{V}})$ processors:*

$$\text{LB}(\hat{\mathcal{V}}) = \left\lceil \frac{\sum_{v_i \in \hat{\mathcal{V}}} C_i}{\max_{v_i \in \hat{\mathcal{V}}} t_i^{\text{LFT}} - \min_{v_j \in \hat{\mathcal{V}}} t_j^{\text{EST}}} \right\rceil \quad (11)$$

Proof. We prove the theorem by contradiction. Assume all the sub-tasks in $\hat{\mathcal{V}}$ are schedulable on $M' \leq \text{LB}(\hat{\mathcal{V}}) - 1$ processors. Then, the maximum response time among all the sub-tasks in $\hat{\mathcal{V}}$ is at least $\sum_{v_i \in \hat{\mathcal{V}}} C_i / M' \geq \sum_{v_i \in \hat{\mathcal{V}}} C_i / (\text{LB}(\hat{\mathcal{V}}) - 1) > \max_{v_i \in \hat{\mathcal{V}}} t_i^{\text{LFT}} - \min_{v_j \in \hat{\mathcal{V}}} t_j^{\text{EST}}$. This implies that some sub-task cannot finish its execution within the required finishing time, which contradicts the assumption that all the sub-tasks in $\hat{\mathcal{V}}$ are schedulable. \square

As $\text{LB}(\hat{\mathcal{V}})$ gives the lower bound of the number of processors required by the subset of task nodes $\hat{\mathcal{V}}$, it is also a valid lower bound of the DAG task's parallelism \underline{M} . By computing the $\text{LB}(\hat{\mathcal{V}})$ of each node subset $\hat{\mathcal{V}}$, we could derive a tighter \underline{M} by taking the maximum $\text{LB}(\hat{\mathcal{V}})$ among all possible node subsets (i.e., $\underline{M} = \max_{\hat{\mathcal{V}}} \text{LB}(\hat{\mathcal{V}})$). However, since the total number of node subsets is exponential with respect to the number of nodes in the DAG, deriving the lower bound of every possible node subset is not computationally tractable. In this paper, we consider two special cases of $\hat{\mathcal{V}}$: (i) all the nodes in the DAG (i.e., $\hat{\mathcal{V}} = \mathcal{V}$), (ii) the nodes with the largest lateral width (i.e., $\hat{\mathcal{V}} = \{v_i \in \mathcal{V} | m_i^{\text{LW}} = W(\mathcal{G}) - 1\}$), and take the larger $\text{LB}(\hat{\mathcal{V}})$ as the lower bound of the task parallelism \underline{M} , as shown in Corollary 5.3.1.

Corollary 5.3.1. *Given a DAG task (\mathcal{G}, D) , the task is not schedulable on fewer than \underline{M} processors:*

$$\underline{M} = \max\{\text{LB}(\mathcal{V}), \text{LB}(\{v_i \in \mathcal{V} | m_i^{\text{LW}} = W(\mathcal{G}) - 1\})\} \quad (12)$$

5.3 EGS framework

Algorithm 2 shows specific procedures of the EGS framework. In line 1, a DAG \mathcal{G}' is initialized as the input graph \mathcal{G} . The width of the DAG and its lower bound are computed accordingly. In line 2, the edge mask of DAG \mathcal{G}' is initialized using Equations (6)-(10). In lines 3-6, edges are added iteratively to DAG \mathcal{G}' until (i) no edges can be added according to the edge mask \mathbf{M} or (ii) the current DAG width reaches its lower bound. In each iteration, an edge is selected according to an edge generation policy π (line 4). The policy takes the current DAG as input and outputs an edge e_{ij} to be added in line 5. The generated edge must comply with the edge mask such that $[\mathbf{M}]_{ij} = 1$. At the end of each iteration, The DAG width and its lower bound are updated in line 6, and the edge mask is updated in line 7.

The time complexity of the proposed EGS framework is analyzed as follows. In each iteration of EGS, the time complexity depends on the complexity of updating the edge masks and selecting the edge to be added to the graph. Recall that we consider four edge masks. Updating each of them requires the update of the transitive closure, which

Algorithm 2: The EGS framework

Input: (\mathcal{G}, D) : the DAG task to be scheduled;
Output: \mathcal{G}' : the DAG with minimized width;

- 1 $\mathcal{G}' \leftarrow \mathcal{G}, \underline{M} \leftarrow \text{LB}(\mathcal{G}')$;
- 2 Compute edge mask \mathbf{M} according to Equation (10);
- 3 **while** $\bigvee_{i=1}^n \bigvee_{j=1}^n [\mathbf{M}]_{ij}$ **and** $\underline{M} < W(\mathcal{G})$ **do**
- 4 $e_{ij} \leftarrow$ Select with policy π and edge mask \mathbf{M} ;
- 5 $\mathcal{G}' \leftarrow \mathcal{G}' \cup \{e_{ij}\}$;
- 6 $\underline{M} \leftarrow \text{LB}(\mathcal{G}')$;
- 7 Update edge mask \mathbf{M} ;
- 8 **return** \mathcal{G}' ;

takes a complexity of $\mathcal{O}(n^2)$. Once we get the updated transitive closure, the redundancy mask and the cycle mask can be updated within $\mathcal{O}(n^2)$. The update of the length and width mask requires recomputing each node's timing and parallelism attributes from scratch, which takes a time complexity of $\mathcal{O}(n^3)$ and $\mathcal{O}(n\sqrt{n}m^*)$, respectively (details see Section 3.3). Thus, the overall complexity of updating the edge masks is bounded by $\mathcal{O}(n^3 + n\sqrt{n} \cdot m^*)$, where m^* denotes the number of edges in the transitive closure. Since in the worst case at most n^2 edges can be added to the original graph, the time complexity of the EGS algorithm is bounded by $\mathcal{O}(n^2 \cdot (n^3 + n\sqrt{n} \cdot m^* + \Omega_\pi))$, where Ω_π is the time complexity of the edge generation policy π .

5.4 Example of edge generation policy

So far, we have not discussed specific edge generation policies that can be used within the proposed EGS framework. Here, we use an example to illustrate two different edge generation policies, which result in two different schedules with different processor usage. This example indicates that the edge generation policy is critical to the scheduling performance of EGS and motivates us to develop a deep reinforcement learning algorithm to learn an efficient edge generation policy in Section 6.

Example 7. *We apply the EGS with two different edge generation policies to the DAG task in Fig. 7a and compare their scheduling results. In the first iteration of the EGS, the two policies select to add edge e_{43} and e_{35} , resulting in the DAGs shown in Fig. 7b and Fig. 7c, respectively.*

6 DEEP REINFORCEMENT LEARNING

Finding an optimal edge generation policy is challenging. Recall that EGS aims to reduce the DAG width by adding one edge in each step. Since the decision made in one step impacts the following decision process, a greedy policy that always selects the edge with the maximum intermediate width reduction may lead to a local optimum. Other simple heuristics also often struggle with the global complexity of NP-hard problems. In this section, we formulate the edge generation as a Markov Decision Process (MDP) that aims to maximize the width reduction. Then, we use the DRL algorithm Proximal Policy Optimization (PPO) [11] to find a policy with good global performance for this NP-hard MDP.

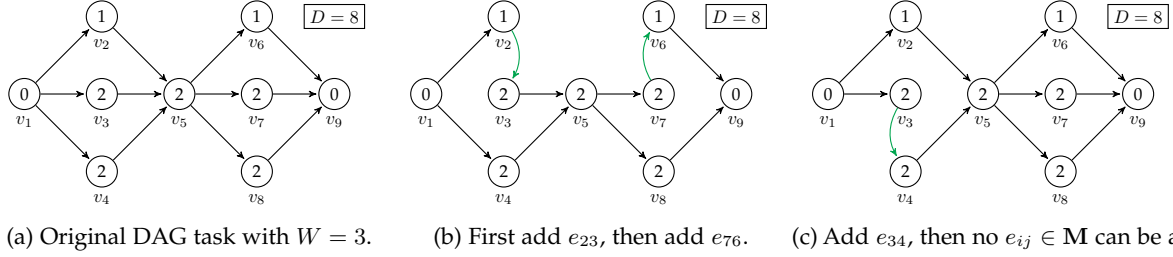


Fig. 7: Example of different edge generation policies. Given a DAG task \$(\mathcal{G}, D)\$ with \$\mathcal{G}\$ illustrated in (a) and \$D = 8\$, the edge generation policy used in (b) reduces the DAG width to 2, while the policy used in (c) cannot reduce the DAG width.

6.1 MDP formulation

An MDP is defined through the tuple \$(\mathcal{S}, \mathcal{A}, \mathbf{R}, \mathbf{T}, \gamma)\$ with the state space \$\mathcal{S}\$ and the action space \$\mathcal{A}\$. A reward function \$\mathbf{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}\$ assigns a scalar reward to state-action pairs. In a deterministic MDP such as used in this work, a transition function \$\mathbf{T} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}\$ determines the next state according to the current state and current action. A discount factor \$\gamma \in [0, 1)\$ balances the importance of immediate and future rewards. The goal in an MDP is to find a policy \$\pi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}^+\$ that assigns a probability to each action given a state. The policy aims to maximize the expected cumulative discounted return

$$\mathcal{R}^\pi(s) = \mathbb{E}_{a \sim \pi(a|s)} \left[\sum_{t=0}^{\infty} \gamma^t \mathbf{R}(s_t, a_t) \mid s_0 = s \right]. \quad (13)$$

In the EGS framework, the state \$s_t \in \mathcal{S}\$ is a DAG \$\mathcal{G}_t = (\mathcal{V}, \mathcal{E}_t)\$ with its constant vertices and increasing set of edges. The initial state is the DAG to be scheduled, *i.e.*, \$\mathcal{G}_0 = \mathcal{G}\$. The action space \$\mathcal{A}\$ is defined as all eligible edges according to the action mask \$\mathbf{M}_t\$, of which the policy can choose an edge \$a_t \in \mathbf{M}_t\$ to add to the graph. Thus, the transition function \$\mathbf{T}\$ results in \$\mathcal{E}_{t+1} = \mathcal{E}_t \cup \{a_t\}\$, and \$\mathbf{M}_{t+1}\$ is updated according to the new edges using (10). In EGS, the goal is to minimize the number of cores needed to schedule the DAG. Therefore, the reward is defined as

$$\mathbf{R}(s_t, a_t) = W(\mathcal{G}_t) - W(\mathcal{G}_{t+1}), \quad (14)$$

giving a reward equal to the width reduction after adding an edge. The MDP terminates if the action mask is empty or the width reaches the lower bound. To aid the agent in solving the MDP we provide it with precomputed node features consisting of (i) the node-level timing attributes, including the WCET, EFT, and LST and (ii) the node-level parallelism attributes, including the LW, IW, and OW.

6.2 PPO algorithm

The PPO algorithm [11] is an on-policy, actor-critic reinforcement learning algorithm that trains a value function \$V_\phi^\pi(s)\$ (known as critic network) that predicts the cumulative return \$\mathcal{R}(s)\$ of a state \$s\$ under the currently active policy \$\pi_\theta(s)\$ (known as actor-network). Conducting a rollout (*i.e.*, a long sequence of state-action-reward tuples), an advantage of the actions in the rollout is approximated using a generalized advantage estimate (GAE) [44]. The advantage indicates whether the action taken was better or worse than the average performance of the current policy. If the action was better (*resp.*, worse) than the expectation, the probability of

taking this action is increased (*resp.*, decreased). We refer the reader to [11] for the details of the PPO algorithm.

6.3 Neural network architecture

We apply an encoder-decoder architecture in the actor and critic network of the PPO algorithm. The encoder learns the node embedding of an input graph, and the decoder uses the node embedding to produce an output depending on its downstream task. Recall that in the PPO algorithm, the actor and critic have the same input but different outputs. Hence, we use the same encoder but different decoder architectures for the two networks. The overall architecture of the actor and critic networks is illustrated in Fig. 8.

