TUM School of Computation, Information, and Technology

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Dissertation in Informatik

Testing the Safe Behavior of Unmanned Aerial Vehicles with Scenario-Based Testing

Tabea Ruth Jasmin Schmidt

Technische Universität München
TUM School of Computation, Information, and Technology

Testing the Safe Behavior of Unmanned Aerial Vehicles with Scenario-Based Testing

Tabea Ruth Jasmin Schmidt

Vollständiger Abdruck der von der TUM School of Computation, Information, and Technology der Technischen Universität München zur Erlangung des akademischen Grades einer

Doktorin der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz:		Prof. Dr.-Ing. Jörg Ott
Prüfer*innen der Dissertation:		
	1.	Prof. Dr. Alexander Pretschner
	2.	Prof. Dr. Stefan Leutenegger

Die Dissertation wurde am 22.12.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information, and Technology am 30.05.2023 angenommen.

Acknowledgments

First, I would like to thank my supervisor, Prof. Dr. Alexander Pretschner, for always asking critical questions that have advanced my research. By raising new research questions and providing honest feedback, he inspired me to investigate various aspects of my research topic and think outside the box. With his high expectations for all of us, he pushed me to do my best and to not only present and discuss my research results but also refine their underlying methodology. Under his guidance, I have improved my preciseness in communicating my findings and methodological understanding. In addition, I was able to enhance my social skills, such as presenting my work or teaching others.

Next, my gratitude goes to my colleagues at the chair for providing me with such a constructive, pleasant, and enjoyable work experience. Since everyone was always open to discussion, new research ideas quickly introduced themselves. Further, I really enjoyed the entertaining evenings during Hütte and other events.

Finally, the utmost thanks belong to my family and friends for their continuous support. They have given me excellent support by celebrating the milestones I have achieved and encouraging me to face challenges instead of giving up easily.

Zusammenfassung

Damit unbemannte Luftfahrzeuge beispielsweise Pakete direkt an unsere Haustüre liefern können, wird sich deren primärer Betriebsmodus bald auf den autonomen Betrieb außerhalb unserer Sichtweite verlagern. Daher benötigen wir eine Methodik, die das sichere Verhalten von unbemannten Luftfahrzeugen in verschiedenen Situationen systematisch testet und ein sicheres Verhalten dieser autonomen Systeme auch unter schwierigsten Bedingungen gewährleistet. In dieser Arbeit stellen wir eine solche Methodik vor, die szenario-basiertes Testen verwendet. Wir konzentrieren uns auf zwei Probleme: (1) die Ableitung relevanter Testsituationen und (2) die Generierung von Testfällen, die potenzielle Fehler im zu testenden System aufdecken können.

Hinsichtlich des ersten Problems erörtern wir die offenen Forschungsfragen bei der automatischen Ableitung von Testsituationen basierend auf gesammelten realen Flugdaten. Darüber hinaus stellen wir eine systematische Methodik zum Aufbau einer Ontologie vor, die Testsituationen auf der Grundlage von mentalen Modellen beschreibt, sowie die daraus resultierende Ontologie für einen Quadrocopter, der eine Art von unbemannten Luftfahrzeugen darstellt. Schließlich stellen wir einen automatisierten Ansatz zur experimentellen Ermittlung von Grenzwerten für die Dimensionen der Ontologie am Beispiel der Ermittlung einer maximalen Anzahl von zu berücksichtigenden Hindernissen dar. Unsere experimentellen Ergebnisse zeigen eine maximale Anzahl von 5 oder 8 relevanten Hindernissen für das zu testende System, abhängig von der für die Datenerfassung verwendeten Fehlerhypothese. Mit sinnvollen Grenzwerten für die Dimensionen der Ontologie können wir die Anzahl der Testsituationen effektiv begrenzen. Diese Grenzen stellen somit eine Grundlage für die Sammlung einer vollständigen Liste relevanter Testsituationen für unbemannte Luftfahrzeuge in zukünftigen Arbeiten dar.

Um das zweite Problem anzugehen, stellen wir eine Methode zum Testen des sicheren Verhaltens von unbemannten Luftfahrzeugen vor, in der wir deren Umgebung einbeziehen und berücksichtigen, dass wir das sichere Verhalten von unbemannten Luftfahrzeugen nicht in allen Situationen explizit definieren können. In unseren Experimenten entdecken wir mit dieser Methodik verschiedene Verletzungen des Sicherheitsabstands und mehrere fragwürdige Verhaltensweisen, wenn wir aufgrund fehlender Spezifikationen oder Vorschriften keinen Sicherheitsabstand definieren können. Diese Ergebnisse zeigen die Anwendbarkeit und Effektivität der vorgestellten Methodik zur Erstellung von Testfällen, die potenzielle Fehler im zu testenden System aufdecken können. Da in der aktuellen Literatur heuristische Optimierungsalgorithmen verwendet werden, um diese Testfälle zu finden, müssen wir die Qualität der generierten Testfälle bewerten. In dieser Arbeit stellen wir eine solche Fallstudie über drei Optimierungsalgorithmen und deren Kombinationen vor, die das Problem aufzeigt, dass selbst der beste Algorithmus mehrere fehlerproduzierende Situationen übersieht. Diese Ergebnisse deuten darauf hin, dass wir bei szenario-basiertem

Testen zusätzlich mehrere Optimierungsalgorithmen verwenden müssen, was den weitverbreiteten Einsatz dieser Technik zum Testen von autonomen Systemen herausfordert. Schließlich stellen wir das Tool StellaUAV vor, das die vorgestellte Methodik systematisch anwendet, und zeigen seine Anwendbarkeit und Effektivität in Experimenten. Diese zeigen auf, dass das zu testende System Probleme hat, sich in der Nähe von Hindernissen, die sich dynamisch bewegen, sicher zu verhalten.

Abstract

To enable use cases such as package delivery to our doorsteps, the primary operation mode of Unmanned Aerial Vehicles (UAVs) will shift to autonomous operation near structures and people soon. If UAVs operate autonomously and Beyond Visual Line of Sight, it is crucial to ensure that these systems behave safely. Thus, we need a methodology that systematically tests the safe behavior of UAVs in various situations and ensures that the UAVs operate safely even in the most challenging circumstances. In this work, we present such a methodology that uses scenario-based testing to evaluate the safe behavior of UAVs. We focus on two problems: (1) the derivation of relevant situations to test and (2) the generation of test cases that can reveal potential faults in the system under test.

Addressing the first problem, we discuss the open research challenges of acquiring test situations automatically by clustering collected real-flight data. In addition, we present the systematic methodology for building an ontology that characterizes these situations based on mental models and the resulting ontology for a quadcopter as one kind of UAV. Finally, we demonstrate an automated approach for experimentally finding lower and upper bounds for the dimensions of the ontology using the example of finding a maximal number of obstacles to consider. Our experimental results show a maximal number of 5 or 8 relevant obstacles for the System Under Test (SUT) with one of the optimization algorithms depending on the applied defect hypothesis used for data collection. With reasonable bounds for the ontology's dimensions, we can effectively limit the number of situations to test the SUT. Thus, these bounds further present a basis for collecting a complete list of relevant test situations for UAVs in future work.

Addressing the second problem, we present a methodology for testing the safe behavior of UAVs that considers their environment and acknowledges the challenge of explicitly defining the safe behavior of UAVs in all situations. In our experiments, we detect various safety distance violations with the presented methodology and several questionable behaviors when we cannot define a safety distance due to missing specifications or regulations. These experimental results show the applicability and effectiveness of the presented methodology for creating test cases that reveal potential faults in the SUT. As current literature suggests using heuristic optimization algorithms for finding these test cases, we need to evaluate the quality of the generated test cases. In this work, we present an assessment of the performance of three optimization algorithms and their combinations that reveals the problem that even the best-performing algorithm misses several challenging situations that can uncover potential faults in the SUT. These results indicate that scenario-based testing comes at the extra cost of having to run multiple optimization algorithms, which challenges the widespread use of this technique for testing autonomous systems. Finally, we introduce the tool StellaUAV, which systematically applies the presented methodology, and show its applicability and effectiveness in experiments by revealing the SUT's problems to behave safely when encountering moving obstacles.

Outline of the Thesis

CHAPTER 1: INTRODUCTION

This chapter introduces the topic of testing the safe behavior of Unmanned Aerial Vehicles (UAVs) with scenario-based testing and describes the gaps in the literature that this thesis aims to close by providing a methodology for generating test cases for UAVs. Parts of this chapter previously appeared in peer-reviewed publications [114, 115, 116] co-authored by the author of this thesis.

CHAPTER 2: BACKGROUND AND PRELIMINARIES

This chapter provides a general overview of autonomously operating UAVs and the concepts of scenario-based testing and search-based techniques for finding worst-case situations to test the safe behavior of UAVs. Parts of this chapter previously appeared in peer-reviewed publications [114, 115, 116] co-authored by the author of this thesis.

CHAPTER 3: METHODS AND CHALLENGES OF DERIVING LOGICAL SCENARIOS FOR UAVS

This chapter outlines one of the fundamental challenges of scenario-based testing: the definition of logical scenarios to test the UAVs' safe behavior. First, it discusses the challenges of automatically deriving logical scenarios from collected flight data before providing an ontology to characterize them based on mental models for a quadcopter as one kind of UAV. Parts of this chapter previously appeared in a peer-reviewed submission under review [118] co-authored by the author of this thesis.

CHAPTER 4: EXPLORATION OF BOUNDS FOR THE ONTOLOGY'S DIMENSIONS

This chapter presents an automated approach for finding reasonable bounds for the parameter values that each dimension of an ontology for logical scenarios for UAVs describes. Further, it illustrates the approach with the example of exploring an upper bound for the number of relevant obstacles to include in logical scenarios. Parts of this chapter previously appeared in a peer-reviewed publication [115] co-authored by the author of this thesis.

CHAPTER 5: UNDERSTANDING AND ASSESSMENT OF THE SAFE BEHAVIOR OF UAVS

This chapter provides a methodology for testing the safe behavior of UAVs while considering their environment and the potential challenge of explicitly defining the safe behavior of UAVs. Throughout the chapter, we explore the two cases of having a safety distance specified and working with no defined safety distance. Parts of this chapter previously appeared in a peer-reviewed publication [114] co-authored by the author of this thesis.

CHAPTER 6: EVALUATION OF OPTIMIZATION ALGORITHMS FOR TESTING THE SAFE BEHAVIOR OF UAVS

This chapter introduces the problem of a missing guarantee for finding worst-case situations with heuristic optimization algorithms. Further, it presents a case study to explore the quality of generated test cases for three optimization algorithms and their sequential combinations when testing the safe behavior of an open-source UAV. Parts of this chapter previously appeared in a peer-reviewed publication [116] co-authored by the author of this thesis.

CHAPTER 7: STELLAUAV: A TOOL FOR TESTING THE SAFE BEHAVIOR OF UAVS

This chapter presents the tool StellaUAV that implements the proposed approach of testing the safe behavior of UAVs with scenario-based testing and an evaluation of its applicability and effectiveness for generating test cases that can reveal potential faults in UAVs. Parts of this chapter previously appeared in a peer-reviewed publication [116] co-authored by the author of this thesis.

CHAPTER 8: RELATED WORK

This chapter discusses related work about testing the safe behavior of UAVs with its challenges of deriving relevant logical scenarios for these systems and generating worst-case situations for them. Parts of this chapter previously appeared in peer-reviewed publications [114, 115, 116] and a peer-reviewed submission under review [118] co-authored by the author of this thesis.

CHAPTER 9: CONCLUSION AND OUTLOOK

This chapter concludes by summarizing the concepts and results for testing the safe behavior of UAVs presented in this thesis. In addition, it includes limitations of our work, lessons learned, and ideas for future work.

N.B.: *Multiple chapters of this dissertation are based on different publications authored or co-authored by the author of this dissertation. Such publications are mentioned in the short descriptions above. Due to the obvious content overlapping, quotes from such publications within the respective chapters are not marked explicitly.*

Contents

Part I.

Introduction and Background

1. Introduction

This chapter introduces the topic of testing the safe behavior of Unmanned Aerial Vehicles (UAVs) with scenario-based testing and describes the gaps in the literature that this thesis aims to close by providing a methodology for generating test cases for UAVs. Parts of this chapter previously appeared in peer-reviewed publications [114, 115, 116] co-authored by the author of this thesis.

1.1. Testing the Safe Behavior of Unmanned Aerial Vehicles

Unmanned Aerial Vehicles (UAVs) conquer more and more air space to fulfill various tasks, e.g., transporting objects, monitoring areas, search & rescue, and more specific tasks such as wildfire fighting, precision farming, or weather forecasting [14, 25, 34, 78, 110, 120, 132]. Examples from the industry include Amazon [123], which is actively working on autonomously operating UAVs that deliver packages to our doorsteps, and Zipline [105], which has transported medicine in Rwanda via UAVs since 2016.