The encoder is built upon a recent graph representation network *Graphormer* [12], which achieves state-of-the-art performance in various graph applications. Graphormer consists of multiple encoder layers, each of which includes a multi-head attention (MHA) module and a multi-layer perceptron (MLP) block with Layer normalization (LN) applied before the MHA and the MLP. The MHA is the key component in Graphormer, which effectively encodes the structural graph information via a residual term in the attention module. In this work, since a directed graph is considered, we encode both the forward and backward connectivity using the adjacency matrices of transitive closure \$\mathbf{T}_c\$ and its transpose \$(\mathbf{T}_c)^\top\$, respectively. Concretely, for each attention head, we assign different trainable scalars to each feasible value (*i.e.*, 0 and 1) in \$\mathbf{T}_c\$ and \$(\mathbf{T}_c)^\top\$. Then, the trainable scalars corresponding to the \$(i, j)\$-th node pair will be added to the \$(i, j)\$-th entry of the attention product matrix \$\mathbf{Attn} \in \mathbb{R}^{n \times n}\$:

$$[\mathbf{Attn}^\top]_{ij} = [\mathbf{Attn}]_{ij} + b_1([\mathbf{T}_c]_{ij}) + b_2([\mathbf{T}_c^\top]_{ij}), \forall i, j \quad (15)$$

where \$b_1([\mathbf{T}_c]_{ij})\$ and \$b_2([\mathbf{T}_c^\top]_{ij})\$ denote the trainable scalars corresponding to \$[\mathbf{T}_c]_{ij}\$ and \$[\mathbf{T}_c^\top]_{ij}\$, respectively. Note that the trainable scalars are different in different attention heads, and shared across all encoder layers. For simplicity of presentation, we illustrate the single-head attention in Fig. 8 and Equation (15). The extension to the multi-head attention is standard and straightforward. We note that there are many other design choices of graph representation networks, *e.g.*, graph neural networks (GNNs). A comprehensive survey can be found in [45]. Evaluating different design choices of graph representation network architectures is beyond the scope of this paper.

The decoder of the actor-network transforms the node embedding learned from the encoder network into edge

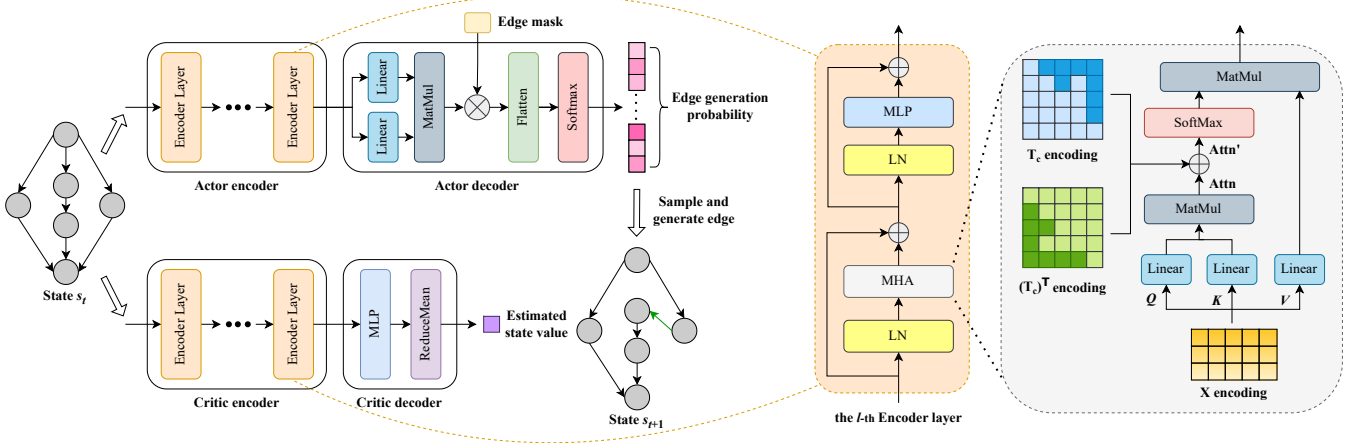


Fig. 8: Illustration of the actor and critic network.

generation probabilities. It first applies two linear layers to generate two linear transformations of each node embedding. Next, it conducts an inner product between the linear transformations of every two nodes to derive a scalar representing the edge generation score between the two nodes. Written in the matrix form, we have:

$$\mathbf{S} = (\mathbf{H}\mathbf{W}_1)(\mathbf{H}\mathbf{W}_2)^\top \quad (16)$$

where $\mathbf{W}_1 \in \mathbb{R}^{d \times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d \times d}$ denote the linear transformation matrices; $\mathbf{H} \in \mathbb{R}^{n \times d}$ denotes the node embedding learned from the encoder network; $\mathbf{S} \in \mathbb{R}^{n \times n}$ denotes the edge generation score matrix where $[\mathbf{S}]_{ij}$ is the edge generation score from node v_i to v_j . Then, the action mask is used to mask out the ineligible edges by setting their generation score to 0. Finally, a Flatten layer transforms the edge generation score matrix into a long vector, and a Softmax layer is applied to convert the edge generation scores into probabilities.

The decoder of the critic network is given by a MLP shared across different nodes. The input dimension of the MLP is the same as the dimension of node embedding, and the output dimension equals one. The MLP downscales each node's embedding into a scalar. The mean of scalars corresponding to all nodes is then used to estimate the cumulative return $\mathcal{R}(s)$ of the current input state s .

7 EVALUATION

7.1 DAG task generation

We follow the DAG generation method proposed in [29] to generate random DAG tasks for evaluation. The C++ implementation of the DAG generation method is available online¹. To evaluate the proposed algorithm on DAG tasks with various characteristics, we generate a DAG task set with two varied parameters: (i) task utilization defined as the sum of sub-task utilization (*i.e.*, $U = \sum_{i=1}^n U_i$, where $U_i = C_i/D$); (ii) task density defined as the ratio of the DAG length and its deadline (*i.e.*, $dens = L(\mathcal{G})/D$). For the task utilization, we consider seven ranges $U \in [\underline{U}, \underline{U} + 1)$ with \underline{U} varied from $\{1.0, 2.0, \dots, 7.0\}$. For the task density,

we consider five ranges $dens \in [\underline{dens}, \underline{dens} + 0.1)$ with \underline{dens} varied from $\{0.5, 0.6, \dots, 0.9\}$. For each combination of the above parameter ranges, we generate 3,000 random DAG tasks, which constitute a set of $7 \times 5 \times 3,000 = 105,000$ tasks. Then, we randomly split the whole task set into train, validation, and test sets using a ratio of 0.6 : 0.2 : 0.2, applying the split equally for each parameter range. We use the splits respectively for training the DRL agent, tuning the DRL hyperparameters and the final evaluation of the pre-trained DRL agent and other comparison algorithms.

We note that the task generation method does not support specifying the number of nodes as a parameter of DAG generation. To show the variety of n in our generated task set, we plot a histogram of n in Fig. 9. From the figure, we can see that $n \in [0, 140]$, with the majority of cases falling between 10 and 80.

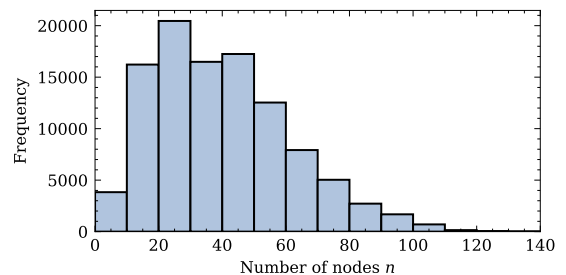


Fig. 9: Histogram of number of nodes n .

7.2 Comparison algorithms

We evaluate the proposed EGS framework and DRL algorithm by comparing the performance of the following DAG scheduling algorithms on the generated task set.

Edge generation scheduling heuristics. We consider three heuristics based on the proposed EGS framework, integrated with different edge generation policies, *i.e.*, PPO policy (EGS-PPO), greedy policy (EGS-GRD), and random policy (EGS-RND). In each iteration of EGS, EGS-PPO generates the edge with the maximum confidence given by a pre-trained PPO agent. EGS-GRD selects an edge that leads

1. <https://github.com/mive93/DAG-scheduling>

TABLE 1: Comparison results on processor usage.