Currently, we perform most of these use cases by remotely controlling UAVs that are not allowed to fly closer than 30 to 50 meters to any person or structure, depending on our home country [8, 21, 35]. However, the primary operation mode of UAVs will shift to an autonomous operation soon. Further, we believe that we will be allowed to operate UAVs Beyond Visual Line of Sight and nearer to structures and people to enable use cases such as package delivery to our houses in the near future. These new operation modes will lead to a significant increase in the use and development of autonomously operating UAVs. If UAVs operate autonomously and Beyond Visual Line of Sight, ensuring that these systems behave safely and do not pose any unreasonable risk to their environment is crucial. This effort is especially needed as there is no human in the loop to correct any unsafe behavior in these use cases. Thus, we need a methodology that systematically tests the safe behavior of UAVs in various situations and ensures that the UAVs operate safely even in the most challenging circumstances.

When developing aerial vehicles, we have to meet the requirements of various standards to get approved by certification authorities, such as EASA and FAA. Depending on the type of UAV and its intended use case, different standards might be relevant, such as DO-178C, DO-254, ARP4754, and ISO/TC 20/SC 16. Several of these standards apply to manned as well as unmanned aerial vehicles, whereas others are for unmanned aerial vehicles in particular. These standards specify requirements, processes, and metrics that we need

to perform or fulfill for certifying aerial vehicles, focusing on different areas, such as the software life cycle, hardware life cycle, system life cycle, or operation of airborne systems. The standards' goal is to minimize the number of failures by defining processes that engineers need to execute consistently and efficiently. Due to these process specifications, the standards provide a first step towards developing correctly and safely behaving UAVs. However, since their focus lies on the specific parts of the development process or operation of UAVs, the standards do not provide a thorough safety argumentation about a UAV's safe behavior in all relevant situations. To further assess the safety of UAVs, we propose to apply additional methods.

For the automotive industry, the authors of [133] show that it is unfeasible to ensure the safety of autonomous cars by executing real-world test drives. They present the need to drive 6.6 billion kilometers for each variant of the System Under Test (SUT) to ensure that it is at least as safe as with a human driver. The authors base these computations on the number of kilometers between two fatal accidents. Since the number of kilometers between two fatal accidents for aerial vehicles is even higher than the one for cars, this approach is likewise unfeasible for UAVs. Thus, [54] proposes a shift to simulation-based testing to enable thorough testing of such autonomous systems. In addition, simulation-based techniques, contrary to real-world flight tests, provide the advantage of not causing any damage or injury when the SUT behaves unsafely and collides with persons or objects in its environment. Since we are interested in testing UAVs in safety-critical situations, simulation-based techniques are well suited for our approach. When using simulation-based testing, we rely on the correctness of the simulation, which presents an abstraction of the SUT and its environment. In literature, there exists a large amount of research about the correctness of simulations [58, 108]. Due to this high effort in verifying and improving the quality of simulations in the research community, we will use existing tools for simulating UAVs and do not incorporate research about further improvements of simulations into this thesis. Instead of concentrating on the verification of the simulation itself, we focus on building a methodology for generating test cases for testing the safe behavior of UAVs with the help of these simulations.

When testing the safe behavior of Automated and Autonomous Driving Systems (ADS), scenario-based testing [23] provides valuable insights into the safe behavior of these systems [48, 82, 89, 126]. Thus, we propose to apply the same approach for testing the safe operation of UAVs. The general idea of scenario-based testing is that we aim to ensure that the SUT behaves safely in all relevant and challenging situations that it might encounter in the real world. Therefore, we first acquire typical situations in which we need to test the behavior of the SUT. An example of such a typical situation for a UAV is that the UAV has the mission to fly to a target point while avoiding two static obstacles and two dynamically moving obstacles. We can further specify the environmental conditions in this scenario by, e.g., describing the existence of light precipitation, cold ambient temperature, cloud coverage, and heavy fog. In the next step, we generate test cases for each of the derived situations that can reveal potential faults in the SUT.

1.1.1. Derivation of Typical Situations

For testing the safe behavior of UAVs with scenario-based testing, one of the fundamental challenges is the derivation of all relevant and challenging situations. For ADS, we have a good understanding of these typical situations and can define their boundaries with the help of rigid road structures and fine-grained traffic rules. In contrast, UAVs operate in an open field, on various missions, and in different environments with only coarse-grained restrictions. When acquiring typical situations for UAVs, we need to take all these challenging circumstances into account. First, we need to specify relevant dimensions for describing these typical situations for testing the safe behavior of UAVs. Secondly, we need to find the correct level of granularity to present challenging situations for the SUT. We propose representing this knowledge in an ontology that characterizes typical situations for testing UAVs.

Authors of related work [36, 72, 110] focus on describing use case scenarios for UAVs, which mainly focus on the mission of the included UAVs and provide a limited perspective on the environment of the UAVs. As an extension of their work, we aim to present an ontology that describes typical situations for testing the safe behavior of UAVs in this thesis that includes information about the UAVs, their missions, and an in-depth specification of their environment. For ADS, [10, 90] present models of typical situations that describe the primary entities and their relationships without denoting concrete dimensions of these typical situations. In addition, we cannot easily compare challenging situations for ADS and UAVs due to rigid road structures and traffic rules for ADS and the non-existence of these for UAVs.

To close these gaps in the literature, we aim to present an ontology that characterizes typical situations for testing the safe behavior of a quadcopter as one kind of UAV in this thesis. Further, to enable test engineers to replicate such an ontology for their SUTs, we provide the applied methodology for systematically creating the resulting ontology. Finally, we discuss the assumptions and conditions needed for providing a complete list of acquired typical situations with the presented methodology.

1.1.2. Generation of Test Cases

After collecting a list of typical situations to test UAVs, we aim to generate test cases for each of them. Instead of randomly creating test cases, we propose to search for challenging situations for the UAVs in each of these typical situations to reveal potential faults in the tested systems. We call these challenging situations also worst-case situations. When evaluating the safe behavior of the SUT in these test cases, we face the challenge of explicitly defining its safe behavior. For ADS, fine-grained traffic rules ease the specification of this safe behavior. However, since we are missing such detailed regulations for UAVs, we need to consider that we might not be able to explicitly define the safe behavior of UAVs in each situation. Thus, a methodology for generating worst-case situations for UAVs needs to take these circumstances into account.

For finding worst-case situations, related work proposes search-based techniques [48, 137, 146, 147]. Due to their heuristic nature, these techniques cannot guarantee that they will detect the optimal solution for a search problem. Thus, we need to evaluate the quality of the created test cases for various optimization algorithms to enable a thorough safety argumentation. In addition, we cannot directly apply the concepts presented in related work about the generation of worst-case situations in the automotive domain [48, 137] since worst-case situations for UAVs look differently than those for ADS. On the other side, the authors of [146, 147] focus on finding collisions between various UAVs in an otherwise empty environment. Since the safe behavior of UAVs in urban areas will be a crucial part of their deployment, we believe that we need to take the environment into account when searching for challenging situations for UAVs. In addition, all four papers concentrate on the case of a specified safety distance to define the safe behavior of the autonomous vehicle or UAV and do not explore the event when we cannot determine such a safety distance. Related work presents various case studies that evaluate the performance of optimization algorithms in different domains [29, 64, 125]. However, to the best of our knowledge, no work on assessing optimization algorithms for scenario-based testing for UAVs exists. Since each optimization algorithm has its advantages and disadvantages, its performance is highly context-specific and cannot be easily generalized over several domains. The authors of [60] focus on investigating the convergence rate of the evaluated optimization algorithms when generating test cases for ADS. Contrary to their work, we aim to assess the performance of optimization algorithms considering the quality of the created worst-case situations.

To close the presented gaps in the literature, we propose a methodology for generating test cases for testing the safe behavior of UAVs in urban areas that present worst-case situations for the SUT in this thesis. In addition, we consider the two cases of (1) having a pre-defined safety distance to lead the search for worst-case situations and (2) creating worst-case situations without such a specification. To assess the quality of the generated test cases, we provide a case study that evaluates the performance of three optimization algorithms and their combinations.

1.2. Problem Statement and Research Gaps

In this work, we present solutions to two fundamental problems that we face when testing the safe behavior of UAVs with scenario-based testing. For each of these problems, we further describe the gaps in the literature.

- **Problem 1** is the derivation of situations for testing the safe behavior of UAVs. For a thorough safety argumentation, these acquired situations should represent all relevant occasions in which we aim to ensure the safe behavior of the SUT. Currently, it is unclear which dimensions are essential for characterizing these situations for UAVs. Further, we cannot apply data-driven methods from the automotive domain [47] to derive additional test situations since we still lack high amounts of data about

real-world UAV flights to use these techniques. To limit the number of test situations for the SUT, we need to find reasonable lower and upper bounds for each dimension that characterizes relevant situations for the SUT. Finally, since it is infeasible to test all combinations of the collected dimensions, we need to select relevant and suitable ones for the SUT.

  - **Gap 1**: Related work on typical situations for UAVs either focuses on the mission of the UAVs or includes a limited number of environmental effects. Thus, there is a need for an in-depth description of the relevant dimensions of typical situations for testing UAVs. Existing work on typical situations for ADS presents meta-models of these situations but lacks concrete dimensions for a direct derivation of concrete situations. Further, different situations are challenging for ADS and UAVs due to rigid road structures and traffic rules for ADS. Thus, there exists the need for an ontology with concrete dimensions to describe relevant situations for UAVs.

- **Problem 2** is the generation of test cases for testing the safe behavior of UAVs that can reveal potential faults in the SUT. Since we cannot easily define the safe behavior of UAVs in all situations, we need to acknowledge this challenge when building a methodology for generating test cases for these systems. When creating a thorough safety argumentation, we need to be sufficiently confident in the quality of the found worst-case situations. Thus, we need to perform an evaluation of various optimization algorithms that can discover these worst-case situations. In addition, since combining different optimization algorithms might present further improvements, we need to investigate their performance concerning the quality of the generated test cases.

  - **Gap 2**: Since different situations are challenging for UAVs and ADS, we cannot directly utilize related work on generating test cases for ADS. Further, the non-existence of fine-grained traffic rules for UAVs increases the challenge of clearly defining a safe behavior for UAVs. Existing work on test case generation for UAVs focuses on testing against a specified safety distance and in an environment with no obstacles and environmental effects. For evaluating the performance of various optimization algorithms for generating worst-case situations for UAVs, we cannot easily generalize the results from other domains since they are highly context-specific. Existing work in the context of ADS focuses on the convergence rate of various optimization algorithms instead of investigating the quality of the created test cases. To the best of our knowledge, no work on assessing optimization algorithms and their combinations for scenario-based testing for UAVs exists. Thus, there is a need for a methodology to generate challenging test cases for testing the safe behavior of UAVs while considering the environment and the possibility that a safety distance might not always be available. Further, an evaluation of different optimization algorithms is needed to gain confidence in the results of the test case generation process that uses these algorithms.

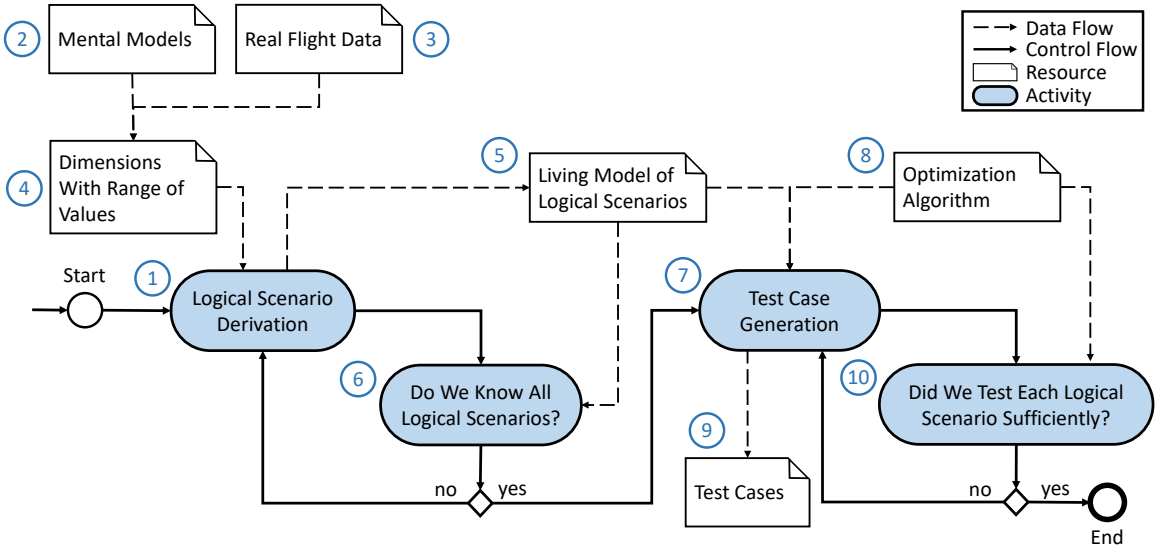


Figure 1.1.: Overview of the process of testing the safe behavior of UAVs with scenario-based testing. Previous versions appeared in [114, 115, 116, 118].

1.3. Solution

This work introduces a methodology for testing the safe behavior of UAVs with scenario-based testing and presents a tool called StellaUAV, which implements this methodology. We share the source code for our presented tool StellaUAV in [117]. We introduce an overview of our approach in Figure 1.1 and describe it in the following. In this thesis, we focus on steps ① and ⑦ and provide a first step towards handling the presented challenge in ⑩.