Utilization	Density	EGS-PPO	EGS-GRD	EGS-RND	He2019	Zhao2020	He2021
[1.0, 2.0)	[0.5, 0.6)	2.06 ^{0.52}	2.22 ^{0.61}	2.26 ^{0.63}	2.06 ^{0.51}	2.09 ^{0.53}	2.06 ^{0.51}
	[0.6, 0.7)	2.10 ^{0.54}	2.19 ^{0.59}	2.23 ^{0.62}	2.14 ^{0.57}	2.15 ^{0.57}	2.12 ^{0.55}
	[0.7, 0.8)	2.14 ^{0.56}	2.20 ^{0.60}	2.28 ^{0.64}	2.20 ^{0.60}	2.23 ^{0.61}	2.17 ^{0.59}
	[0.8, 0.9)	2.20 ^{0.60}	2.29 ^{0.66}	2.31 ^{0.66}	2.29 ^{0.66}	2.33 ^{0.71}	2.27 ^{0.65}
	[0.9, 1.0)	2.39 ^{0.75}	2.46 ^{0.81}	2.48 ^{0.79}	2.47 ^{0.81}	2.55 ^{0.91}	2.47 ^{0.80}
[2.0, 3.0)	[0.5, 0.6)	3.32 ^{0.47}	3.73 ^{0.57}	3.68 ^{0.56}	3.38 ^{0.49}	3.41 ^{0.49}	3.36 ^{0.48}
	[0.6, 0.7)	3.38 ^{0.49}	3.74 ^{0.59}	3.78 ^{0.57}	3.50 ^{0.51}	3.56 ^{0.53}	3.50 ^{0.50}
	[0.7, 0.8)	3.49 ^{0.51}	3.79 ^{0.62}	3.84 ^{0.65}	3.65 ^{0.56}	3.70 ^{0.59}	3.60 ^{0.54}
	[0.8, 0.9)	3.67 ^{0.58}	3.99 ^{0.66}	4.05 ^{0.69}	3.92 ^{0.67}	4.09 ^{0.76}	3.90 ^{0.63}
	[0.9, 1.0)	3.99 ^{0.74}	4.33 ^{0.83}	4.35 ^{0.84}	4.32 ^{0.93}	4.58 ^{1.03}	4.28 ^{0.86}
[3.0, 4.0)	[0.5, 0.6)	4.17 ^{0.38}	4.82 ^{0.57}	4.74 ^{0.52}	4.39 ^{0.49}	4.46 ^{0.50}	4.37 ^{0.49}
	[0.6, 0.7)	4.44 ^{0.51}	4.97 ^{0.57}	4.90 ^{0.56}	4.63 ^{0.54}	4.72 ^{0.55}	4.63 ^{0.50}
	[0.7, 0.8)	4.60 ^{0.56}	5.07 ^{0.62}	5.10 ^{0.63}	4.90 ^{0.56}	5.01 ^{0.64}	4.85 ^{0.59}
	[0.8, 0.9)	4.91 ^{0.67}	5.47 ^{0.75}	5.51 ^{0.79}	5.33 ^{0.82}	5.59 ^{0.92}	5.24 ^{0.72}
	[0.9, 1.0)	5.39 ^{0.84}	5.95 ^{0.97}	6.06 ^{0.98}	5.95 ^{1.16}	6.44 ^{1.43}	5.87 ^{1.09}
[4.0, 5.0)	[0.5, 0.6)	5.48 ^{0.51}	6.26 ^{0.68}	6.07 ^{0.61}	5.59 ^{0.51}	5.67 ^{0.56}	5.60 ^{0.51}
	[0.6, 0.7)	5.63 ^{0.55}	6.49 ^{0.74}	6.35 ^{0.66}	5.90 ^{0.60}	5.97 ^{0.57}	5.86 ^{0.56}
	[0.7, 0.8)	5.89 ^{0.60}	6.71 ^{0.85}	6.60 ^{0.79}	6.24 ^{0.66}	6.44 ^{0.81}	6.20 ^{0.68}
	[0.8, 0.9)	6.30 ^{0.73}	7.12 ^{0.98}	7.19 ^{0.96}	6.80 ^{0.92}	7.20 ^{1.15}	6.71 ^{0.85}
	[0.9, 1.0)	6.87 ^{0.99}	7.71 ^{1.13}	7.92 ^{1.25}	7.77 ^{1.63}	8.55 ^{1.99}	7.47 ^{1.26}
[5.0, 6.0)	[0.5, 0.6)	6.40 ^{0.49}	7.51 ^{0.70}	7.22 ^{0.62}	6.67 ^{0.53}	6.80 ^{0.54}	6.63 ^{0.51}
	[0.6, 0.7)	6.68 ^{0.53}	7.82 ^{0.77}	7.56 ^{0.77}	7.05 ^{0.64}	7.19 ^{0.69}	7.04 ^{0.62}
	[0.7, 0.8)	7.01 ^{0.59}	8.05 ^{0.79}	7.95 ^{0.75}	7.53 ^{0.69}	7.72 ^{0.79}	7.39 ^{0.65}
	[0.8, 0.9)	7.47 ^{0.72}	8.61 ^{0.98}	8.60 ^{1.07}	8.13 ^{1.03}	8.54 ^{1.19}	8.00 ^{0.88}
	[0.9, 1.0)	8.41 ^{1.05}	9.62 ^{1.28}	9.66 ^{1.36}	9.71 ^{1.95}	10.73 ^{2.47}	9.14 ^{1.38}
[6.0, 7.0)	[0.5, 0.6)	7.66 ^{0.54}	9.01 ^{0.82}	8.63 ^{0.74}	7.88 ^{0.57}	8.01 ^{0.60}	7.85 ^{0.59}
	[0.6, 0.7)	8.03 ^{0.59}	9.37 ^{0.87}	9.05 ^{0.83}	8.36 ^{0.72}	8.55 ^{0.75}	8.32 ^{0.69}
	[0.7, 0.8)	8.48 ^{0.80}	9.77 ^{1.05}	9.73 ^{1.05}	9.05 ^{0.88}	9.25 ^{0.98}	8.91 ^{0.83}
	[0.8, 0.9)	9.03 ^{0.89}	10.41 ^{1.11}	10.45 ^{1.23}	9.81 ^{1.11}	10.19 ^{1.39}	9.60 ^{0.96}
	[0.9, 1.0)	10.10 ^{1.42}	11.71 ^{1.74}	11.89 ^{1.71}	11.83 ^{2.62}	13.14 ^{3.34}	11.20 ^{2.00}
[7.0, 8.0)	[0.5, 0.6)	9.08 ^{0.65}	10.91 ^{1.06}	10.26 ^{0.86}	9.28 ^{0.65}	9.42 ^{0.68}	9.26 ^{0.65}
	[0.6, 0.7)	9.42 ^{0.76}	11.21 ^{1.17}	10.73 ^{0.96}	9.88 ^{0.81}	10.03 ^{0.84}	9.79 ^{0.80}
	[0.7, 0.8)	10.02 ^{0.90}	11.72 ^{1.26}	11.59 ^{1.15}	10.69 ^{1.05}	10.84 ^{1.05}	10.55 ^{1.00}
	[0.8, 0.9)	10.78 ^{1.06}	12.60 ^{1.43}	12.73 ^{1.44}	11.71 ^{1.39}	12.28 ^{1.79}	11.46 ^{1.21}
	[0.9, 1.0)	12.00 ^{1.80}	14.13 ^{2.20}	14.37 ^{2.35}	14.32 ^{3.49}	15.76 ^{4.22}	13.10 ^{2.18}

to the maximum intermediate width reduction, with a tie-breaking strategy of selecting the one with the minimum intermediate length increase. EGS-RND generates an edge uniformly at random. We note that all three policies only generate edges that are deemed eligible by the edge masks, as defined in the EGS framework. The effectiveness of the proposed PPO policy can be evaluated through the comparison with EGS-GRD and EGS-RND.

Mixed-integer linear programming (MILP). We formulate the DAG scheduling problem as a mixed-integer linear program that can be solved by standard mathematical programming solvers to obtain optimal solutions. Although the optimality can be guaranteed, MILP is not computationally tractable. Thus, it can only be used to solve relatively small instances within a reasonable time (in our experiments, DAGs with $n \leq 20$ are solved by MILP within a 2-hour time limit). Through the comparison with MILP, the optimality gap of each comparison algorithm can be acquired. The detailed formulation of the MILP is reported in Appendix A.

State-of-the-art DAG scheduling heuristics. The proposed EGS framework is compared with three state-of-the-art DAG scheduling algorithms: He2019 [18], Zhao2020 [19], and He2021 [20]. These algorithms are all developed based on *list scheduling* framework. The main differences are the priorities assigned to the DAG nodes. For example, He2021 uses the vertex lengths (*i.e.*, $t_i^{\text{EFT}} + D - t_i^{\text{LFT}}$) as node priorities, while He2019 and Zhao2020 developed more sophisticated priority assignment rules. We note that

the objective used in these list scheduling heuristics is to minimize the *makespan* of a DAG task given a fixed number of processors. However, they can be easily adapted to minimize processor usage given task deadline through an incremental search as shown in Appendix B.

7.3 Experimental setup

All the experiments are conducted on a workstation equipped with AMD EPYC 7763 CPUs and Nvidia A100 GPUs running GNU/Linux. The proposed EGS framework is implemented in Python 3.8.12, and the PPO algorithm is implemented using Tensorflow 2.7.0. The MILP is solved by a standard mathematical programming solver Gurobi 9.5.0² with a Python interface. The hyper-parameters used in our PPO implementation are reported in Appendix C.

The training of the DRL agent takes approximately 30 hours. The trained actor network is used to make edge generation decisions on unseen tasks in the test set, taking on average 3.53 seconds to schedule one DAG task.

7.4 Comparison results

7.4.1 Overall results

Table 1 compares the proposed EGS algorithms with the state-of-the-art DAG scheduling heuristics. It summarizes each algorithm’s average processor usage and associated

2. <https://www.gurobi.com>

TABLE 2: Average optimality gap $\frac{M_{alg}-M_{opt}}{M_{opt}}$ when $n \leq 20$.

EGS-PPO	EGS-GRD	EGS-RND	He2019	Zhao2020	He2021
1.78%	7.20%	8.92%	5.95%	8.86%	5.29%

standard deviation (indicated in the superscript) under different task utilization and density. We use **boldface** type to indicate the best results within the comparison. The results show that EGS-PPO algorithm outperforms other list scheduling heuristics in terms of processor usage across all tested utilization and density levels, which demonstrates the effectiveness of the proposed EGS framework and PPO policy. Additionally, the performance gain of EGS-PPO improves as the task utilization and density increase. EGS-GRD and EGS-RND show a similar trend. In particular, they perform worse than other algorithms at lower densities but perform better than Zhao2020 at higher densities. This indicates that EGS framework has more potential to schedule DAGs with higher densities than list scheduling methods.

By comparing EGS-GRD and EGS-RND, it shows that the greedy policy performs better than the random when utilization is low, or density is high. This is expected since the greedy policy tends to lead the decision process to local optima, thus it is more difficult to achieve globally good performance for a long decision episode.

To further understand how the algorithms perform compared to the optimal solution, we compute their optimality gaps, which are defined as the relative performance deviations between the test algorithms and the optimal solution, *i.e.*, $\frac{M_{alg}-M_{opt}}{M_{opt}}$, where M_{alg} and M_{opt} denote the number of processors used by the test algorithm and the optimal solution, respectively. In our experiments, the optimal solutions are obtained by solving MILP. Since the MILP solver is not computationally tractable, we run it on DAGs with $n \leq 20$ and present the average optimality gap of each comparison algorithm in Table 2. It shows that EGS-PPO achieves the best optimality gap (smaller than 2%) among all comparison algorithms. Moreover, EGS-GRD and EGS-RND achieve similar optimality gaps to Zhao2020.

7.4.2 Sensitivity of task utilization

Fig. 10 illustrates the average number of processors required by each algorithm under different task utilization U . The figure shows similar findings as in Table 1 that EGS-PPO outperforms other algorithms for all task utilization levels, and the performance gain increases with task utilization. With regards to the list scheduling heuristics, He2021 achieves the best performance, demonstrating that its priority assignment is better than He2019 and Zhao2020.

Additionally, we compare the acceptance ratio (*i.e.*, $\frac{\text{\# schedulable tasks}}{\text{\# tasks in the test set}}$) for each utilization level with increments of 0.1) achieved by each algorithm given $M = 8$. We can see from Fig. 11 that EGS-PPO achieves the best acceptance ratio among all algorithms. While EGS-GRD and EGS-RND achieve high acceptance ratio for low utilization tasks (*i.e.*, better than He2019 and Zhao2020 for $U < 4$ and $U < 5$, respectively), their performance drops rapidly as $U > 6$. This is because the advantage of EGS-GRD and EGS-RND lies in scheduling high-density tasks (see Table 1). As $U \rightarrow M$, most high-density tasks require more than M processors for

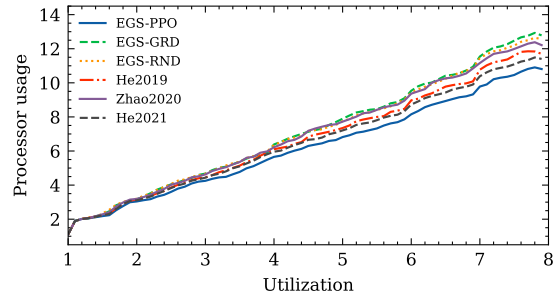


Fig. 10: Average processor usage with various U .

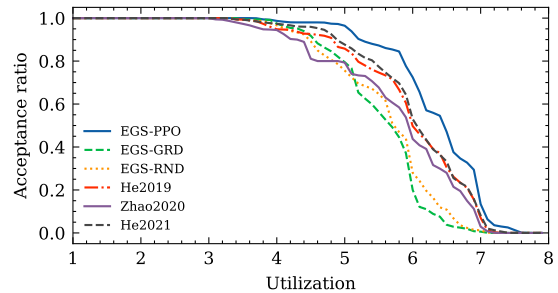


Fig. 11: Acceptance ratio with $M = 8$ and various U .

all algorithms (see Table 1). Thus, the advantage of achieving lower processor usage for high-density tasks cannot contribute to the acceptance ratio. More details about the impact of task density on processor usage are discussed in the following.

7.4.3 Sensitivity of task density

Fig. 12 illustrates the processor usage with different task densities with a violin plot. The ticks of each violin show the maximum, average, and minimum (from top to bottom) processor usage among all the test instances within each density range. The figure shows that the overall processor usage increases with task density, and EGS-PPO outperforms other algorithms in terms of average processor usage and performance stability, as indicated by the violin size. In particular, when $dens \geq 0.9$, EGS-PPO can save up to 5 and 8 processors compared to He2019 and Zhao2020, respectively. Moreover, it shows similar findings as in Table 1 that EGS-GRD and EGS-RND perform better than He2019 and Zhao2020 for high task density ($dens \geq 0.9$).

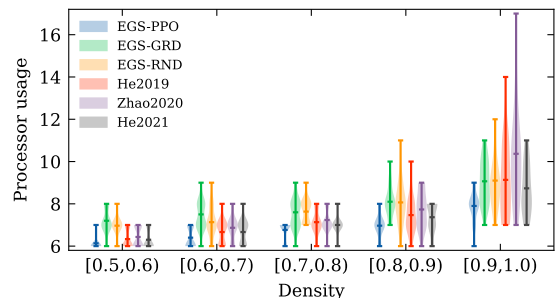


Fig. 12: Processor usage with $U = 5$ and various $dens$.