First, we need to derive those situations in which we aim to test the UAV's behavior ①. We call these situations logical scenarios and introduce them in more detail in Chapter 2. In this first step, we build an ontology that characterizes these logical scenarios for testing the safe behavior of UAVs ⑤. We can derive the dimensions for such an ontology (1) manually from mental models ② presented in specifications, literature, and expert knowledge or (2) automatically by clustering real-flight data ③ that we previously gathered. After acquiring the dimensions of the ontology, we need to explore a lower and upper bound for the values of each dimension ④ before dividing them system-specifically into categories that present various relevant situations for the SUT. Note that we view the resulting ontology as a "living model" that we can adapt to newly discovered challenging situations or applications for the SUT. To provide a thorough safety argumentation, we need to collect all relevant logical scenarios in which we need to test our system's behavior ⑥. The derivation of such a complete list of logical scenarios is still an open research challenge. If we lack confidence in the completeness of the derived logical scenarios, we need to go back to step ① and gather additional logical scenarios. Once we are sufficiently confident that we found a

complete list of logical scenarios, we can generate test cases for each logical scenario ⑦. Instead of randomly creating test cases, we aim to search for worst-case situations in which a correct system operates safely meanwhile an incorrect system behaves unsafely by, e.g., violating specified safety distances. We can use various optimization algorithms ⑧ to find test cases that represent worst-case situations for the SUT ⑨. Finally, we need to guarantee that we tested each logical scenario sufficiently ⑩ to ensure a high level of safety for the SUT. If we do not meet this requirement, we need to generate more test cases ⑦ before stopping the testing process. Current literature [48, 147] proposes using heuristic optimization algorithms for generating worst-case situations in step ⑦. Since these algorithms, however, cannot ensure finding an optimal solution due to their heuristic nature, guaranteeing sufficient testing of each logical scenario ⑩ gets more challenging. As a first step towards handling this additional challenge, we investigate the performance of different optimization algorithms for finding worst-case situations in this work. Further, we explore whether sequentially combining optimization algorithms improves the quality of the generated worst-case situations for testing the safe behavior of UAVs.

With our proposed methodology, we intend to yield the *basis for a well-founded safety argumentation* about the safe behavior of UAVs with scenario-based testing. We present a solution to two fundamental challenges of this technique. First, we provide a methodology for deriving a system-specific ontology that characterizes logical scenarios for UAVs and the resulting ontology. In addition, we discuss the necessary conditions and assumptions for deriving a complete list of relevant situations for UAVs with this approach. Secondly, we present a methodology for generating worst-case situations for these logical scenarios to ensure the safe behavior of the UAV even in the most challenging circumstances of each situation. Further, to assess the quality of the created test cases, we emphasize the need to evaluate their quality for various optimization algorithms and present a case study for three of them and their combinations.

1.4. Contributions

This work includes the following contributions to the presented gaps in the literature:

- To fill **Gap 1** and provide a solution for ① in Fig. 1.1, we present an ontology to characterize logical scenarios in which we aim to test the safe behavior of a quadcopter as one kind of UAV. This ontology includes concrete dimensions and sub-categories to allow for a direct derivation of specific logical scenarios to test the SUT. We can acquire the lower and upper bounds for the parameter values of each dimension either (1) from expert knowledge and specifications or (2) from experimental results for different bounds. To perform these experiments, we present an automated approach for exploring the bounds of a dimension on the example of finding an upper bound on the number of relevant obstacles that we need to consider in logical scenarios for our SUT. Further, we represent the systematic methodology to generate an ontology

that characterizes logical scenarios for testing the safe behavior of UAVs to enable researchers and test engineers to produce a similar ontology for their SUT. In addition, we discuss the necessary conditions for the completeness of the derived logical scenarios from the provided ontology to *provide a basis* for (6) in Fig. 1.1.

- To fill **Gap 2** and provide a solution for (7) in Fig. 1.1, we demonstrate how we can effectively apply scenario-based testing and search-based techniques to generate challenging test cases for UAVs. Further, we investigate two use cases for this methodology: (1) we can specify a safety distance that the UAV should keep to all obstacles and can use this safety distance to explicitly define a safe behavior of the SUT, and (2) we cannot determine such a safety distance and need to generate worst-case situations for our SUT differently. Further, to provide a basis for solving (10) in Fig. 1.1, we present a case study about the performance of various optimization algorithms and their sequential combinations for generating test cases for testing the safe behavior of UAVs. This case study reveals one of the crucial problems of scenario-based testing: different optimization algorithms create substantially varying worst-case situations and frequently miss finding the optimal ones. These results lead to the insight that we have the additional cost of performing multiple optimization algorithms when applying scenario-based testing. Finally, we present the tool StellaUAV that can generate worst-case situations and evaluate the performance of various optimization algorithms for the open-source PX4 autopilot for UAVs [81].

Parts of these contributions have previously appeared in the following peer-reviewed publications, co-authored by the author of this thesis:

- **Tabea Schmidt**, Florian Hauer, Alexander Pretschner, "Understanding Safety for Unmanned Aerial Vehicles in Urban Environments", IEEE Intelligent Vehicle Symposium (IV), 2021
- **Tabea Schmidt**, Florian Hauer, Alexander Pretschner, "Exploring a Maximal Number of Relevant Obstacles for Testing UAVs", International Conference on Computer Safety, Reliability, and Security, 2022
- **Tabea Schmidt**, Alexander Pretschner, "StellaUAV: A Tool for Testing the Safe Behavior of UAVs with Scenario-Based Testing", IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE), 2022
- **Tabea Schmidt**, Alexander Pretschner, "Ontology-Based Collection of Scenarios for Testing UAVs", IEEE Robotics and Automation Letters, 2022, under review

1.5. Summary of Results

As the first result of this thesis, we present an ontology that describes logical scenarios for testing the safe behavior of a quadcopter as one kind of UAV. We view the resulting

ontology as a "living model" that we can adjust to newly discovered challenging situations and circumstances. In addition, we formulated two conditions that we need to fulfill to derive a complete list of logical scenarios from the created ontology: (1) the generated ontology needs to be complete, and (2) the defect hypothesis that we apply for selecting specific logical situations from the ontology needs to represent all challenging situations for the SUT.

Our evaluation results for exploring an upper bound for the number of relevant obstacles reveal that the two investigated optimization algorithms, Multiobjective Evolutionary Algorithm Based on Decomposition (MOEA/D) and Non-dominated Sorting Genetic Algorithm II (NSGAII), show different performances when searching for extreme parameter values. This observation indicates the need to find the most suitable algorithm system-specifically for the presented methodology. Further, the experimental results for MOEA/D imply that we need to consider a maximum of 5 to 8 obstacles depending on the fault hypothesis used for data collection. By providing system-specific lower and upper bounds for the parameter values of each dimension of the ontology, we can effectively limit the number of logical scenarios in which we need to test the SUT's behavior.

With our proposed methodology for generating test cases for testing the safe behavior of UAVs, we discover 4 to 62 safety distance violations for four logical scenarios for the open-source PX4 autopilot [81] when we can specify the safe behavior of UAVs with a safety distance. Further, we find questionable behaviors of the tested UAV with our proposed methodology when no safety distance is defined. These results show the effectiveness and applicability of our proposed approach for generating worst-case situations that reveal potential faults in the SUT. Even though experts still need to investigate the behavior of the UAV manually when we cannot explicitly specify the safe behavior of the UAV, we minimize the experts' efforts by presenting worst-case situations that they solely need to inspect.

When evaluating optimization algorithms and their combinations, we discover that one combination outperforms the algorithm used in the literature by 20%, on average, considering the quality of the created test cases. In addition, several evaluated algorithms even generate test cases that reveal the unsafe behavior of the SUT in the tested logical scenarios. However, the results also demonstrate that we cannot blindly trust optimization algorithms to find worst-case situations, even if we select the best-performing one from a context-specific analysis of different optimization algorithms. Even though we know that heuristic optimization algorithms do not guarantee to find an optimal solution, our results show that this is not an exceptional case. These results indicate that we do not only need to perform several runs per logical scenario but also apply multiple optimization algorithms when generating test cases for testing the safe behavior of UAVs with scenario-based testing. Considering the required resources for performing scenario-based testing, this may be seen as a crucial challenge to the application of scenario-based testing.

In our experiments for evaluating the applicability of StellaUAV, we detect several safety distance violations of the open-source PX4 autopilot [81] in various logical scenarios. The results show that the SUT has problems when facing dynamic obstacles as we detect safety

distance violations for all 24 logical scenarios with dynamic obstacles. On the other side, the SUT behaves safely when only encountering static obstacles in our experiments by not violating any safety distances. Finally, the experimental results present the applicability of StellaUAV and its effectiveness for finding worst-case situations in which the SUT shows potentially unsafe behavior.

While working on this thesis and building a methodology for testing the safe behavior of UAVs with scenario-based testing, we gathered several lessons learned: (1) we need to derive logical scenarios for testing the safe behavior of UAVs system-specifically, (2) collecting a complete list of logical scenarios presents various open challenges, (3) explicitly defining the safe behavior of UAVs is not easily accomplished due to missing regulations, (4) we need to execute multiple optimization algorithms to reliably find worst-case situations for UAVs, and (5) scenario-based testing increases the confidence in the safe behavior of UAVs.

1.6. Structure

In Chapter 2, we introduce the essential aspects of autonomously operating UAVs and the concepts of scenario-based testing and search-based techniques. Chapter 3 describes an ontology that characterizes logical scenarios for testing the safe behavior of quadcopters and depicts the corresponding methodology. Next, Chapter 4 presents an automated approach for finding relevant bounds for the parameter values of the ontology dimensions. In Chapter 5, we represent a methodology for testing the safe behavior of UAVs while considering their environment. Chapter 6 presents a case study that explores the quality of generated test cases for three optimization algorithms and their sequential combinations. In Chapter 7, we outline the tool StellaUAV that we implemented for applying the presented methodology. Finally, Chapter 8 discusses related work about testing the safe behavior of UAVs and scenario-based testing before Chapter 9 concludes with a discussion of the results of this thesis and ideas for future work.

2. Background and Preliminaries

This chapter provides a general overview of autonomously operating UAVs and the concepts of scenario-based testing and search-based techniques for finding worst-case situations to test the safe behavior of UAVs. Parts of this chapter previously appeared in peer-reviewed publications [114, 115, 116] co-authored by the author of this thesis.

2.1. Autonomously Operating UAVs

As there are various types of UAVs, we first outline their different characteristics. Table 2.1 presents an overview based on [73, 129] that focuses on the payload weight that the UAVs can carry in kilograms, their flight endurance in hours, and the altitude at which they operate in meters. Note that we can include additional characteristics such as launch and recovery systems or flight control interfaces to specify different types of UAVs in more detail [73]. In this work, we focus on quadcopters representing small UAVs in Table 2.1 to show the applicability of our proposed approach. However, note that we can also apply our methodology to other types of UAVs.

As we present a methodology for testing the safe behavior of autonomously operating UAVs in this work, we need to define what level of autonomy we are considering. In the literature, we can find several ideas for categorizing different levels of autonomy. The National Institute of Standards and Technology describes three primary dimensions to define autonomy: the complexity of the UAV's mission, the difficulty of the UAV's

Table 2.1.: Overview of the characteristics of different types of UAVs, as presented in [73, 129].

Category	Payload Weight	Endurance	Altitude
Micro	< 1 kg	< 1 hour	near ground level
Small	$1 - 10$ kg	$2 - 5$ hours	near ground level
Tactical	$10 - 50$ kg	$4 - 8$ hours	$0 - 1,500$ m
Medium Altitude and Endurance	$50 - 300$ kg	> 12 hours	$4,500 - 9,000$ m
High Altitude and Endurance	$50 - 800$ kg	> 24 hours	$> 15,000$ m

environment, and the UAV's independence from humans [52]; In addition, in [50], the authors describe four principles that autonomous systems implement: the knowledge of the UAV about itself and its environment, the UAV's adaptation mechanisms to cope with a dynamically changing environment, the UAV's self-awareness, and the UAV's emergency from its simple components to its complex characteristics; Further, the US Air Force Research Laboratory defines eleven levels of autonomy in their work [28] that characterize the situational awareness of the UAV, its decision-making characteristics, and its communication and cooperation capabilities for each level. Finally, the authors of [140] from NASA consider this classification too fine-grained for their use case of high-altitude long-endurance UAV missions and, thus, propose a simplified version to describe the levels of autonomy. They describe in [140] the following levels of autonomy:

- **Level 0 - Remotely Controlled**: "Remotely piloted aircraft with a human in the loop, making all the decisions. Operator is in constant control."
- **Level 1 - Simple Automation**: "Remotely piloted with some automation techniques to reduce pilot workload. Human monitoring to start/stop tasks."
- **Level 2 - Remotely Operated**: "Human operator allows UAV on-board systems to do the piloting. As part of the outer control loop, the human makes decisions as to where to go, when, what to do once there. Remotely supervised, with health monitoring and limited diagnostics. Operator allows UAV to execute preprogrammed tasks, only taking over if the UAV is unable or fails to properly execute them."
- **Level 3 - Highly-Automated or Semi-Autonomous**: "UAV automatically performs complex tasks. System understands its environment (situational awareness) and makes routine decisions and mission refinements to dynamically adjust to flight and mission variables. Limited human supervision, managed by exception. Adaptive to failures and evolving flight conditions."
- **Level 4 - Fully Autonomous**: "UAV receives high-level mission objective (e.g., location, time), translates them into tasks that are executed without further human intervention. UAV has the ability and authority to make all decisions. Extensive situational awareness (internal and external), prognostics, and on-board flight re-planning capability. Single vehicle operations."
- **Level 5 - Collaborative Operations**: "Brings in aspects of multiple UAVs working autonomously together as a collective intelligent system. Group coordination. Individual vehicles/systems in a collaborative group will have a least semi-autonomous LOA (3) to keep the operator workload of the collaborative operation at a manageable level."