8 CONCLUSION AND FUTURE WORK

In this paper, we studied the problem of scheduling real-time DAG tasks on multicore platforms to minimize processor usage while guaranteeing schedulability. We presented a new schedulability test based on the observation that a DAG task is schedulable if its width is not greater than the number of available processors and its length is less than or equal to the deadline. A new Edge Generation Scheduling (EGS) framework is proposed that converts a DAG task into a trivially schedulable DAG by iteratively adding edges. A DRL algorithm combined with a graph representation neural network is developed to learn an efficient edge generation policy for EGS. The effectiveness of three EGS variants (*i.e.*, EGS-PPO, EGS-GRD, and EGS-RND) was evaluated by comparing to exact solutions and state-of-the-art DAG scheduling algorithms. Experimental results show that EGS-PPO outperforms other approaches, while EGS-GRD and EGS-RND achieve comparable results to the state-of-the-art for low-utilization and high-density tasks.

Although the main focus of the paper is on the schedulability aspects of a DAG task, we note that the approach can also be extended to solve joint optimization of schedulability and other timing attributes such as reaction time or data age. Additionally, given that the proposed method requires fewer processors than the state-of-the-art, it is expected to also provide better results when scheduling multiple tasks under a federated scheduling policy (*e.g.*, [24]).

In the future, we plan to study real-time DAG scheduling in heterogeneous platforms. We are also interested in extending the proposed method to minimize the makespan of a DAG, which is important for production scheduling and cloud computing applications.

REFERENCES

- [1] M. Andreozzi, G. Gabrielli, B. Venu, and G. Travaglini, "Industrial Challenge 2022: A High-Performance Real-Time Case Study on Arm," in *EuroMicro Conference on Real-Time Systems (ECRTS)*, vol. 231, 2022, pp. 1:1–1:15.
- [2] A. Hamann, D. Dasari, F. Wurst, I. Saudo, N. Capodici, P. Burgio, and M. Bertogna, "WATERS industrial challenge," in *Proceedings of the 10th International Workshop on Analysis Tools and Methodologies for Embedded Real-Time Systems (WATERS)*, 2019.
- [3] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, "Latency-aware generation of single-rate DAGs from multi-rate task sets," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 226–238.
- [4] J. D. Ullman, "NP-complete scheduling problems," *Journal of Computer and System sciences*, vol. 10, no. 3, pp. 384–393, 1975.
- [5] M. Verucchi, "A comprehensive analysis of DAG tasks: solutions for modern real-time embedded systems," Doctoral Dissertation, University of Modena and Reggio Emilia, Italy, 2020.
- [6] J. Li, K. Agrawal, and C. Lu, "Parallel real-time scheduling," in *Handbook of Real-Time Computing*. Springer, 2022, pp. 447–467.
- [7] M. Verucchi, I. S. Olmedo, and M. Bertogna, "A survey on real-time DAG scheduling, revisiting the global-partitioned infinity war," *Real-Time Systems*, vol. 59, no. 3, pp. 479–530, 2023.
- [8] S. Baruah, "Scheduling DAGs when processor assignments are specified," in *ACM International Conference on Real-Time Networks and Systems (RTNS)*, 2020, pp. 111–116.
- [9] S. Chang, J. Sun, Z. Hao, Q. Deng, and N. Guan, "Computing exact WCRT for typed DAG tasks on heterogeneous multi-core processors," *Journal of Systems Architecture*, vol. 124, p. 102385, 2022.
- [10] S. Ahmed and J. H. Anderson, "Exact response-time bounds of periodic DAG tasks under server-based global scheduling," in *IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 447–459.
- [11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
- [12] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 28 877–28 888, 2021.
- [13] A. Minaeva, D. Roy, B. Akesson, Z. Hanzalek, and S. Chakraborty, "Control performance optimization for application integration on automotive architectures," *IEEE Transactions on Computers*, vol. 70, no. 7, pp. 1059–1073, 2021.
- [14] AUTOSAR, "Requirements on Timing Extensions," Standard, 2022. [Online]. Available: https://www.autosar.org/fileadmin/standards/R22-11/FO/AUTOSAR_RS_TimingExtensions.pdf
- [15] ARINC, "Avionics Application Software Standard Interface, Part 0, Overview of ARINC 653," Standard, 2021. [Online]. Available: <https://aviation-ia.sae-itc.com/events/avionics-application-executive-apex-software-subcommittee>
- [16] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *IEEE Real-Time Systems Symposium (RTSS)*, 2012, pp. 63–72.
- [17] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.
- [18] Q. He, N. Guan, Z. Guo *et al.*, "Intra-task priority assignment in real-time scheduling of DAG tasks on multi-cores," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2283–2295, 2019.
- [19] S. Zhao, X. Dai, I. Bate, A. Burns, and W. Chang, "DAG scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency," in *IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 128–140.
- [20] Q. He, M. Lv, and N. Guan, "Response time bounds for DAG tasks with arbitrary intra-task priority assignment," in *EuroMicro Conference on Real-Time Systems (ECRTS)*, 2021, pp. 8:1–8:21.
- [21] Q. He, N. Guan, M. Lv, X. Jiang, and W. Chang, "Bounding the response time of DAG tasks using long paths," in *IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 474–486.
- [22] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese, "Feasibility analysis in the sporadic DAG task model," in *EuroMicro Conference on Real-Time Systems (ECRTS)*, 2013, pp. 225–233.
- [23] S. Baruah, "Improved multiprocessor global schedulability analysis of sporadic DAG task systems," in *EuroMicro Conference on Real-Time Systems (ECRTS)*, 2014, pp. 97–105.
- [24] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," in *26th EuroMicro Conference on Real-Time Systems (ECRTS)*, 2014, pp. 85–96.
- [25] R. Pathan, P. Voudouris, and P. Stenström, "Scheduling parallel real-time recurrent tasks on multicore platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 4, pp. 915–928, 2018.
- [26] Y. Yadlapalli and C. Liu, "LAG-based analysis techniques for scheduling multiprocessor hard real-time sporadic DAGs," in *IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 316–328.
- [27] S. Zhao, X. Dai, and I. Bate, "DAG scheduling and analysis on multi-core systems by modelling parallelism and dependency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4019–4038, 2022.
- [28] S. Baruah, V. Bonifaci, and A. Marchetti-Spaccamela, "The global edf scheduling of systems of conditional sporadic DAG tasks," in *EuroMicro Conference on Real-Time Systems (ECRTS)*, 2015, pp. 222–231.
- [29] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo, "Response-time analysis of conditional DAG tasks in multiprocessor systems," in *EuroMicro Conference on Real-Time Systems (ECRTS)*, 2015, pp. 211–221.
- [30] N. Ueter, M. Günzel, and J.-J. Chen, "Response-time analysis and optimization for probabilistic conditional parallel DAG tasks," in *IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 380–392.
- [31] K. Yang, M. Yang, and J. H. Anderson, "Reducing response-time bounds for DAG-based task systems on heterogeneous multicore platforms," in *ACM International Conference on Real-Time Networks and Systems (RTNS)*, 2016, pp. 349–358.

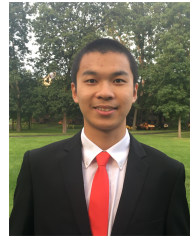
- [32] S. Chang, X. Zhao, Z. Liu, and Q. Deng, "Real-time scheduling and analysis of parallel tasks on heterogeneous multi-cores," *Journal of Systems Architecture*, vol. 105, p. 101704, 2020.
- [33] H. Zahaf, N. Capodiceci, R. Cavicchioli, G. Lipari, and M. Bertogna, "The HPC-DAG task model for heterogeneous real-time systems," *IEEE Transactions on Computers*, vol. 70, no. 10, pp. 1747–1761, 2021.
- [34] F. Reghenzani, A. Bhuiyan, W. Fornaciari, and Z. Guo, "A multi-level DPM approach for real-time DAG tasks in heterogeneous processors," in *IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 14–26.
- [35] R. Bi, Q. He, J. Sun, Z. Sun, Z. Guo, N. Guan, and G. Tan, "Response time analysis for prioritized DAG task with mutually exclusive vertices," in *IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 460–473.
- [36] H. Mao, M. Schwarzkopf, S. B. Venkatakrisnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM special interest group on data communication (SIGCOMM)*, 2019, pp. 270–288.
- [37] P. Sun, Z. Guo, J. Wang, J. Li, J. Lan, and Y. Hu, "Deepweave: Accelerating job completion time with deep reinforcement learning-based coflow scheduling," in *International Conference on International Joint Conferences on Artificial Intelligence (IJCAI)*, 2021, pp. 3314–3320.
- [38] H. Lee, S. Cho, Y. Jang, J. Lee, and H. Woo, "A global DAG task scheduler using deep reinforcement learning and graph convolution network," *IEEE Access*, vol. 9, pp. 158 548–158 561, 2021.
- [39] W. Jeon, M. Gagrani, B. Bartan, W. W. Zeng, H. Teague, P. Zappi, and C. Lott, "Neural DAG scheduling via one-shot priority sampling," in *International Conference on Learning Representations (ICLR)*, 2023.
- [40] R. W. Floyd, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.
- [41] R. Dilworth, "A decomposition theorem for partially ordered sets," *Annals of Mathematics*, pp. 161–166, 1950.
- [42] J. E. Hopcroft and R. M. Karp, "An $n^5/2$ algorithm for maximum matchings in bipartite graphs," *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.
- [43] J. Bang-Jensen and G. Z. Gutin, *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.
- [44] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.
- [45] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.



Binqi Sun received the master's degree in control science and engineering from the Department of Automation, Tsinghua University, Beijing, China, in 2020. He is currently a doctoral candidate in computer science with TUM School of Computation, Information and Technology, Technical University of Munich, Germany. His research interests include real-time scheduling, combinatorial optimization, and cyber-physical systems.



Mirco Theile received the M.Sc. degree in electrical engineering and information technology from Technical University of Munich, Germany, in 2018, where he is currently pursuing the Ph.D. degree. Currently, he is a visiting researcher at University of California in Berkeley, USA. His current research interests extend to reinforcement learning in applications of cyber-physical systems, including UAVs, robotics, and real-time systems.



Ziyuan Qin received his B.Sc. degree in operations research from Cornell University in 2018. He is a master student studying computer science at Technical University of Munich, Germany. His current research interests include graph neural networks, explainable artificial intelligence, and robot learning.



Daniele Bernardini received his M.Sc. degree in Theoretical Physics at the University of Florence in 1997. After graduation, he spent 2 more years as a researcher at the Ludwig Maximilians University, Munich before transitioning to the industry, where he gained more than 20 years of experience in software development and data science. In 2021 he joined the Technical University of Munich as research group leader where he focuses on advancing perception for robotic manipulation. Since 2021 he is a co-founder and CEO of Cognivix, a startup specializing in automation solutions for industries requiring high variability and low volume production.



Debayan Roy obtained his Ph.D. in electrical and computer engineering from the Technical University of Munich, Germany in 2020. Since 2022, he is a Senior Research Engineer at Huawei Munich Research Center, Germany in the Automotive Software Platform Laboratory. His research interests primarily include modeling, design and analysis of real-time cyber-physical systems. He won the best paper award at RTCSA 2017.



Andrea Bastoni received his Ph.D. in computer engineering from the University of Rome Tor Vergata in 2011. Since 2007, he has been working in the industry as Software Engineer and Software Architect and has acquired more than 10 years of experience in real-time operating systems for safety-critical certified products in the avionics, railway, and automotive fields. In 2020, he joined the Technical University of Munich as Research Fellow focusing on safety-critical RTOS, predictability on complex heterogeneous architectures, and virtualization.



Marco Caccamo earned his Ph.D. in computer engineering from Scuola Superiore Sant'Anna (Italy) in 2002. Shortly after graduation, he joined University of Illinois at Urbana-Champaign as assistant professor in Computer Science and was promoted to full professor in 2014. Since 2018, Prof. Caccamo has been appointed to the chair of Cyber-Physical Systems in Production Engineering at Technical University of Munich, Germany. In 2003, he was awarded an NSF CAREER Award. He is a recipient of the Alexander von Humboldt Professorship and he is IEEE Fellow.

APPENDIX A MILP FORMULATION

The DAG scheduling problem is formulated as a mixed-integer linear program (MILP) with the notations in Table 3.

TABLE 3: Table of notations used in the MILP.

Notation	Implication
Problem data	
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Task graph with n nodes (sub-tasks);
D	Task deadline;
m	Width of the task graph (<i>i.e.</i> , $W(\mathcal{G}) = m$);
i, j	Node index, $i = 1, \dots, n, j = i, \dots, n$;
k	Processor index, $k = 1, \dots, m$;
C_i	WCET of node i ;
M_1, M_2	Two large constant numbers.
Decisions	
x_{ik}	1, if node v_i executes on processor k ; 0, otherwise;
y_k	1, if any node executes on processor k ; 0, otherwise;
γ_{ij}	1, if node v_i and j execute on the same processor, and node v_i executes later than v_j ; 0, otherwise;
f_i	Finishing time of node v_i .

$$\text{minimize } \sum_{k=1}^m y_k \quad (17)$$

subject to:

$$\sum_{k=1}^m x_{ik} = 1, \quad \forall v_i \in \mathcal{V} \quad (18)$$

$$f_i \leq f_j - C_j + M_1 \cdot \gamma_{ij} + M_2 \cdot (2 - x_{ik} - x_{jk}), \forall v_i \neq v_j, k = 1, \dots, m \quad (19)$$

$$f_j \leq f_i - C_i + M_1 \cdot (1 - \gamma_{ij}) + M_2 \cdot (2 - x_{ik} - x_{jk}), \forall v_i \neq v_j, k = 1, \dots, m \quad (20)$$

$$C_i \leq f_i, \quad \forall v_i \in \mathcal{V} \quad (21)$$

$$f_i \leq D, \quad \forall v_i \in \mathcal{V} \quad (22)$$

$$f_i + C_j \leq f_j, \quad e_{ij} \in \mathcal{E} \quad (23)$$

$$x_{ik} \leq y_k, \quad \forall v_i \in \mathcal{V}, k = 1, \dots, m \quad (24)$$

Objective (17) minimizes the number of processors used to schedule the DAG task. Constraints (18) ensure that each node is assigned to one and only one processor to execute. Constraints (19) and (20) guarantee the execution order of the nodes assigned to the same processor and make sure there is at most one node running on each processor at each time instant. Constraints (21) and (22) ensure all the nodes start and finish their execution no earlier than the release time 0 and no later than the deadline D , respectively. Constraints (23) implement the precedence constraints between the nodes. Constraints (24) indicate busy processors (*i.e.*, the processors to which at least one node is assigned).

Algorithm 3: Incremental search to minimize processor usage with list scheduling heuristics

Input: (\mathcal{G}, D) : the DAG task to be scheduled;
Output: M^* : number of processors used;

```

1 for  $M^* \leftarrow 1$  to  $n$  do
2    $Makespan \leftarrow \text{ListSched}(\mathcal{G}, M^*)$  [3], [18], [19];
3   if  $\mathcal{G}$  is schedulable (i.e.,  $Makespan \leq D$ ) then
4     return  $M^*$ ;

```

APPENDIX B INCREMENTAL SEARCH FOR LIST SCHEDULING

Algorithm 3 presents the incremental search procedures of minimizing processor usage with list scheduling heuristics.

APPENDIX C PPO HYPERPARAMETERS

We summarized the hyperparameters of the PPO implementation in Table 4.

TABLE 4: PPO hyper-parameters.

Hyper-parameter	Value	Hyper-parameter	Value
Discount factor γ	0.99	Number of encoder layers	2
GAE parameter λ	0.97	Number of attention heads	8
Clipping parameter ϵ	0.2	Node embedding dimension	64
Number of iterations	500	MLP hidden dimension	64
Length of rollout	50,000	Initial learning rate	10^{-4}
Batch size	100	End learning rate	10^{-5}
Epochs per iteration	10	Learning rate decay steps	10^6

Chapter 7

Discussion

7.1 Summary

This thesis has explored the application of reinforcement learning (RL) to cyber-physical systems (CPS), specifically focusing on planning, control, and scheduling challenges. Motivated by the development of a long-endurance solar unmanned aerial vehicle (UAV), the TUM-UIUC Solar Flyer, the different chapters explored the diverse challenges of applying RL to ultimately real-world CPS.

The autopilot development in Chapter 3 emphasized the complexity of a CPS, focusing mainly on the cyber component. It showed the development of the autopilot framework *uavAP*¹, its core *cpsCore*², and its implementation of a planning and control stack. The applications of the framework to different fixed-wing aircraft showed its adaptability and the long-endurance flight tests showed its reliability. The derived *beta trajectory* describing the evasion trajectory of small-size fixed-wing UAVs with roll-rate constraints shows how feasibility models can be created for this aircraft type, enabling the future implementation of safety layers for RL algorithms.

Chapter 4 showed how maps of the environment can be used for UAV path planning with RL. It elaborated on the importance of map processing in aiding shallow neural network structures in learning generalizing policies for coverage path planning and path planning for data harvesting. Additionally, maps enable implicit collaboration among multi-agent UAVs as they allow the straightforward expression of all UAV positions in the maps. The projects further lead to the open-source code repositories *uavSim*³ for coverage path planning, and *uav_data_harvesting*⁴ for the data harvesting problem. The chapter illustrates that RL could be used in increasingly more complex challenges, hinting at the scalability of RL toward real-world applications in UAV path planning.

The papers on the RL challenges in real-world control problems in Chapter 5 introduced a *cloud-edge*⁵ framework for online learning RL policies in a real-time setting and

¹<https://github.com/theilem/uavAP>

²<https://github.com/theilem/cpsCore>

³<https://github.com/theilem/uavSim>

⁴https://github.com/hbayerlein/uav_data_harvesting

⁵https://github.com/HP-CAO/cloud_edge

an action mapping concept to efficiently exploit model knowledge. The former allows training or fine-tuning off-policy RL algorithms when the computing unit of the controller is limited, and the sim-to-real gap is too large to directly deploy a pretrained agent to the real-world system. It further elaborated on the challenges of real-world systems, such as non-Markovian behavior, and emphasized the importance of timing accuracy for the collection of training data. The action mapping framework presented in the latter is intended to efficiently utilize models that predict the feasibility of actions during RL training. To enable this framework, it established how to learn to generate all feasible actions, leading to a feasibility policy that can be used as a map from a latent space to all feasible actions. An objective policy that aims to maximize a value function can use this map by searching for the optimal action in the latent space instead of in the action space, likely leading to significant training acceleration.

Chapter 6 focuses on directed acyclic graphs (DAGs) for task scheduling in CPS. The first paper shows how periodic multi-rate tasksets can be transformed into DAGs by expressing timing constraints through dummy and synchronization nodes. It further shows how end-to-end latencies of task chains can be bounded and minimized by adding edges to the DAG. The second paper dives deeper into the scheduling of DAG tasks by observing that DAGs with specific characteristics are trivial to schedule and that all DAGs that are schedulable can be transformed into a trivially schedulable DAG by adding edges. For the resulting edge generation scheduling (*EGS*⁶) framework, RL can be used to find appropriate edges to add to find a schedule for a given DAG. Using action masks that disallow adding specific edges derived from topological and temporal features of the DAG, model knowledge was used to significantly accelerate the training of the RL agent. After adding all the necessary edges, EGS yields a trivially schedulable DAG, which is proven to be schedulable, providing guarantees required in real-time scheduling.

The critical challenges addressed in this thesis are formulating problems in CPS to effectively apply RL and including prior knowledge in the learning process. Besides insights and novel methodologies to address these challenges, the thesis also yielded open-source code repositories to help advance the applicability of RL to CPS. While RL is a powerful tool for many challenges in CPS and other fields, an open question is, which problems are suitable for RL?

7.2 The Law of the Hammer

“If the only tool you have is a hammer,
it is tempting to treat everything as if it were a nail.”
– Abraham Maslow, 1966 [85]

The **law of the hammer**, which is commonly attributed to Abraham Maslow, describes the cognitive bias of overly relying on a familiar tool. Reinforcement learning is not necessarily a familiar tool but an attractive one in research and industry. Consequently, there is a tendency to apply RL to problems for which it may not be the most suitable

⁶<https://github.com/binqi-sun/egs>

solution. This section aims to summarize the experiences gathered for this thesis in identifying problems that are true nails for the RL hammer.

A guideline for deciding whether to use RL for a given problem could be answering the following questions about the problem:

1. Is the problem complex to solve?
2. Can the problem be solved interactively?
3. Is the optimal policy unknown?
4. Can the problem be simulated?
5. Does the problem require generalization?

If all of these questions are answered with *yes*, RL may be an applicable solution for the problem. Otherwise, other algorithms may be preferred. The following provides details to the questions.

1) Is the problem complex to solve?

The first question should be whether the problem is too complex to solve efficiently with traditional algorithms or hand-crafted heuristics. This question can be reframed to whether simpler solutions fail, are insufficient, or lack scalability. Generally, the state space should be large or not directly observable, yielding large observation spaces. In small state or observation spaces, it is often easier to formulate a heuristic to solve the problem. Control problems should be non-linear since otherwise optimal solutions are trivial to derive. If it is a combinatorial problem, NP-hard problems are usually complex enough to motivate using RL, as no efficient and scalable optimal solution exists.

One motivation for using RL is its potential to scale while the problem formulation progressively approaches real-world applications. Consider the CPP problem discussed in Section 4.1. While the initial problem could have been solved with other algorithms, the proposed RL approach could be adapted to solve the more complex challenge in Section 4.3 and an even harder challenge in preliminary results in [86]. It thus holds the potential to be a successful approach for the continuous-world CPP problem that the UIUC-TUM Solar Flyer faces in the agricultural monitoring case study discussed in Section 1.3.

2) Can the problem be solved interactively?

The second question ensures that the problem at hand is an optimization problem that can be formulated as an MDP and solved iteratively in an interactive fashion. It further verifies that the goal of the problem is to find a policy that solves this problem adequately. Most classification or regression problems do not typically meet this condition, indicating that RL may not be applicable in such cases. However, some classification and regression problems can be reformulated.

Take the DAG scheduling problem from Section 6.2 as an example. The original question in that paper was: “Is a given DAG task schedulable using a specific number

of cores?”, a classification problem. The classification problem was converted into a regression problem by asking: “How many cores are needed to schedule the DAG task?” If the answer is lower than the number of available cores, it is schedulable on these cores. Ultimately, the question was formulated as an optimization problem: “Minimize the number of cores needed to schedule the DAG task.” Therefore, formulating it as an MDP and solving it with RL is possible.

3) Is the optimal policy unknown?

For some interactive problems where a policy is needed, there already exist an optimal policy, such as numerical solvers, or sufficiently good policies, such as human demonstration data. In both cases, the task is usually to replace the expensive optimal solver or the human with a cheaper approximate solution. If the quantity of data is sufficiently large, it is likely advantageous to utilize techniques such as imitation learning [87] instead of RL. In case a policy is searched for that is better than the policies from which the data is gathered, e.g., if the data is coming from different suboptimal heuristics, offline or batch RL methods [88] (not to be confused with off-policy RL) are the go-to solution.