In this work, we follow this last classification presented in [140] as its granularity level seems reasonable for our use case. Following the terminology, we focus on testing the safe

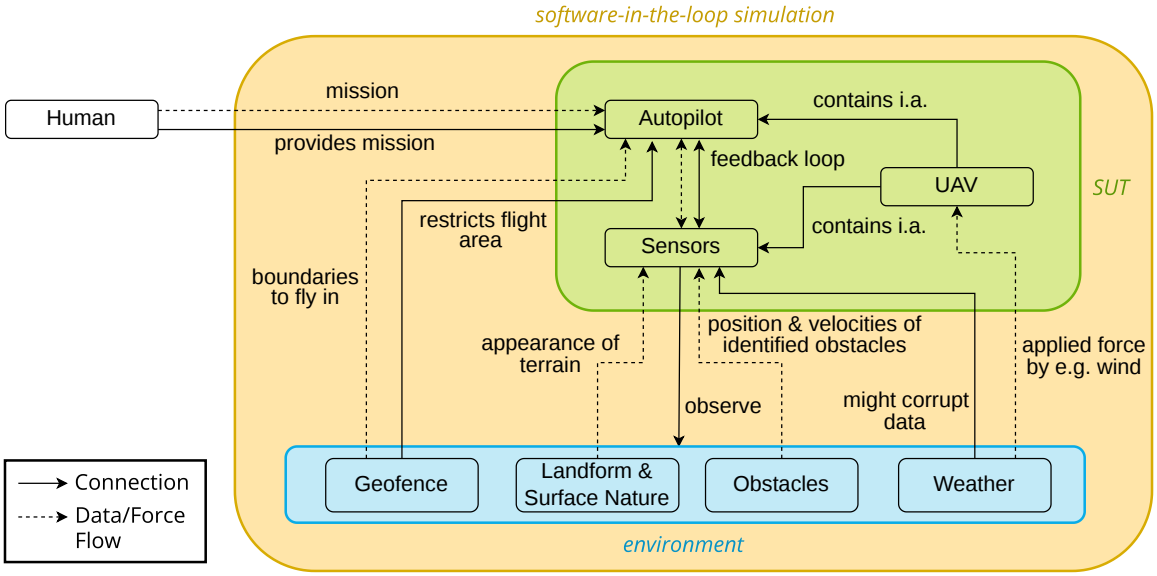


Figure 2.1.: We depict the boundary of the SUT in this work and its connections to its environment and a human. In this work, we simulate both the SUT and its environment.

behavior of UAVs of autonomy level 4 in this work, where the UAV receives a mission that it performs completely autonomously without human interaction. Note that we can also apply our proposed methodology to level 3 systems. In future work, we aim to extend our approach to test the safe behavior of autonomously operating cooperative UAVs, which represent an autonomy level of 5.

Finally, we describe the boundary of the SUT and its connections to other entities in its environment. In this work, the SUT is an autonomously operating small UAV of autonomy level 4 consisting of an autopilot, sensors, actuators, a hardware platform, and several software components for, e.g., detecting obstacles in its environment. We depict this SUT and its connections to its environment and humans in Fig. 2.1. As the SUT operates completely autonomously, the human only provides the UAV's mission and does not interfere with the UAV in any other way. The UAV's environment might include different elements, such as a geo-fence to restrict the flight area of the UAV and a specific landform and surface nature that presents various terrains for the UAV. In addition, the environment might contain obstacles that the UAV needs to avoid and different weather conditions that might impact the sensor readings and flight capabilities of the UAV. Note that we apply a Software-in-the-Loop approach in this work that includes a simulation of the SUT and its environment. However, we can also use the presented methodology for Hardware-in-the-Loop or Model-in-the-Loop testing.

Table 2.2.: Description of the parameters and their value ranges of an exemplary logical scenario for testing the safe behavior of UAVs. The logical scenario includes light precipitation, cold ambient temperature, complete cloud coverage, and heavy fog. A previous version appeared in [116].

	Parameter P	Value Range (Min, Max)
p_0	precipitation [cm/h]	(0.1, 0.25)
p_1	temperature [°C]	(5.0, 10.0)
p_2	cloud coverage [%]	(90.0, 100.0)
p_3	reduced visibility [m]	(40.0, 200.0)

Table 2.3.: Potential concrete scenarios for the logical scenario described in Table 2.2. A previous version appeared in [116].

	Parameter P	Test Case 1	Test Case 2	Test Case 3	Test Case 4
p_0	precipitation [cm/h]	0.23	0.17	0.21	0.11
p_1	temperature [°C]	6.6	9.5	8.3	5.9
p_2	cloud coverage [%]	98.0	90.2	96.3	92.5
p_3	reduced visibility [m]	175.0	88.5	134.7	41.2

2.2. Abstraction Level of Test Scenarios

In this work, we differentiate between logical scenarios and concrete scenarios for scenario-based testing, as proposed by the authors of [82]. As previously mentioned, logical scenarios describe the typical situations in which we aim to test the UAV's safe behavior. We use parameters $P = \{p_0, p_1, ..., p_n\}$ and corresponding value ranges for these parameters to characterize these logical scenarios. In Table 2.2, we present an example of a logical scenario for testing the safe behavior of UAVs. Note that this exemplary logical scenario does not include all necessary parameters to comprehensively describe relevant test situations for UAVs for simplicity of presentation. The represented logical scenario describes the UAV's environment with several environmental conditions that the UAV faces, such as light precipitation, cold ambient temperature, cloud coverage, and heavy fog. With the parameter p_0, we describe the heaviness of the rain in the logical scenario with 0.1 to 0.25 centimeters per hour. Further, we represent the cold ambient temperature in a range of 5.0 to 10.0 degrees Celsius with p_1. For the cloud coverage, we specify its heaviness between 90 and 100 percent coverage with the parameter p_2. Finally, we limit the visibility of the UAV to 40.0 to 200.0 meters with the parameter p_3 to describe heavy fog. Note that when specifying a logical scenario, we must determine system-specific parameter ranges for each parameter of that logical scenario to define what, for example, a cold ambient temperature represents for the SUT.

For a concrete scenario, we pick concrete values from the ranges of all parameters of a logical scenario. A concrete scenario, thus, represents one test case for the given logical scenario. Table 2.3 presents four potential concrete scenarios that we can choose for the presented logical scenario in Table 2.2. In the first test case, we set the precipitation rate to 0.23 centimeters per hour, the ambient temperature to 6.6 degrees Celsius, the cloud coverage to 98.0 percent, and the reduced visibility to 175.0 meters.

2.3. Generation of "Good" Test Cases

When specifying test cases for a SUT, we need to define the test input for this test case and its expected output. The selected concrete scenario denotes the test input by, e.g., describing the UAV's mission, the UAV's environment, and failures that the UAV needs to handle. For this test input, as an expected output, we anticipate a safe behavior of the UAV during its operation. However, it is not easy to explicitly define this safe behavior in all situations. We discuss this challenge and its implications for testing the safe behavior of UAVs in Chapter 5. The concrete scenarios of a logical scenario present all possible test cases for this logical scenario. Since it is infeasible to test all of them, we need to choose specific ones. When we apply a random selection, we cannot measure the quality of the selected test cases. Thus, we propose to, instead, use the idea of [97] and search for so-called "good" test cases that can reveal potential faults in the SUT. These "good" test cases, thus, represent challenging situations for the SUT. When we can define a UAV's safe behavior by specifying a safety distance that it should keep to all obstacles, a safe system would approach this safety distance but not overstep it in a "good" test case. On the other hand, an unsafe system would violate the specified safety distance in the same test case.

To perform such a search with optimization algorithms, we first need to specify the search space in which the search should take place and the fitness function, which guides our search for "good" test cases. The parameter ranges of a logical scenario represent the search space for this logical scenario. When testing the safe behavior of UAVs, we can use different objectives to guide our search for "good" test cases. If there exists a specified safety distance s that the UAV should keep to all obstacles, we can assess the UAV's behavior by comparing the distance d that it keeps to all obstacles with this safety distance s. If $d \geq s$, the UAV behaves safely. Otherwise, it violates the specified safety distance. However, if we cannot define such a safety distance for a specific situation, we need to adapt our search for challenging situations. We describe fitness functions for both of these cases in Chapter 5. Another approach is to specify a so-called safe operating envelope, in which the UAV can freely operate but which it should not leave. Finally, we can define additional quality attributes, such as smoothness, that the UAV should also optimize while keeping sufficient safety distance from all obstacles. We depict these three alternatives in Fig. 2.2. In this work, we focus on the first case, even though we can also apply our proposed approach to the other options.

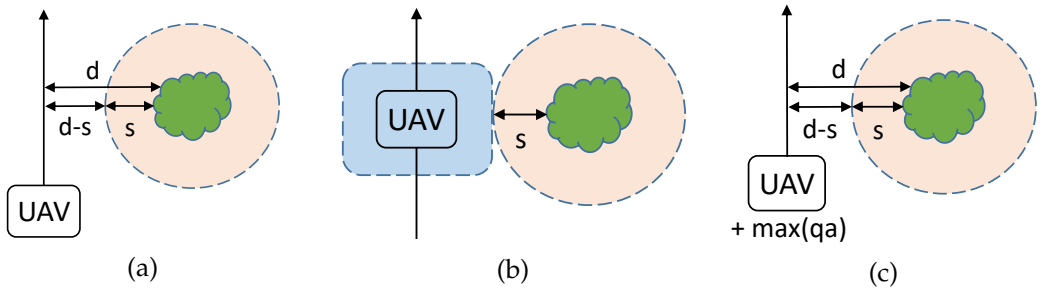


Figure 2.2.: When testing the safe behavior of UAVs, we can test against (a) a given safety distance s, (b) with a safe operating envelope around the UAV, here in blue, or (c) against a given safety distance s and additional quality attributes qa.

2.4. Optimization Algorithms

When searching for "good" test cases, our search space is too high dimensional to apply exact algorithms such as integer linear programming, dynamic programming, or branch-and-bound [98]. Instead, we can use different optimization algorithms such as heuristic algorithms or surrogate optimization algorithms to compute sound solutions for our problem with a decent cost-benefit ratio. However, these optimization algorithms cannot provide a guarantee to find an optimal solution for a given problem. We discuss the implications of this downside and a potential approach to solve this problem in Chapter 6. In this work, we focus on three optimization algorithms, namely NSGAII, Particle Swarm Optimization (PSO), and Bayesian Optimization (BO), which we describe in more detail in the following subsections.

2.4.1. Non-dominated Sorting Genetic Algorithm II (NSGAII)

NSGAII [32] is the state-of-the-art optimization algorithm for generating "good" test cases for ADS and UAVs [48, 114, 146]. The algorithm works with populations which are a set of candidates that represent test cases for the SUT in our use case. During its operation, NSGAII tries to improve the quality of the test cases in subsequent populations to achieve the specified goal described by the fitness function. The algorithm is based on the survival-of-the-fittest principle to find sound solutions. Further, it applies elitism to guarantee that these solutions are part of future populations and do not get lost. To achieve these two purposes, NSGAII uses so-called non-dominated sets of the last two populations for choosing new candidates for the next population. A non-dominated set includes all candidates of a population that are not dominated by other candidates, as the name suggests. We depict the workflow of NSGAII in Fig. 2.3 and outline it in the following: First, NSGAII creates an initial population of N candidates P_0 that it chooses randomly from the given search space ①. As the termination criterion is not yet met ②, the algorithm computes the fitness of each candidate of this population with the provided fitness function ③. Based on

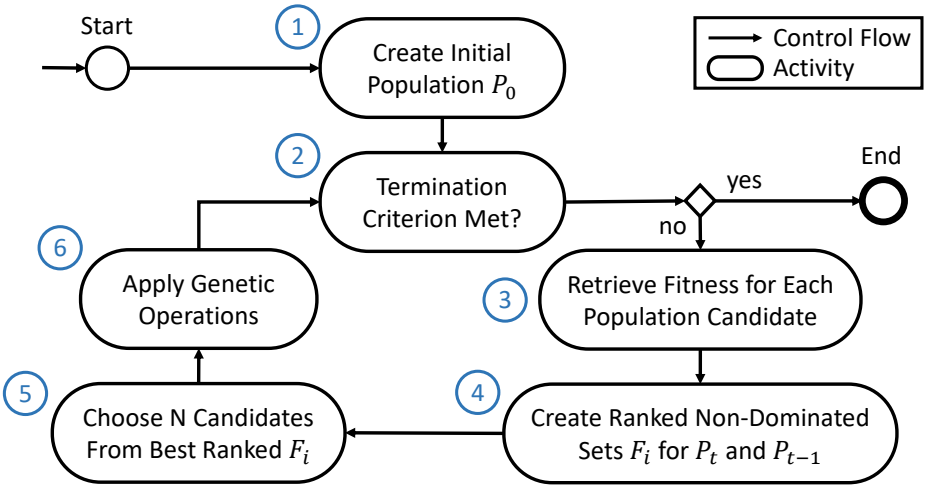


Figure 2.3.: Workflow of the optimization algorithm NSGAII.

these fitness values, NSGAII creates the non-dominated sets F_i ④. The first set F_1 contains those candidates of the population that are not dominated by other candidates. Next, we exclude the candidates of F_1 and collect the candidates that are not dominated by the others in this reduced set in F_2. We repeat this process until we have ranked all candidates into non-dominated sets. After the initial population, we include the candidates of the previous population P_{t-1} and the current population P_t for building F_i in this step. Then, NSGAII chooses N candidates from the best ranked non-dominated sets F_i for the next population ⑤. Before repeating the described process, the algorithm applies genetic operations such as selection, mutation, and crossover operations to explore candidates close to the already evaluated ones ⑥. When the termination criterion is met ②, NSGAII returns the best performing candidates according to the fitness function.