The primary application area of RL is the set of problems for which optimal solutions or good solutions are unknown, and humans do not naturally solve the problem well. The problems in this thesis, primarily CPP and DAG scheduling, do not have efficient optimal policies, and human demonstration data is not beneficial as these tasks are too complex for humans to solve.

4) Can the problem be simulated?

Since RL requires learning from interactions, it is highly beneficial if the problem can be simulated efficiently, and the sim-to-real gap, discussed in Section 5.1, needs to be manageable. If RL algorithms become more sample efficient and robust and safe methods are developed, learning directly on the real system may be possible. However, given the algorithms currently available, direct application is feasible only in particular contexts, such as in [84], underscoring the necessity of simulations for RL.

In some real-world problems, no models exist for the environment with which the agent interacts, but there is an abundance of data. If the environment is unaffected by the agent’s behavior, it may be possible to train an RL agent based on a learned environment model. However, this may become impossible if the agent’s decisions affect the environment. This problem is common in multi-agent scenarios, in which the other agents are unknown but reactive to the agent. Therefore, training an RL agent in data-driven models is only applicable if the agent’s decisions do not change the environment.

5) Does the problem require generalization?

The final question is critical to determining if RL is the appropriate tool for a given problem. The primary advantage of RL, specifically deep RL, over other optimization solvers lies in generalizing a policy to unseen problem instances. Therefore, RL should

only be used for problems where a good policy for *all* instances of a problem class is sought.

Consider the CPP example in Section 4.3 and [86]. If only one specific scenario is given, i.e., one map with one set of target zones and one level of battery charge, RL could be used to find the optimal solution for this scenario. However, it would most likely require much longer than employing any numerical solver or branch-and-bound method. Similarly, in the DAG scheduling example in Section 6.2, other optimization methods would be better suited if searching for the schedule of one specific DAG task. However, in this thesis’s CPP and DAG scheduling problems, the objective is to find a policy for *all* instances of a problem, motivating the use of RL.

Summary

Thus, a one-sentence summary of RL could be: Reinforcement learning is a tool that utilizes large quantities of interactions to find a generalizing policy for an optimization problem that solves all instances of that problem or problem class sufficiently well.

7.3 Future Work

For RL in planning, specifically in the field of UAV path planning, future work focuses on scaling the problem toward the real-world application of quadcopter or fixed-wing aircraft planning. A challenge that needs to be addressed is that most tasks, including the “agricultural monitoring as a service” case study in this thesis, require long-horizon planning. Preliminary work on the problem of power-constrained coverage path planning with recharge [86] shows the difficulties of long-horizon problems and offers mitigating strategies that can be further explored. An additional problem is the exorbitant amount of interactions with the environment needed for RL. This could be reduced by leveraging existing system knowledge, such as symmetries, as done in other preliminary work [89].

When transitioning to continuous state and action spaces, temporal abstraction of the action space by parameterizing splines as trajectory segments to follow could be beneficial to not exacerbate the long-horizon challenge. Additionally, the local planner in *uavAP* is designed to follow splines that are currently generated by the *SplineGlobalPlanner*. With the parameters of splines as the action space, deriving a feasibility model is possible, yielding the question of how to efficiently include it in the learning process. Action mapping, conceptualized in this thesis, will be explored as the methodology to integrate the model knowledge and safety assurance into the RL agent. Besides the specific challenge of path planning, future work will also explore action mapping’s potential for efficiently safe RL in other contexts.

If action mapping can provide safety guarantees during and after training, the RL agent can be deployed and fine-tuned on the real physical UAV. However, the computation unit on the physical system is likely constrained in performance due to space, cooling, and power constraints. To enable online fine-tuning, the proposed cloud-edge framework can be used to offload the training to the ground station and only keep the inference on

board. However, the framework needs to be adapted to handle the intermittent data communication between the UAV edge and the base station cloud.

Despite offloading the training computation to the ground station, the computations for perception, planning, control, and data processing will remain onboard. The resulting computational load on the onboard device will further require advances in real-time scheduling for large parallelizable tasksets. Formulating the multi-rate taskset as a DAG and using EGS for scheduling may prove advantageous. However, advances in the EGS algorithm will be required to handle pipelined tasksets and tasks with arbitrary deadlines, which are likely found in the data processing tasks. Furthermore, the EGS algorithm can be reformulated for makespan minimization to enable its application to various other fields.

In conclusion, the projects in this thesis enable further development in many branches of RL in CPS, ultimately leading to the development of an intelligent version of the TUM-UIUC Solar Flyer.

Appendices

Appendix A

Reuse Statements

A.1 Trajectory Estimation for Geo-Fencing Applications on Small-Size Fixed-Wing UAVs

4/4/24, 2:01 PM

Rightslink® by Copyright Clearance Center



Sign in/Register

Trajectory Estimation for Geo-Fencing Applications on Small-Size Fixed-Wing UAVs

Conference Proceedings:
2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

Author: Mirco Theile
Publisher: IEEE
Date: November 2019

Copyright © 2019, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis online.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK
CLOSE WINDOW

© 2024 Copyright - All Rights Reserved | Copyright Clearance Center, Inc. | [Privacy statement](#) | [Data Security and Privacy](#)
 | [For California Residents](#) | [Terms and Conditions](#) Comments? We would like to hear from you. E-mail us at customercare@copyright.com



<https://s100.copyright.com/AppDispatchServlet#formTop>


1/1

A.2 uavAP: A Modular Autopilot Framework for UAVs

Rights and Permissions | AIAA
20/024, 2:57 PM

submission process. All authors and/or authorized individuals must assert that the manuscript is cleared for release, if required; acknowledge the originality and publication status of the work; and grant copyright or a license for AIAA's use of the material.

An explanation of the submission requirements and copyright options available to authors can be found here.



copyright ownership and appropriate transfer or license.

Reuse Licenses and Permission Requests

Author Reuse Rights and Posting Policy ↓

(click to hide)

Transfer of copyright to AIAA does not prevent authors from reproducing or adapting their work, in whole or in part, for their own private use, including for educational purposes, provided the material is not systematically reproduced or distributed and is not for sale.

Examples of permitted uses retained by authors or their employers include incorporating material into lectures and in-house training materials and presentations, and posting accepted manuscripts of conference papers and journal articles on a personal website or in an institutional or government archive. Refer to AIAA's Self-Archiving and Posting Policy, which addresses posting the accepted manuscript version on private websites and in institutional archives; for additional details on sharing your work before and after publication, also go to How Can I Share My Research? Links to the version of record (VOR) in AIAA's electronic library, **Aerospace Research Central (ARC)** (<https://arc.aiaa.org>), should be maintained, as appropriate.

In most cases when AIAA is the copyright holder of a work, authors will be automatically granted permission by AIAA to reprint their own material in subsequent works, to include

Data Mining of Text →

(click to show)

Text data mining of AIAA content available in **Aerospace Research Central (ARC)** (<https://arc.aiaa.org>) requires a bilateral agreement with AIAA. Terms of use for all content remain subject to copyright.

How to Request Permission to Reprint from AIAA ↓

(click to hide)

If you wish to reuse your own or someone else's material previously published by AIAA, in print or electronically, first determine whether or not AIAA is the copyright owner of the publication. Please review the copyright statement for the source material before submitting a reprint permission request:

- For AIAA conference papers, journal articles, or individual chapters in multi-authored books, look at the bottom of the first full-text page (not the cover page). There will be a footnote indicating who holds copyright.
- For single-author books, look at the copyright statement on the back of the title page.
- AIAA owns the copyright on all articles published in **Aerospace America** (<https://www.aiaa.org/publications/aerospace-america/>).

If the statement reads "Copyright © by 'the author ...' or by 'a university or other corporate entity ...'" then AIAA does not hold copyright, and you must seek permission to reprint from the copyright owner.

In the case of a U.S. government-sponsored work, where the work is "not subject to copyright protection in the United States," then the material is in the public domain and can be reused without permission *within the United States* so long as the original source is acknowledged and fully cited.

The **Copyright Clearance Center (CCC)** (<https://www.copyright.com>) processes permission requests on behalf of AIAA. If AIAA is the copyright owner, you may submit your request by visiting www.copyright.com (<http://www.copyright.com>).

<https://www.aiaa.org/publications/Publish-with-AIAARights-and-...> Page 3 of 6
<https://www.aiaa.org/publications/Publish-with-AIAARights-and-...> Page 5 of 6

Rights and Permissions | AIAA
20/024, 2:57 PM

figures, tables, and verbatim portions of text, upon request. Explicit permission should be sought from AIAA through **Copyright Clearance Center (CCC)** (<https://www.copyright.com>), as described below; all reprinted material must be acknowledged and the original source cited in full.

All versions of a publication are subject to the copyright terms and conditions executed between AIAA and the authors. AIAA has no copyright claim on material in the public domain or owned by individuals, institutions, or foreign governments.

Comprehensive End-User License →

(click to show)

All versions of AIAA content are subject to this end-user license, to include accepted manuscript versions and other open access publications. [Click here for more information](#) (publications/comprehensive-end-user-license#SO=true).

Funding Agency Mandates and Open Access →

(click to show)

Authors whose works are the result of research performed under a grant from a government funding agency are free to exercise all rights pertaining to public access as specified by the contract and to fulfill author deposit mandates from that funding agency provided that the mandate allows for a minimum 12-month embargo, starting from the official date of publication by AIAA, and so long as the accepted manuscript version, and not the AIAA published VOR is used for this purpose. Reproductions in whole or in part shall include a full citation in reference to the AIAA publication and notice of copyright.

As a member of **CHORUS** (<https://www.chorusaccess.org>), AIAA also will make available the accepted manuscript versions of journal articles that are subject to U.S. federal funding agency public access mandates, following the 12-month embargo period. These Open Access versions will be available through **Aerospace Research Central (ARC)** (<https://arc.aiaa.org>).

Sharing and reusing material from AIAA Open Access publications is subject to the terms of copyright.

When requesting to reuse material from AIAA conference papers, journal or magazine articles, or book chapters, be sure to search for the conference proceedings title (e.g., *Plasmasdynamics and Lasers Conference*), the journal title (e.g., *Journal of Aircraft*) (<https://arc.aiaa.org/journals/ja>) or book title, not the article/chapter title. If you are unable to find the appropriate publication, place a Special Order with CCC to work on your behalf to obtain permission.

Depending on who is making the request and the intended use, a modest reprinting fee may apply. Upon approval by AIAA to reprint, the author should acknowledge that AIAA has granted permission for reuse, and the original source should be fully cited in the author's reference list.

Any additional questions can be directed to Katrina Buckley at katb@aiaa.org (<mailto:katb@aiaa.org>).

(<https://www.copyright.com/CCDirect?publishername=AIAA&WT.mc.id=AIAA>)
Copyright Clearance Center (<https://marketplace.copyright.com/rs-ui-web/mpi/>)

American Institute of Aeronautics and Astronautics
 12700 Sunrise Valley Drive, Suite 200
 Reston, VA 20191-5807
 © 2024 American Institute of Aeronautics and Astronautics
 800-639-AIAA (2422)

<https://www.aiaa.org/publications/Publish-with-AIAARights-and-...> Page 4 of 6
<https://www.aiaa.org/publications/Publish-with-AIAARights-and-...> Page 6 of 6

A.3 UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning

4/4/24, 1:59 PM

Rightslink® by Copyright Clearance Center



Sign in/Register



UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning

Conference Proceedings:
2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Author: Mirco Theile
Publisher: IEEE
Date: 24 October 2020

Copyright © 2020, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis online.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK

CLOSE WINDOW

© 2024 Copyright - All Rights Reserved | Copyright Clearance Center, Inc. | [Privacy statement](#) | [Data Security and Privacy](#)
 | [For California Residents](#) | [Terms and Conditions](#) Comments? We would like to hear from you. E-mail us at customercare@copyright.com



<https://s100.copyright.com/AppDispatchServlet#formTop>

1/1

A.4 UAV Path Planning for Wireless Data Harvesting: A Deep Reinforcement Learning Approach

4/4/24, 2:00 PM

Rightslink® by Copyright Clearance Center



Sign in/Register



UAV Path Planning for Wireless Data Harvesting: A Deep Reinforcement Learning Approach

Conference Proceedings: GLOBECOM 2020 - 2020 IEEE Global Communications Conference

Author: Harald Bayerlein

Publisher: IEEE

Date: December 2020

Copyright © 2020, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis online.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK

CLOSE WINDOW

© 2024 Copyright - All Rights Reserved | Copyright Clearance Center, Inc. | Privacy statement | Data Security and Privacy | For California Residents | Terms and Conditions Comments? We would like to hear from you. E-mail us at customercare@copyright.com



<https://s100.copyright.com/AppDispatchServlet#formTop>

1/1

A.5 UAV Path Planning using Global and Local Map Information with Deep Reinforcement Learning

4/4/24, 1:58 PM

Rightslink® by Copyright Clearance Center



Sign in/Register



Requesting permission to reuse content from an IEEE publication

UAV Path Planning using Global and Local Map Information with Deep Reinforcement Learning

Conference Proceedings: 2021 20th International Conference on Advanced Robotics (ICAR)
Author: Mirco Theile
Publisher: IEEE
Date: 06 December 2021

Copyright © 2021, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis online.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK

CLOSE WINDOW

© 2024 Copyright - All Rights Reserved | Copyright Clearance Center, Inc. | Privacy statement | Data Security and Privacy | For California Residents | Terms and ConditionsComments? We would like to hear from you. E-mail us at customer-care@copyright.com



<https://s100.copyright.com/AppDispatchServlet#formTop>

1/1

A.6 Multi-UAV Path Planning for Wireless Data Harvesting with Deep Reinforcement Learning

The image shows a screenshot of the Creative Commons Attribution 4.0 International license page. The page is divided into several sections:

- Header:** CC BY 4.0 Deed | Attribution 4.0 International | Creative Commons. Date: 20/04, 4:44 PM.
- Navigation:** WHO WE ARE, WHAT WE DO, LICENSES AND TOOLS, BLOG, SUPPORT US.
- Logo:** CC BY 4.0 DEED Attribution 4.0 International.
- Legal Code:** See the legal code. Canonical URL: <https://creativecommons.org/licenses/by/4.0/>.
- You are free to:**
 - Share** — copy and redistribute the material in any medium or format for any purpose, even commercially.
 - Adapt** — remix, transform, and build upon the material for any purpose, even commercially.
- The licensor cannot revoke these freedoms as long as you follow the license terms.**
- Under the following terms:**
 - Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
 - No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- Footnotes:**
 - appropriate credit** — If supplied, you must provide the name of the creator and attribution partners, a copyright notice, a license notice, a disclaimer notice, and a link to the material. CC license prior to version 4.0 also require you to provide the title of the material if supplied, and may have other slight differences.
 - indicate if changes were made** — In 4.0, you must indicate if you modified the material and retain an indication of previous modifications. In 3.0 and earlier license versions, the indication of changes is only required if you create a derivative.
 - technological measures** — The license prohibits application of effective technological measures, defined with reference to Article 11 of the WIPO Copyright Treaty.
 - exception or limitation** — The rights of users under exceptions and limitations, such as fair use and fair dealing, are not affected by the CC license.
 - publicity, privacy, or moral rights** — You may need to get additional permissions before using the material as you intend.
- Contact Us:** Creative Commons PO Box 5869, Mountain View, CA 94042. Email: help@creativecommons.org.
- Subscribe to our Newsletter:** Input field and SUBSCRIBE button.
- Support our Work:** Our work relies on your help. Keep the Internet free and open. DONATE NOW button.
- Footer:** Page 1 of 3. URL: <https://creativecommons.org/licenses/by/4.0/deed.en>.

A.7 Cloud-Edge Training Architecture for Sim-to-Real Deep Reinforcement Learning

4/4/24, 1:58 PM

Rightslink® by Copyright Clearance Center



Sign in/Register



Cloud-Edge Training Architecture for Sim-to-Real Deep Reinforcement Learning

Conference Proceedings:
2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Author: Hongpeng Cao
Publisher: IEEE
Date: 23 October 2022

Copyright © 2022, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis online.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK

CLOSE WINDOW

© 2024 Copyright - All Rights Reserved | Copyright Clearance Center, Inc. | Privacy statement | Data Security and Privacy
| For California Residents | Terms and Conditions Comments? We would like to hear from you. E-mail us at
customercare@copyright.com



<https://s100.copyright.com/AppDispatchServlet#formTop>

1/1

A.8 Learning to Generate All Feasible Actions

The image shows a screenshot of the Creative Commons Attribution 4.0 International license deed page. The page is divided into two columns. The left column contains the main text of the deed, including the Creative Commons logo, the title "CC BY 4.0 DEED Attribution 4.0 International", and several sections: "You are free to:", "Under the following terms:", and "Notice". The right column contains a "Public Domain List" and "Footnotes" with detailed explanations of terms like "appropriate credit", "indicate if changes were made", "technological measures", "exception or limitation", and "publicity, privacy, or moral rights". At the bottom of the page, there is a "CONTACT US" section, a "SUBSCRIBE TO OUR NEWSLETTER" form, and a "DONATE NOW" button. The page number "Page 1 of 3" is visible at the bottom left, and "Page 2 of 3" is visible at the bottom right.


CC BY 4.0 Deed | Attribution 4.0 International | Creative Commons

30/NOV/2014, 4:44 PM

CC BY 4.0 Deed | Attribution 4.0 International | Creative Commons

30/NOV/2014, 4:44 PM

WHO WE ARE WHAT WE DO LICENSES AND TOOLS BLOG SUPPORT US


CC BY 4.0 DEED
Attribution 4.0 International

Ceasical URL: <https://creativecommons.org/licenses/by/4.0/> See the legal code

You are free to:

Share — copy and redistribute the material in any medium or format for any purpose, even commercially.

Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict

Footnotes

- appropriate credit** — If supplied, you must provide the name of the creator and attribution partners, a copyright notice, a license notice, a disclaimer notice, and a link to the material. CC license prior to version 4.0 also require you to provide the title of the material if supplied, and may have other slight differences.
 - More info
- indicate if changes were made** — In 4.0, you must indicate if you modified the material and retain an indication of previous modifications. In 3.0 and earlier license versions, the indication of changes is only required if you create a derivative.
 - Linking guide
 - More info
- technological measures** — The license prohibits application of effective technological measures, defined with reference to Article 11 of the WIPO Copyright Treaty.
 - More info
- exception or limitation** — The rights of users under exceptions and limitations, such as fair use and fair dealing, are not affected by the CC license.
 - More info
- publicity, privacy, or moral rights** — You may need to get additional permissions before using the material as you intend.
 - More info

Contact Newsletter Privacy Policies Terms

CONTACT US
Creative Commons PO Box 5865,
Mountain View, CA 94032
help@creativecommons.org
creativecommons.org

SUBSCRIBE TO OUR NEWSLETTER

SUBSCRIBE

SUPPORT OUR WORK
Our work relies on your help to keep the Internet free and open.
DONATE NOW

<https://creativecommons.org/licenses/by/4.0/deed.en> Page 1 of 3

<https://creativecommons.org/licenses/by/4.0/deed.en> Page 2 of 3

CC BY 4.0 Deed | Attribution 4.0 International | Creative Commons

30/NOV/2014, 4:44 PM

others from doing anything the license permits.

Notices:

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

Notice

This deed highlights only some of the key features and terms of the actual license. It is not a license and has no legal value. You should carefully review all of the terms and conditions of the actual license before using the licensed material.

Creative Commons is not a law firm and does not provide legal services. Distributing, displaying, or linking to this deed or the license that it summarizes does not create a lawyer-client or any other relationship.

Creative Commons is the nonprofit behind the open licenses and other legal tools that allow creators to share their work. Our legal tools are free to use.

- [Learn more about our work](#)
- [Learn more about CC Licensing](#)
- [Support our work](#)
- [Use the license for your own material.](#)
- [Licenses List](#)

<https://creativecommons.org/licenses/by/4.0/deed.en> Page 1 of 3

A.9 Latency-Aware Generation of Single-Rate DAGs from Multi-Rate Task Sets

4/4/24, 2:00 PM

Rightslink® by Copyright Clearance Center



Sign in/Register

Latency-Aware Generation of Single-Rate DAGs from Multi-Rate Task Sets

Conference Proceedings:
2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)

Author: Micaela Verucchi

Publisher: IEEE

Date: April 2020

Copyright © 2020, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis online.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK
CLOSE WINDOW

© 2024 Copyright - All Rights Reserved | Copyright Clearance Center, Inc. | [Privacy statement](#) | [Data Security and Privacy](#)
 | [For California Residents](#) | [Terms and Conditions](#) Comments? We would like to hear from you. E-mail us at customercare@copyright.com



https://s100.copyright.com/AppDispatchServlet#formTop

1/1

A.10 Edge Generation Scheduling for DAG Tasks Using Deep Reinforcement Learning

4/4/24, 1:54 PM

Rightslink® by Copyright Clearance Center



Sign in/Register



Edge Generation Scheduling for DAG Tasks Using Deep Reinforcement Learning

Author: Binqi Sun
Publication: IEEE Transactions on Computers
Publisher: IEEE
Date: April 2024

Copyright © 2024, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant.

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis online.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK

CLOSE WINDOW

© 2024 Copyright - All Rights Reserved | Copyright Clearance Center, Inc. | Privacy statement | Data Security and Privacy
| For California Residents | Terms and Conditions Comments? We would like to hear from you. E-mail us at
customercare@copyright.com

Privacy - Terms

https://s100.copyright.com/AppDispatchServlet#formTop

1/1

Bibliography

- [1] M. Theile, S. Yu, O. D. Dantsker, and M. Caccamo, “Trajectory estimation for geofencing applications on small-size fixed-wing UAVs,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1971–1977, IEEE, 2019.
- [2] M. Theile, O. Dantsker, R. Nai, M. Caccamo, and S. Yu, “uavAP: A modular autopilot framework for UAVs,” in *AIAA AVIATION 2020 FORUM*, p. 3268, 2020.
- [3] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, “UAV coverage path planning under varying power constraints using deep reinforcement learning,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1444–1449, IEEE, 2020.
- [4] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, “UAV path planning using global and local map information with deep reinforcement learning,” in *2021 20th International Conference on Advanced Robotics (ICAR)*, pp. 539–546, IEEE, 2021.
- [5] M. Theile, D. Bernardini, R. Trumpp, C. Piazza, M. Caccamo, and A. L. Sangiovanni-Vincentelli, “Learning to generate all feasible actions,” *IEEE Access*, vol. 12, pp. 40668–40681, 2024.
- [6] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, “Latency-aware generation of single-rate dags from multi-rate task sets,” in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 226–238, IEEE, 2020.
- [7] H. Bayerlein, M. Theile, M. Caccamo, and D. Gesbert, “UAV path planning for wireless data harvesting: A deep reinforcement learning approach,” in *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2020.
- [8] H. Bayerlein, M. Theile, M. Caccamo, and D. Gesbert, “Multi-UAV path planning for wireless data harvesting with deep reinforcement learning,” *IEEE Open Journal of the Communications Society*, vol. 2, pp. 1171–1187, 2021.
- [9] H. Cao, M. Theile, F. G. Wyrwal, and M. Caccamo, “Cloud-edge training architecture for sim-to-real deep reinforcement learning,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9363–9370, IEEE, 2022.