2.4.2. Particle Swarm Optimization (PSO)

As another heuristic optimization algorithm, PSO [56] is frequently part of empirical case studies about the performance of various optimization algorithms for different problems [122, 125]. For generating sound solutions, PSO imitates how animals behave in a flock with the help of so-called particles. Each of these particles presents a test case for the SUT and has a position that represents its location in the search space and a velocity with which it moves through the search space in a specified direction. Each particle further stores the best fitness value it achieved so far in a variable called *pbest*. In addition, PSO stores the overall best fitness value found until now in a variable with the name *gbest*. We present the process of PSO for creating sound solutions in Fig. 2.4. First, the algorithm spreads N particles randomly over the search space ①. As the termination criterion is not yet met ②, PSO calculates the fitness value for each particle with the provided fitness function ③.

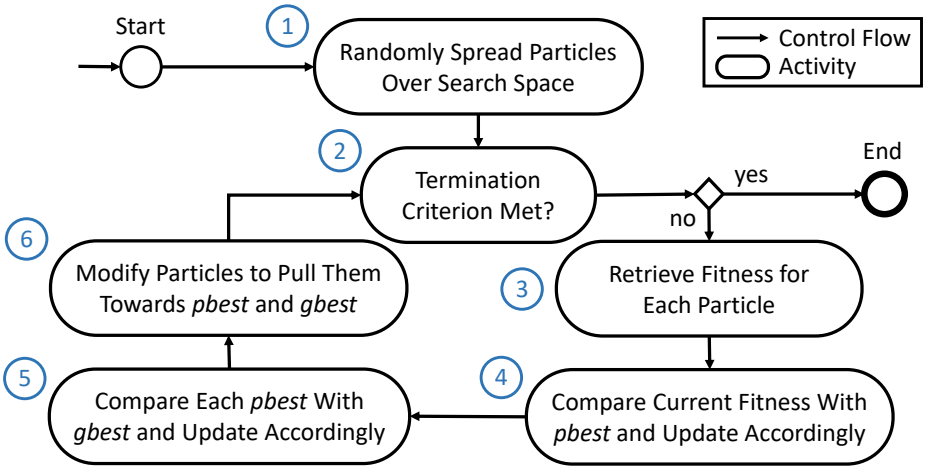


Figure 2.4.: Workflow of the optimization algorithm PSO.

Then, it checks whether the current fitness of each particle is better than their stored *pbest* ④. If this is the case, PSO updates *pbest* to the current location and fitness value. Next, PSO compares the *pbest* of each particle with the fitness value of *gbest* ⑤, which denotes the globally best fitness value achieved so far. If one of the particles has a *pbest* with a better fitness value than *gbest*, we change the position and fitness value of *gbest* to the parameter values of this particle. Finally, the algorithm alters the particles' positions and velocities by pulling them towards their *pbest* and *gbest* ⑥. The algorithm repeats this process until a termination criterion is met ② and afterward returns the candidates with the best fitness value.

2.4.3. Bayesian Optimization (BO)

The optimization algorithm BO [96] uses another approach than NSGAII and PSO to generate "good" test cases for our SUT. As there are various implementations of this algorithm in the literature, we decided to focus on the one from Frazier et al. [42]. Contrary to the other presented algorithms, BO models the fitness function with a Gaussian process and retrieves candidates based on the posterior distribution of this process. For generating a new candidate, the algorithm applies an acquisition function to yield a promising candidate. There are various acquisition functions, e.g., probability of improvement, upper confidence bound, or expected improvement. In Fig. 2.5, we present an overview of the process of BO for generating sound solutions for a given problem. First, BO uses a Gaussian process to represent the provided fitness function ①, as mentioned before. Next, it randomly chooses candidates from the search space and evaluates their fitness with the given fitness function ②. As the termination criterion is unmet ③, the algorithm updates the Gaussian process with the candidates and their fitness values ④. Then, BO utilizes an acquisition function to gain the next promising candidate for a decent solution ⑤. Finally, the algorithm

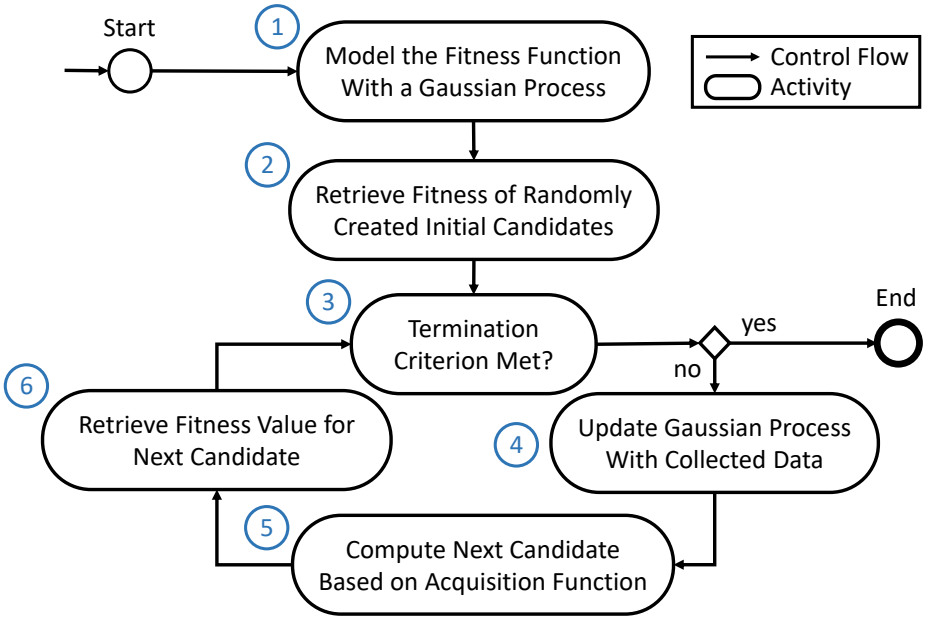


Figure 2.5.: Workflow of the optimization algorithm BO.

calculates the fitness of the retrieved candidate with the provided fitness function ⑥ before reiterating through the process. When the termination criterion is met ③, BO returns the best performing candidates for the given search problem as the other optimization algorithms.

In addition, BO offers the possibility to compute a 95% credible interval. With such a credible interval, we describe a range of values in which the fitness values of the given fitness function reside with a probability of 95%. However, to achieve tight credible intervals that provide insight into the safe behavior of the SUT beyond the evaluated candidates, we need to perform a large number of evaluations. As testing the safe behavior of UAVs is highly time-consuming, we usually cannot perform such a large number of evaluations. In addition, the high dimensionality of our search space for this problem complicates the computations concerning memory consumption.

Part II.

Logical Scenario Derivation

3. Methods and Challenges of Deriving Logical Scenarios for UAVs

This chapter outlines one of the fundamental challenges of scenario-based testing: the definition of logical scenarios to test the UAVs' safe behavior. First, it discusses the challenges of automatically deriving logical scenarios from collected flight data before providing an ontology to characterize them based on mental models for a quadcopter as one kind of UAV. Parts of this chapter previously appeared in a peer-reviewed submission under review [118] co-authored by the author of this thesis.

3.1. Introduction

When testing the safe behavior of UAVs with scenario-based testing, we aim to ensure that the SUT will behave safely in each relevant situation that it might encounter in the real world. We describe these situations with various parameters and call them logical scenarios, as explained in Section 2.2. We can derive logical scenarios for testing the safe behavior of UAVs in two ways: (1) automatically by clustering relevant real-world flight data that we previously collected, or (2) manually based on mental models of relevant situations presented in the literature, specifications, or expert knowledge. In this chapter, we first present an approach for (1) and discuss its open research challenges before focusing on acquiring logical scenarios based on mental models, as introduced in (2). When applying this second approach, we can divide the main challenge of deriving all relevant logical scenarios based on mental models into the following four sub-problems: (1) finding relevant dimensions to represent logical scenarios, (2) investigating a suitable level of granularity for describing logical scenarios for a SUT, (3) presenting the derived logical scenarios in a machine-readable way, and (4) combining the acquired dimensions to collect specific logical scenarios that represent relevant and challenging situations for the SUT.

In related work, use case scenarios for UAVs [36, 72, 110] focus on the mission of the UAVs and include a limited number of environmental effects. Extending their work, we aim to provide an ontology that presents relevant dimensions for characterizing logical scenarios for UAVs in more detail. For testing the safe behavior of ADS, there exist such ontologies [10, 90] that describe logical scenarios for these systems. However, these ontologies present meta-models of these situations and lack concrete dimensions to derive specific logical

scenarios. In addition, these ontologies focus on logical scenarios for ADS which look different than those for UAVs. Considering sub-problem (1), several papers [22, 51, 124] focus on specific use cases or systems when describing relevant dimensions for logical scenarios for UAVs. We use their work and papers from other domains [11, 67] as a basis for providing an ontology that describes all relevant dimensions for testing the safe behavior of UAVs in this work. In the automotive domain, several papers [46, 47, 90] stress the importance of finding a suitable level of granularity when describing logical scenarios, which presents sub-problem (2). Still, choosing an adequate granularity level remains an open research challenge. In terms of sub-problem (3), existing languages concentrate on describing logical and concrete scenarios [7, 53, 99] for testing ADS or specific situations for aircraft landing scenarios. Finally, [68] presents one approach for tackling sub-problem (4) by applying combinatorial methods for selecting specific logical scenarios from the dimensions of an ontology for ADS. In our opinion, this selection process should be performed system-specifically and based on a corresponding defect hypothesis of challenging situations for the SUT.

The **contribution of this chapter** is, firstly, a discussion about the applicability and the open research challenges for automatically deriving logical scenarios for UAVs. In addition, we secondly present an ontology that characterizes relevant and challenging logical scenarios for testing the safe behavior of a quadcopter as one kind of UAV based on mental models. To allow test engineers and researchers to create similar ontologies for relevant logical scenarios for their SUTs, we further introduce the systematic methodology to derive such an ontology. Note that we consider the presented ontology as a "living model" that we can adapt to newly arising challenging situations for the SUT.

3.2. Challenges of Clustering Collected Data to Automatically Acquire Logical Scenarios

As mentioned earlier, we can acquire logical scenarios (1) automatically with clustering techniques, or (2) based on mental models. When we aim to apply the first approach and automatically derive logical scenarios for our SUT with clustering techniques, we need high amounts of diverse and relevant real-world flight data from UAVs similar to our SUT as input. Ideally, we then gain logical scenarios from this data that can complement the manually derived ones from mental models. However, we currently lack these high amounts of diverse data for UAVs to apply this automatic derivation. Nonetheless, we show the general idea of using these clustering techniques to acquire logical scenarios and point out their challenges in this section by using simulated data.

3.2.1. Automated Clustering Approach

For automatically clustering collected real-flight data to derive logical scenarios, we adapt approaches from the literature [47, 113] co-authored by the author of this thesis. Figure 3.1

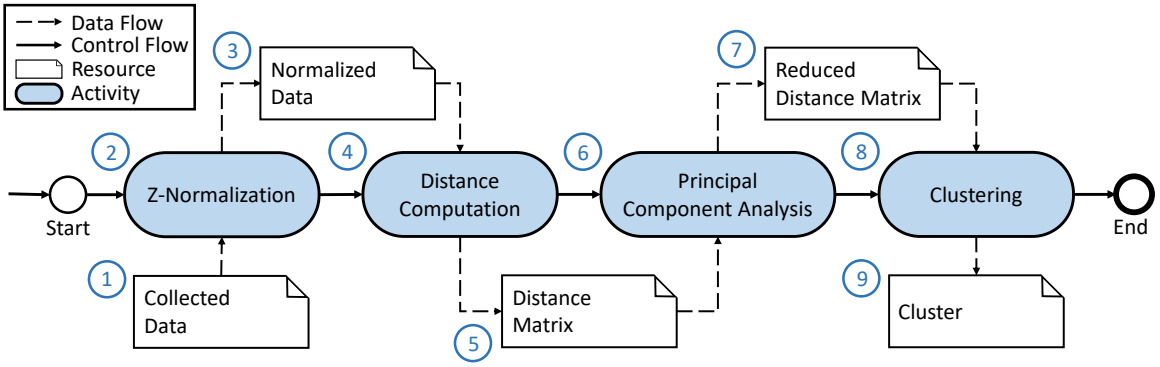


Figure 3.1.: Process overview of the clustering approach for automatically deriving logical scenarios for UAVs from collected real-world data.

depicts an overview of this automated clustering approach for acquiring logical scenarios for testing the safe behavior of UAVs. The input to the clustering approach is the collected real-flight data ① representing diverse and relevant data for the SUT in the form of concrete scenario instances. As logical scenarios group concrete scenarios of a similar structure, we first apply z-normalization ② to the data to focus on the structure of the data rather than the absolute values we collected. For example, a logical scenario might describe several concrete scenarios in which the UAV speeds up by 5 kilometers per hour in 1 second. In this logical scenario, we focus on the speed increase and neglect the concrete starting speeds such as 1 kilometers per hour or 10 kilometers per hour of the various concrete scenarios. As a result of this step, we gain normalized data ③. Next, we feed this normalized data into the distance computation step ④. The distance between two concrete scenarios describes their similarity, with two identical concrete scenarios having a distance of zero. We calculate the distance between the collected concrete scenarios with Dynamic Time Warping (DTW) [86], as proposed in [16, 47, 69, 113]. We apply DTW since it can detect similar concrete scenarios even if their behavior is distorted along the temporal axis, as it often occurs in real-world data. After computing DTW distances between the parameters of two concrete scenarios, we can either work with these non-aggregated distances or aggregate them using, e.g., the squared sum of these values. Finally, we apply a min-max-normalization to the computed distances to scale all parameters to the interval $[0, 1]$. As a result of step ④, we gain a distance matrix ⑤ that represents the similarity between each of the collected concrete scenarios of our data. Before clustering based on this distance matrix, we apply a Principal Component Analysis (PCA) ⑥ to reduce its dimensionality and improve the clustering results. On this reduced distance matrix ⑦, we then apply a selected clustering method ⑧ such as K-Means [75], Hierarchical Clustering algorithms [87], or DBSCAN [39]. Depending on the chosen clustering method, we further need to apply techniques for finding an optimal number of clusters k. The literature proposes to compute the clustering for all possible numbers of clusters k and then use a knee/elbow detector such as Kneedle [111] to find

the optimal k. We can detect this knee or elbow in, e.g., the Silhouette Scores [106], the Calinski-Harabasz Indices [18], or the distortion values discovered for the varying cluster numbers. As a final result of this approach, we gain the computed clusters ⑨ that represent the logical scenarios acquired from the collected real-flight data.

Note that the gathered data can contain different collected parameters from the real-world flights of UAVs and that this clustering approach includes various parameters, such as the applied aggregation method, the used clustering algorithm, and the utilized metric for finding an optimal k. Due to these different parameters, we encourage researchers to execute an analysis for the various parameters when applying this approach. When performing such an analysis, we face the open research challenge of evaluating the quality of the generated clustering. In general, we look for a small number of large clusters that represent common logical scenarios that the UAV encounters and a high number of small clusters that represent rarely occurring logical scenarios still relevant for the UAV. This description, however, does not provide a concrete measurement to assess the quality of a clustering result. In related work, one approach in the automotive domain [47, 103, 136] is the manual analysis of the trajectories in each computed cluster to derive a corresponding logical scenario and evaluate the quality of the created clusters. However, such a manual analysis does not scale for the high amounts of data we need to investigate when acquiring logical scenarios from collected concrete scenarios. This problem of automatically assessing the quality of the resulting clustering is a general open research challenge in the clustering community. Accordingly, we believe that solving this research challenge in the future will be fundamental for the automatic derivation of logical scenarios for testing the safe behavior of UAVs with the presented methodology.

3.2.2. Experiments

Since we currently lack high amounts of relevant and diverse real-world flight data for UAVs to apply the presented clustering approach, we show its general applicability and its challenges with simulated flight data on a small scale. Note that we base the recording of this simulated data on logical scenarios based on mental models, which might impact the clustering results. In addition, we only conduct experiments on small data sets since we want to present the existing challenges and do not aim to acquire new logical scenarios from the evaluated data sets.

Setup and Implementation

In our experiments, we evaluate two data sets, one that includes concrete scenarios from distinct logical scenarios and one that incorporates concrete scenarios from similar logical scenarios. For each of these data sets, we simulate the UAV's behavior in five logical scenarios and collect data from 10 concrete scenarios for each of them. Thus, each data set consists of 50 concrete scenarios. In Table 3.1, we present the characteristics of the logical scenarios for which we collect data for the two data sets in our experiments by defining

their included landform, surface nature, obstacles, wind force, and reduced visibility. In the presented analysis, we focus on the difference between the two clustering algorithms, KMeans and Agglomerative Hierarchical Clustering, between the two metrics Silhouette Score and distortion values for finding an optimal k for these clustering algorithms, and between collecting 7 and 20 parameters of the UAV in the gathered data. We acquire data about the UAV's position in x-, y-, and z-direction and its orientation in x-, y-, z-, and w-direction. For the 20 parameters, we further include the linear acceleration in x-, y-, and z-direction, the linear speed in x-, y-, and z-direction, the angular velocity of one of the rotors in x-, y-, and z-direction and the distance of the UAV to each of the up to four obstacles in the concrete scenarios. We record these parameters every 0.33 seconds to enable a decent representation of the UAV's behavior. In addition, we use squared sum to aggregate the computed distances and do not compare other aggregation methods in our experiments. Note that this analysis does not present an evaluation of all possible settings for the proposed clustering approach but concentrates on those that seem to provide promising results. Finally, we set the PCA to retain 95% of the variance in the data in our experiments.

For implementing the proposed clustering approach, we use the fastdtw library [107] to calculate the DTW distances between the concrete scenarios and the Scikit-learn framework [95] for implementing the clustering algorithms, the metrics for finding an optimal number of clusters k, the min-max-normalization, and the PCA. We apply the Kneedle algorithm [111] to locate the optimal k in the presented metric values. To enable the reproducibility of these experiments, we present the version number of the used libraries, in Appendix B. In addition, we provide the concrete scenarios of the data sets used as input for the experiments, the computed DTW distances, and the clustering results for all settings in [112].

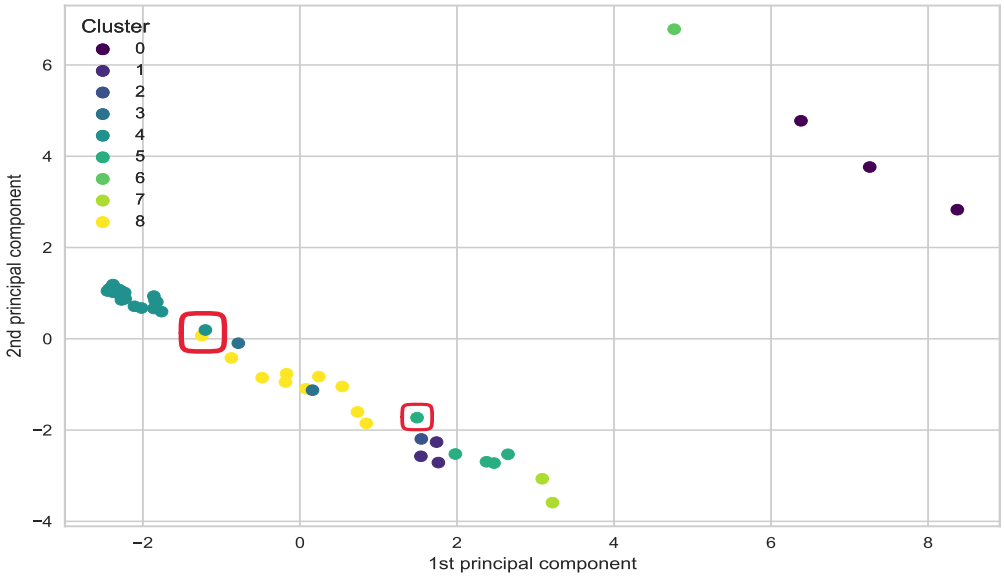
Experimental Results

In Table 3.2, we present the results of our analysis for the two data sets, one with the distinct logical scenarios D1-D5 and one with the similar logical scenarios S1-S5, which we characterize in Table 3.1. We further mark those clustering results which are identical for different settings. The presented results focus on the number of clusters that each setting produces and the size of these clusters. We consider clusters with one to five scenario instances as "small" and those with six or more scenario instances as "large". For the distinct logical scenarios and 20 collected parameter values, KMeans and Hierarchical clustering based on the Silhouette Scores produce the identical two large clusters, whereas the same algorithms based on the distortion values produce nine to ten clusters of which only four are large. When only considering seven parameters in our collected data, most of the settings create the identical nine clusters for this data set, with the exception of Hierarchical clustering based on silhouette scores with eleven clusters. For the second data set of similar logical scenarios, only the clustering results of KMeans and Hierarchical clustering for seven parameters and based on the Silhouette Score values are identical with

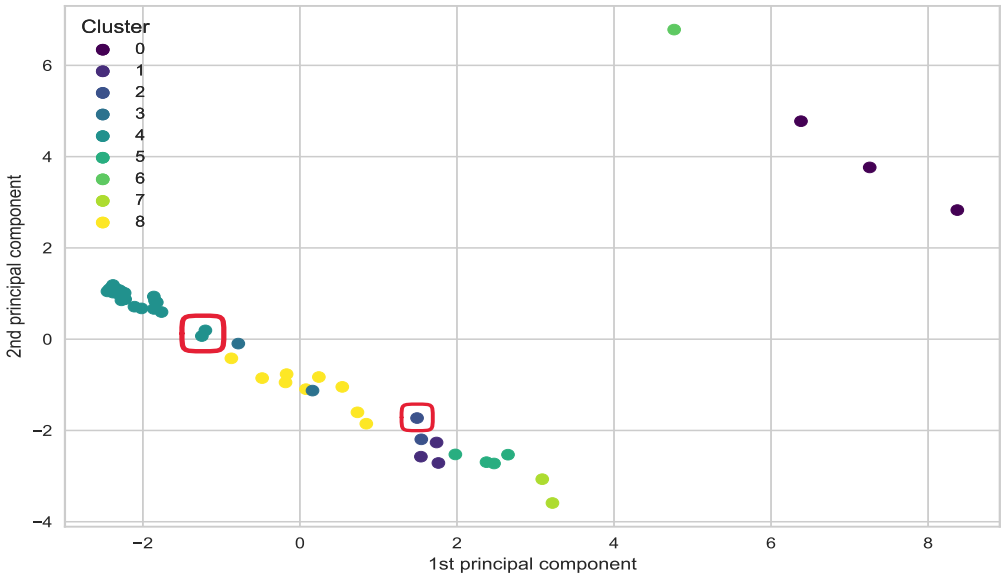
Table 3.1.: The logical scenarios on which we base the data collection in our experiments. They include a landform (flat F, elevation E, depression D, or steep transition ST), a surface nature (land L, water W, or a mixture M of them), the obstacles' kind (static ST or dynamic DY), the obstacles' size (small S, medium M, or large L), the obstacles' form (cuboid CU, sphere SP, or cylinder CY), the included wind force (none N, light L, moderate M, or strong S), and the reduced visibility (none N, fog F, heavy fog HF, or thick fog TF).

	Distinct Logical Scenarios					**Similar Logical Scenarios**				
Scenario	D1	D2	D3	D4	D5	S1	S2	S3	S4	S5
Landform	F	F	D	E	ST	D	D	D	D	D
Nature	L	M	L	L	M	L	W	L	M	W
Wind	L	S	N	N	S	L	M	S	L	S
Red. Visibility	N	TF	F	N	N	HF	HF	HF	N	N
# Obstacles	3	4	2	4	4	3	4	4	1	4
	ST	DY	DY	ST	DY	ST	DY	ST	DY	ST
Obstacle	ST	DY	DY	DY	ST	ST	DY	ST	—	ST
Kinds	DY	DY	—	ST	DY	ST	DY	DY	—	ST
	—	DY	—	DY	DY	—	DY	ST	—	DY
	L	S	L	L	S	L	M	M	S	L
Obstacle	L	S	S	L	M	M	L	S	—	L
Sizes	M	S	—	S	M	M	M	L	—	L
	—	S	—	S	L	—	M	S	—	M
	CU	CY	CY	CY	SP	SP	CY	CU	CU	SP
Obstacle	CU	SP	CY	CU	SP	CY	CY	SP	—	CY
Forms	SP	CY	—	CU	CU	CU	CU	CU	—	CY
	—	CY	—	SP	SP	—	SP	CU	—	CY

one small and one large cluster. The other settings produce different cluster results of six to twelve clusters, even though some of these clustering results are very similar. In Fig. 3.2, we display this slight difference by plotting the two principal dimensions of the clusters for seven collected parameters and the clustering results of KMeans and Hierarchical clustering based on the distortion values. This figure displays that the different settings assign only two of the 50 data points to other clusters, which we highlight with the red boxes in the figure.