-
- [10] B. Sun, M. Theile, Z. Qin, D. Bernardini, D. Roy, A. Bastoni, and M. Caccamo, "Edge generation scheduling for dag tasks using deep reinforcement learning," *IEEE Transactions on Computers*, 2024.
- [11] K.-D. Kim and P. R. Kumar, "Cyber-physical systems: A perspective at the centennial," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1287–1308, 2012.
- [12] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.
- [13] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects: Steven m. lavalley, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan," *Algorithmic and computational robotics*, pp. 303–307, 2001.
- [14] R. Geraerts and M. H. Overmars, "A comparative study of probabilistic roadmap planners," in *Algorithmic foundations of robotics V*, pp. 43–57, Springer, 2004.
- [15] T. Wescott, "Pid without a phd," *Embedded Systems Programming*, vol. 13, no. 11, pp. 1–7, 2000.
- [16] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.
- [17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.
- [18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [19] T. Wu, S. He, J. Liu, S. Sun, K. Liu, Q.-L. Han, and Y. Tang, "A brief overview of chatgpt: The history, status quo and potential future development," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 5, pp. 1122–1136, 2023.
- [20] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [21] E. Salvato, G. Fenu, E. Medvet, and F. A. Pellegrino, "Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning," *IEEE Access*, vol. 9, pp. 153171–153187, 2021.
- [22] O. Dantsker, "The future of agriculture, powered by the sun," 2022.
- [23] O. D. Dantsker, M. Theile, M. Caccamo, S. Yu, M. Vahora, and R. Mancuso, "Continued development and flight testing of a long-endurance solar-powered unmanned aircraft: Uiuc-tum solar flyer," in *AIAA Scitech 2020 Forum*, p. 0781, 2020.

- [24] Precision Hawk, “Precision Agriculture, Commercial UAV and Farm Drones For Sale.” <http://precisionhawk.com/>, Accessed Dec. 2023.
- [25] MicroPilot, “MicroPilot - MP-Vision.” <https://www.micropilot.com/>, Accessed Dec. 2023.
- [26] Pix4D SA, “Pix4D.” <https://www.pix4d.com/>, Accessed Dec. 2023.
- [27] Reconstruct Inc., “Reconstruct.” <https://www.reconstructinc.com/>, Accessed Dec. 2023.
- [28] Silent Falcon UAS Technologies, “Silent Falcon.” <https://silentfalconuas.com/>, Accessed May. 2023.
- [29] AeroVironment, “Aerovironment solar-powered puma ae small unmanned aircraft achieves continuous flight for more than nine hours.” <https://www.avinc.com/resources/press-releases/view/aerovironment-solar-powered-puma-ae-small-unmanned-aircraft-achieves-contin>, Accessed Dec. 2023.
- [30] BAE Systems, “Autonomous Real-Time Ground Ubiquitous Surveillance Imaging System (ARGUS-IS).” <https://www.baesystems.com/en/product/autonomous-realtime-ground-ubiquitous-surveillance-imaging-system-argusis>, Accessed Dec. 2023.
- [31] SZ DJI Technology Co., Ltd., “DJI.” <https://www.dji.com/>, Accessed Dec. 2023.
- [32] Shenzhen Hubsan Technology Co., Ltd., “Hubsan.” <https://www.hubsan.com/>, Accessed Dec. 2023.
- [33] A. Noth, *Design of Solar Powered Airplanes for Continuous Flight*. PhD thesis, ETH Zurich, 2008.
- [34] Oettershagen, P. et al., “Design of small hand-launched solar-powered uavs: From concept study to a multi-day world endurance record flight,” *Journal of Field Robotict*, vol. 34, p. 1352–1377, 2017.
- [35] ETH Zurich, Autonomous Systems Lab, “Atlantik-Solar.” <https://www.atlantiksolar.ethz.ch/>, Accessed Dec. 2023.
- [36] I.-Y. Ahn, J.-S. Bae, S. Park, and Y.-M. Yang, “Development and flight test of a small solar powered uav,” vol. 41, 11 2013.
- [37] A. Weider, H. Levy, I. Regev, L. Ankri, T. Goldenberg, Y. Ehrlich, A. Vladimirsky, Z. Yosef, and M. Cohen, “Sunsailor: solar powered uav,” *Technion IIT, Haifa, Israel*, vol. 6, 2006.
- [38] S. Morton, R. D’Sa, and N. Papanikolopoulos, “Solar powered uav: Design and experiments,” *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2460–2466, 2015.

-
- [39] N. J. P. Betancourth and et al., “Design and Manufacture of a Solar-Powered Unmanned Aerial Vehicle for Civilian Surveillance Missions,” *Journal of Aerospace Technology and Management*, vol. 8, pp. 385 – 396, 12 2016.
- [40] O. D. Dantsker, M. Theile, and M. Caccamo, “A cyber-physical prototyping and testing framework to enable the rapid development of UAVs,” *Aerospace*, vol. 9, no. 5, p. 270, 2022.
- [41] O. D. Dantsker, M. Theile, M. Caccamo, and R. Mancuso, “Design, development, and initial testing of a computationally-intensive, long-endurance solar-powered unmanned aircraft,” in *2018 Applied Aerodynamics Conference*, p. 4217, 2018.
- [42] O. Dantsker, M. Caccamo, and S. Imtiaz, “Propulsion system design, optimization, simulation, and testing for a long-endurance solar-powered unmanned aircraft,” in *AIAA Propulsion and Energy 2020 Forum*, p. 3966, 2020.
- [43] Al Volo LLC, “Al Volo: Flight Systems.” <http://www.alvolo.us>, Accessed Jan. 2024.
- [44] O. D. Dantsker, M. Caccamo, and R. Mancuso, “Energy system instrumentation and data acquisition for flight testing a long-endurance, solar-powered unmanned aircraft,” in *AIAA Propulsion and Energy 2021 Forum*, p. 3721, 2021.
- [45] O. D. Dantsker, S. Yu, M. Vahora, and M. Caccamo, “Flight testing automation to parameterize unmanned aircraft dynamics,” in *AIAA Aviation 2019 Forum*, p. 3230, 2019.
- [46] M. Theile, O. D. Dantsker, R. Nai, and M. Caccamo, “uavEE: A modular, power-aware emulation environment for rapid prototyping and testing of UAVs,” in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 217–224, IEEE, 2018.
- [47] Laminar Research, “X-Plane 11.” <https://www.x-plane.com/>, Accessed Jan. 2024.
- [48] O. D. Dantsker, M. Theile, and M. Caccamo, “A high-fidelity, low-order propulsion power model for fixed-wing electric unmanned aircraft,” in *2018 AIAA/IEEE Electric Aircraft Technologies Symposium (EATS)*, pp. 1–16, IEEE, 2018.
- [49] O. D. Dantsker, M. Theile, and M. Caccamo, “Integrated power modeling for a solar-powered, computationally-intensive unmanned aircraft,” in *2020 AIAA/IEEE Electric Aircraft Technologies Symposium (EATS)*, pp. 1–21, IEEE, 2020.
- [50] O. D. Dantsker, M. Theile, M. Caccamo, and S. Hong, “Integrated power simulation for a solar-powered, computationally-intensive unmanned aircraft,” in *AIAA Propulsion and Energy 2021 Forum*, p. 3317, 2021.
- [51] InertiaSoft, Inc, “FS One RC Flight Simulator.” <https://www.fsone.com/>, Accessed Jan. 2024.

- [52] O. D. Dantsker, M. Theile, and M. Caccamo, “Long endurance flight testing results for the uiuc-tum solar flyer,” in *AIAA AVIATION 2021 FORUM*, p. 3196, 2021.
- [53] S. D. Alwis, Z. Hou, Y. Zhang, M. H. Na, B. Ofoghi, and A. Sajjanhar, “A survey on smart farming data, applications and techniques,” *Computers in Industry*, vol. 138, p. 103624, June 2022.
- [54] X. Jin, L. Kumar, Z. Li, H. Feng, X. Xu, G. Yang, and J. Wang, “A review of data assimilation of remote sensing and crop models,” *European Journal of Agronomy*, vol. 92, pp. 141–152, Jan. 2018.
- [55] J. Kim, S. Kim, C. Ju, and H. I. Son, “Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications,” *IEEE Access*, vol. 7, pp. 105100–105115, 2019.
- [56] A. I. de Castro, Y. Shi, J. M. Maja, and J. M. Peña, “UAVs for vegetation monitoring: Overview and recent scientific contributions,” *Remote Sensing*, vol. 13, p. 2139, May 2021.
- [57] M. F. Aslan, A. Durdu, K. Sabanci, E. Ropelewska, and S. S. Gültekin, “A comprehensive survey of the recent studies with UAV for precision agriculture in open fields and greenhouses,” *Applied Sciences*, vol. 12, p. 1047, Jan. 2022.
- [58] A. Sassu, J. Motta, A. Deidda, L. Ghiani, A. Carlevaro, G. Garibotto, and F. Gambella, “Artichoke deep learning detection network for site-specific agrochemicals UAS spraying,” *Computers and Electronics in Agriculture*, vol. 213, p. 108185, Oct. 2023.
- [59] C. Cavalaris, C. Karamoutis, and A. Markinos, “Efficacy of cotton harvest aids applications with unmanned aerial vehicles (UAV) and ground-based field sprayers – a case study comparison,” *Smart Agricultural Technology*, vol. 2, p. 100047, Dec. 2022.
- [60] E. Galceran and M. Carreras, “A survey on coverage path planning for robotics,” *Robotics and Autonomous systems*, vol. 61, no. 12, pp. 1258–1276, 2013.
- [61] R. Bellman, “A markovian decision process,” *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.
- [62] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, pp. 279–292, 1992.
- [63] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [64] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [65] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

-
- [66] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [67] M. G. Bellemare, W. Dabney, and R. Munos, “A distributional perspective on reinforcement learning,” in *International conference on machine learning*, pp. 449–458, PMLR, 2017.
- [68] V. Konda and J. Tsitsiklis, “Actor-critic algorithms,” *Advances in neural information processing systems*, vol. 12, 1999.
- [69] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [70] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.
- [71] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
- [72] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in *International conference on machine learning*, pp. 1587–1596, PMLR, 2018.
- [73] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.
- [74] “Softlearning.” <https://github.com/rail-berkeley/softlearning>. Accessed: January 15, 2024.
- [75] “Soft Actor-Critic.” <https://spinningup.openai.com/en/latest/algorithms/sac.html>. Accessed: January 15, 2024.
- [76] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [77] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” *arXiv preprint arXiv:1506.02438*, 2015.
- [78] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

- [79] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, “Mastering chess and shogi by self-play with a general reinforcement learning algorithm,” *arXiv preprint arXiv:1712.01815*, 2017.
- [80] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, “Dota 2 with large scale deep reinforcement learning,” *arXiv preprint arXiv:1912.06680*, 2019.
- [81] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, “Grandmaster level in starcraft ii using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.
- [82] W. Koch, R. Mancuso, R. West, and A. Bestavros, “Reinforcement learning for uav attitude control,” *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, pp. 1–21, 2019.
- [83] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, “Learning dexterous in-hand manipulation,” *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.
- [84] L. Smith, I. Kostrikov, and S. Levine, “A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning,” *arXiv preprint arXiv:2208.07860*, 2022.
- [85] A. Maslow, *The Psychology of Science: A Reconnaissance*. Gateway edition, Harper & Row, 1966.
- [86] M. Theile, H. Bayerlein, M. Caccamo, and A. L. Sangiovanni-Vincentelli, “Learning to recharge: UAV coverage path planning through deep reinforcement learning,” *arXiv preprint arXiv:2309.03157*, 2023.
- [87] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, “Imitation learning: A survey of learning methods,” *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.
- [88] S. Levine, A. Kumar, G. Tucker, and J. Fu, “Offline reinforcement learning: Tutorial, review, and perspectives on open problems,” *arXiv preprint arXiv:2005.01643*, 2020.
- [89] M. Theile, H. Cao, M. Caccamo, and A. L. Sangiovanni-Vincentelli, “Equivariant ensembles and regularization for reinforcement learning in map-based path planning,” *arXiv preprint arXiv:2403.12856*, submitted to 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2024.