(i) Clustering results with KMeans based on distortion values for seven recorded parameters of the data set based on the similar logical scenarios S1-S5.



(ii) Clustering results with Hierarchical clustering based on distortion values for seven recorded parameters of the data set based on the similar logical scenarios S1-S5.

Figure 3.2.: We depict the slight difference between the clustering results of the presented two settings with the red boxes in 2D plots of the clusters. The clustering algorithms assign two of the 50 data points to different clusters.

Table 3.2.: Experimental results of an analysis of different settings for our proposed clustering approach. We present the number of clusters in the resulting clustering and their size. A small cluster S includes one to five concrete scenarios, whereas a large cluster L contains six or more instances. In addition, we mark those clustering results that produce identical clusters for different settings with the symbols $*$, $\diamondsuit$, and $\triangledown$.

	20 Parameter				**7 Parameter**			
	KMeans		**Hierarchical**		**KMeans**		**Hierarchical**	
	Dist.	**Sil.**	**Dist.**	**Sil.**	**Dist.**	**Sil.**	**Dist.**	**Sil.**
Distinct Logical Scenarios D1-D5								
# cluster	9	2^*	10	2^*	$9^\diamondsuit$	$9^\diamondsuit$	$9^\diamondsuit$	11
cluster sizes	5S/4L	0S/2L	6S/4L	0S/2L	4S/5L	4S/5L	4S/5L	6S/5L
Similar Logical Scenarios S1-S5								
# cluster	12	6	9	6	9	$2^\triangledown$	9	$2^\triangledown$
cluster sizes	8S/4L	2S/4L	4S/5L	2S/4L	7S/2L	1S/1L	7S/2L	1S/1L

Discussion

With our experiments, we aim to demonstrate the applicability of the presented clustering approach for automatically deriving logical scenarios for UAVs and outline its challenges. Since we currently lack high amounts of real-world flight data, we perform these experiments on simulated flight data, which is reasonable for our presented goals. As mentioned earlier, assessing the quality of the clustering results is still an open research challenge. In general, we are interested in a small number of large clusters and a large number of small clusters to represent logical scenarios for testing the safe behavior of UAVs.

Our experimental results show a considerable difference in the clustering when recording the UAV's behavior with 7 or 20 parameters. While finding an optimal number of clusters k based on the Silhouette Score, the clustering algorithms generate two large clusters for the distinct scenarios and 20 recorded parameters. On the contrary, they create nine to eleven clusters for seven parameters. Since two large clusters do not provide enough insight to derive several relevant logical scenarios, only recording seven parameters seems favorable. However, our results show a reversed effect for the similar logical scenarios in the second data set and the same settings. These contradicting observations show the difficulty of finding the best settings for the proposed clustering approach and further indicate that the parameters of our collected data heavily impact the clustering results. Thus, we need to ensure that we consciously acquire data for this approach and explore the effect of different parameter combinations of this data. When inspecting the clustering results generated based on the distortion values, we can find a more stable behavior in the total number of clusters and their sizes when applying different settings. In addition,

in Fig. 3.2, we can observe only a slight difference between the results for KMeans and Hierarchical clustering on data with seven parameters. Nonetheless, these different settings produce various clustering results that might be relevant when automatically acquiring logical scenarios from collected real-world flight data. Thus, we encourage researchers to apply the presented clustering approach to various settings and consider all generated results when deriving logical scenarios for testing the safe behavior of UAVs. Finally, we encounter the problem of acquiring specific logical scenarios from the created clusters, which is currently an active research area [62, 83, 127]. Here the question arises which parameters of the data we should consider and whether all concrete scenarios in a cluster provide the intervals for these parameters or only those instances at the cluster's center. In addition, when our data does not include any parameters about the environment of the UAV, we might not be able to represent the derived logical scenario adequately. Even though environment data is missing in the recorded flight data, the clustering algorithm might still be able to see different effects of the environment on the UAV's behavior and cluster the data accordingly. However, we cannot trace these effects back when describing logical scenarios when we lack the required information about the environment.

Overall, we showed the applicability of the proposed method for clustering collected data to acquire logical scenarios. During our experiments, we discovered several open research challenges that we currently face for the presented technique: (1) we need high amounts of relevant and diverse data for our SUT that we currently lack to complement logical scenarios based on mental models, (2) since various settings for the clustering approach provide different results, we need to either define a measure to describe the quality of the generated clustering or consider the results of all reasonable settings when deriving logical scenarios, (3) since the characteristics of the collected data heavily influence the clustering results, we need to consciously acquire this data for our use case, and (4) missing information about the environment of the UAV might complicate explicitly describing the derived logical scenarios. Due to these presented open research challenges, we set our research focus on acquiring logical scenarios based on mental models, as suggested in [11, 82], which we introduce in the second part of this chapter.

3.3. Systematic Derivation of Logical Scenarios Based On Mental Models

When collecting logical scenarios for UAVs based on mental models randomly, it gets challenging to acquire a complete list of relevant logical scenarios for a SUT. To ease this challenge, we propose a systematic methodology for the derivation. Note, however, that a systematic method does not entail completeness by itself. In this section, we present a systematic approach for building an ontology to describe relevant logical scenarios for a SUT. We believe that the derived ontology will not be static but will change over time when discovering new hazardous situations or new applications for the SUT.

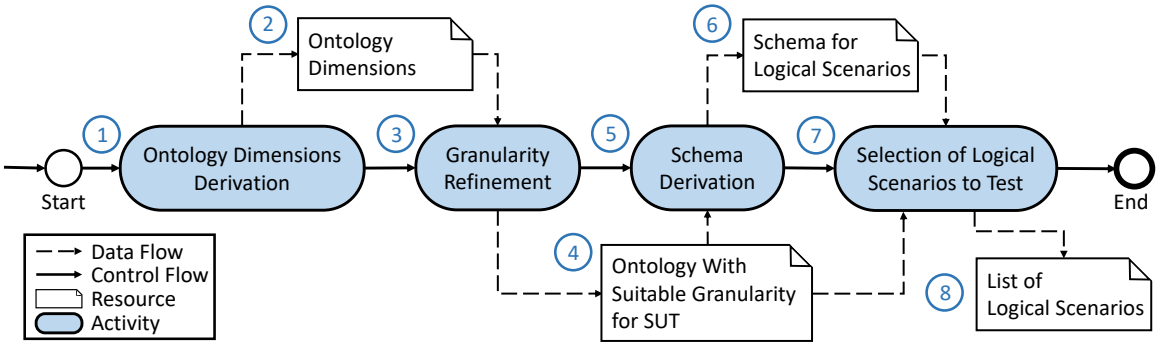


Figure 3.3.: Methodology for systematically deriving logical scenarios for testing the safe behavior of UAVs. A previous version appeared in [118].

3.3.1. Methodology

Figure 3.3 presents the proposed methodology. First, we describe the relevant dimensions of logical scenarios for UAVs with an ontology ①. In this step, we emphasize which aspects are essential when testing the safe behavior of UAVs, e.g., characteristics of the tested UAVs, their mission, and their environment. As a result of this step, we gain dimensions for an ontology that characterizes logical scenarios for UAVs ②. In the next step ③, we need to refine the generated outline of an ontology for logical scenarios. By defining suitable sub-categories for each dimension, we specify a detailed and adjusted ontology for logical scenarios for the SUT ④. An example of such a suitable sub-categorization is the refinement of the wind force as one dimension of the UAVs' environment. Different wind forces will be challenging for (1) a small UAV that operates near the ground or (2) a large UAV that flies in high altitudes. For system (1), light wind might be represented by wind forces of 1 to 4 kilometers per hour, while for system (2), wind forces of 1 to 20 kilometers per hour might present light wind conditions. By providing a system-specific and fine-grained categorization of different wind forces, we can test each system in wind situations that are challenging for this system. After creating the system-specific ontology, we need to provide a way of writing down specific logical scenarios. To this end, we derive a JavaScript Object Notation (JSON) or Extensible Markup Language (XML) schema from the ontology ⑤ in the next step. The generated schema ⑥ enables us to describe specific logical scenarios and verify that the created specific logical scenarios are valid with respect to the provided ontology. The number of all parameter combinations quickly becomes intractable. In step ⑦, we hence select and write down specific logical scenarios ⑧ in which we aim to test the UAV's safe behavior. We can base this selection process on different general defect hypotheses, which influence the completeness of the collected challenging situations for the SUT.

3.3.2. Application to a Quadcopter

In this subsection, we apply the described approach to systematically derive logical scenarios for a quadcopter as one kind of UAV which, e.g., transports a package or monitors an area. We show the results of each step of the methodology presented in Fig. 3.3.

Derivation of Ontology Dimensions

First, we derive the ontology dimensions characterizing logical scenarios for UAVs. This step is mainly generic for different types of UAVs and needs little adaptation for specific systems. In this work, we acquire these ontology dimensions from the literature, expert knowledge about hazardous situations for UAVs, and essential aspects for testing these systems. We categorize the derived dimensions into system- and environment-related ones.

System-related dimensions specify details about (1) the maneuvers and failures of the UAVs present in the logical scenario and (2) their cooperation if existing. Since various maneuvers lead to different flight trajectories, we need to test if the UAV behaves safely in each maneuver that it might perform during operation. In addition, we need to acknowledge that UAVs will encounter failures of different types and degrees. A safely behaving UAV should handle such failures and still present a safe behavior by, e.g., opening a parachute to land. When defining logical scenarios for cooperative UAVs, we further need to describe the cooperation between the UAVs. Based on several works [20, 40, 94, 138, 145] that categorize cooperation mechanisms for autonomous robots, we derive the following sub-dimensions: Communication, Coordination, Organization, and Knowledge. The type of cooperation will influence the behavior of each UAV in the group of UAVs. Each UAV might take different actions or flight trajectories depending on the cooperation type. Consequently, we need to take these various aspects into account when testing the safe behavior of cooperative UAVs.

Environment-related dimensions define the characteristics of (1) the flight area, (2) the present obstacles, and (3) the weather that occurs during the logical scenario. We can describe the flight area in which the UAV performs its mission by its landform and surface nature. Since various landforms such as elevation or depression can cause different challenging situations for the UAV, we need to investigate their impact in various logical scenarios. Further, different types of surfaces such as water or land might present varying difficulties for the UAV. These diverse circumstances might compromise sensor readings and, thus, should be included when testing the safe behavior of UAVs. In addition, we specify the included obstacles by defining their kind, size, and form. A UAV that avoids an obstacle, e.g., by flying around it, changes its flight trajectory compared to its regular one. For situations that the UAV encounters after avoiding an obstacle, this changed trajectory might lead to an unsafe behavior of the UAV. Thus, we need to test the UAV in logical scenarios with different obstacles to enable thorough testing of the UAV in all situations that it might encounter. We base the sub-dimensions for describing the weather of logical scenarios on well-known weather hazards found in the literature. We can refine the weather in a logical scenario into lighting conditions [63], wind force [2, 14, 63, 128], ambient

temperature [2, 128], precipitation [2, 14, 63], cloud coverage [63], reduced visibility [63], and the presence of lightning [2, 14, 63]. The weather conditions might impact the sensor readings or rotor capabilities of the UAV. Due to low light or heavy rain, the UAV might not detect obstacles timely to avoid them. This example shows that we need to consider these conditions when testing the safe behavior of UAVs that we aim to apply outdoors and that might encounter these conditions. Note that we acquired these ontology dimensions based on existing literature and expert knowledge about relevant dimensions for logical scenarios for quadcopters. Additional dimensions may be suitable to describe these logical scenarios, such as the wind direction or wind gusts. Since we view the generated ontology as a "living model", we can adjust and expand these dimensions to newly discovered challenging situations and circumstances for the SUT. Thus, we do not claim to represent all relevant dimensions for all kinds of UAVs with the presented ones but, instead, aim to provide an overview of relevant ones that we can adapt for the SUT.

System-Specific Granularity Refinement

In the next step, we derive sub-categories for each dimension with a suitable granularity for a quadcopter. Note the need to refine these dimensions system-specifically since different environments are challenging for various systems. A small UAV, for example, can cope with only light wind conditions, while a larger one might be able to handle heavier wind conditions. We can derive these sub-categories from expert knowledge, specifications, and existing regulations relevant to the SUT. Figures 3.4 and 3.5 depict the derived fine-granular ontology for a quadcopter.

Depending on the concrete use cases of the UAV, experts might find various maneuvers and possible failures relevant for their SUTs. For inspiration, [14] provides a list of maneuvers and different types of failures for multiple UAVs such as *motor*, *rudder*, *aileron*, *GPS*, or *control loss* failures. Further, we can simplify the presented maneuvers to the following basic maneuvers: *Take Off*, *Hover*, *Landing*, *Move to Waypoint*, and *Exploration Mission*. For describing the cooperation dimension of the ontology, we rely on several works [20, 40, 94, 138, 145] about cooperation mechanisms for autonomous robots that present the four sub-dimensions: Communication, Coordination, Organization, and Knowledge. When UAVs cooperate, they can communicate *directly*, *indirectly*, or *not* at all. The indirect communication can occur via a base station [145], stigmergy [20, 40, 94, 138], or sensing [20, 94, 138]. The coordination sub-dimension describes the influence of the UAVs' actions on the behavior of the other UAVs. If coordination takes place, we can perform it *with* or *without* a given coordination protocol [40]. Further, the organization of cooperative UAVs can either be *centralized*, *decentralized*, or a *hybrid combination*. In a centralized organization, a predefined or dynamically selected leader UAV provides commands to the other UAVs. In contrast, UAVs decide how to fulfill the mission autonomously in a decentralized organization. The UAVs build groups with local leader UAVs in a hierarchical organization, which the authors of [20, 138] regard as a decentralized organization, while [94] considers it a hybrid approach. Finally, in [20, 40], the authors distinguish cooperation mechanisms

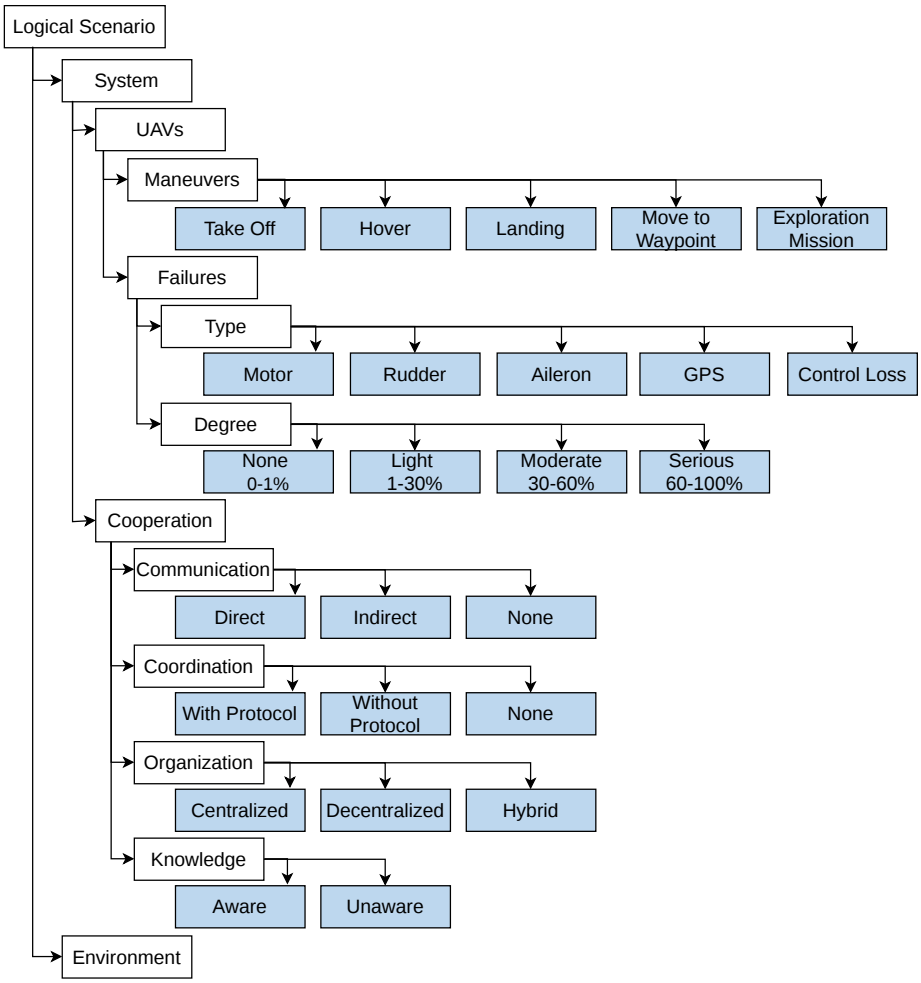


Figure 3.4.: The first of two parts of the derived ontology with a suitable granularity level for a quadcopter. A previous version appeared in [118].

on whether the systems are *aware* or *unaware* of each other. If a UAV knows about the intentions and actions of other UAVs, the UAV can derive possible subsequent actions and flight maneuvers of the other UAVs.

For the environment dimensions, we group the various landforms described by [79] into the following four categories, excluding underwater and underground ones: *flat*, *depression*, *elevation*, and *steep transition* describing, e.g., a cliff. In addition, we describe the surface nature of the flight area as primarily *land*, mainly *water*, or a *mixture* of these two options. In Fig. 3.6, we present exemplary worlds for the presented sub-categories landform and surface nature. For the kind of obstacles, we consider *static* obstacles placed at a specific point in the simulation world and *dynamic* obstacles defined by a starting position, target

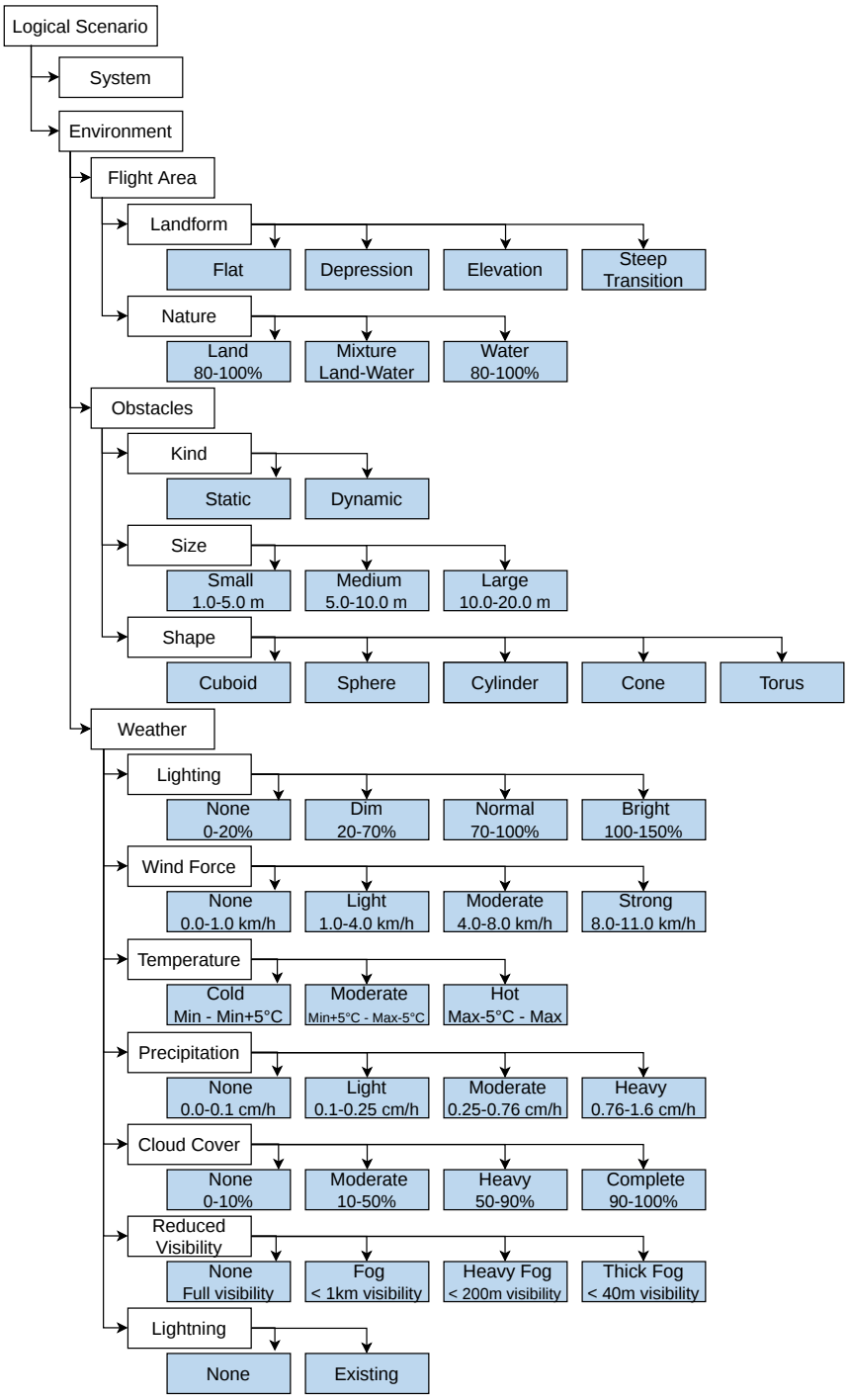


Figure 3.5.: The second of two parts of the derived ontology with a suitable granularity level for a quadcopter. A previous version appeared in [118].

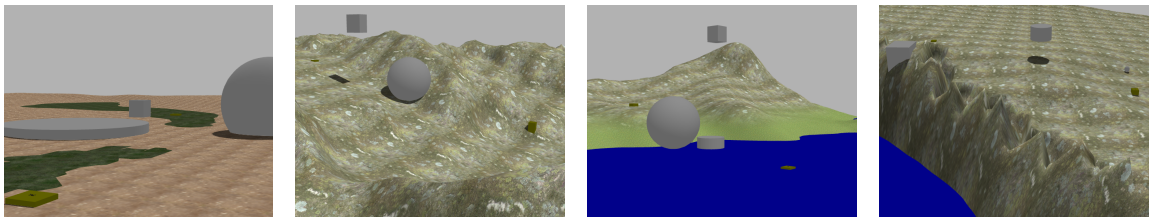


Figure 3.6.: Exemplary worlds for the derived sub-categories landform and surface nature: flat/land, depression/land, elevation/water, and steep transition/water.

position, and velocity. Considering the size of obstacles, we decided to distinguish between *small*, *medium*, and *large* obstacles. Note that the size of each of these categories is dependent on the size of the UAV. Considering the obstacle's form, we approximate it to primitive shapes such as *cuboids*, *spheres*, *cylinders*, *cones*, and *tori*, as mentioned in [119], for example. For the lighting options, we base our categorization on common sense and regulations such as AC 90-89B, which specifies how to perform flight tests for testing amateur-built aircraft. We distinguish lighting into *none*, which equals a flight at night, *dim*, which represents dawn or twilight conditions, *normal*, which presents normal light conditions during the day, and *bright*, which denotes bright artificial light or light reflecting from snow. As a basis for finding sub-categories for the wind force dimension, we can use the Beaufort Scale [44], which categorizes different wind speeds. For our ontology, we assume that the considered quadcopter loses control at wind forces larger than 11.0 kilometers per hour. Thus, we adjust the categorization from the scale to a suitable level: *none* stands for a wind force between 0.0 and 1.0 kilometers per hour, *light* describes a wind force between 1.0 and 4.0 kilometers per hour, *moderate* presents a wind force between 4.0 and 8.0 kilometers per hour, and *strong* represents wind forces between 8.0 and 11.0 kilometers per hour. Since each UAV is certified for a given range of ambient temperatures (Min, Max), we recommend using limit testing for this dimension. Limit testing focuses on testing parameter values at the borders of intervals. The category *cold*, thus, represents an operating temperature between Min and $Min + 5$ degrees Celsius, the category *moderate* denotes operating temperatures between $Min+5$ and $Max-5$ degrees Celsius and the category *hot* defines an operating temperature between $Max - 5$ and Max degrees Celsius. In addition, we follow the suggestion in [3] to sub-divide the precipitation dimension into *none* denoting no precipitation, *light* with a precipitation rate below 0.25 centimeters per hour, *moderate* with a precipitation rate between 0.25 and 0.76 centimeters per hour, and *heavy* with a precipitation rate above 0.76 centimeters per hour. For the cloudiness of the sky, we propose to apply limit testing again. We consider the two extremes of *none* and *complete* cloud cover to investigate effects such as a potentially impaired GPS connection. For realistic scenarios, we further suggest adding two categories between these extremes: *moderate*, which represents a sky filled up to 50% with clouds and *heavy* for a sky covered by clouds for more than 50% but without full cloud coverage. Further, we adopt the idea of [121] to divide the reduced visibility dimension

into four categories. *None* denotes a clear view with full visibility, *fog* describes a visibility below 1 kilometer, *heavy fog* represents a visibility below 200 meters, and *thick fog* presents a visibility below 40 meters. Finally, we distinguish between *none* lightning present and lightning *existing*.

In this work, we aim to provide a comprehensive description of relevant dimensions that characterize logical scenarios for a quadcopter as represented in the literature by collecting all presented dimensions for the ontology based on existing specifications and literature. In addition, we enable the adaptation of the derived dimensions by regarding the built ontology as a "living model" and encourage the community to change it according to newly arising relevant dimensions for their SUTs. Currently, we utilize constant values to describe the categories in the presented ontology, e.g., moderate wind denotes a constant wind force in the range of 4.0 to 8.0 kilometers per hour. We can further change these constant values to functions to, e.g., represent wind gusts with varying wind speeds in this range over time.

Schema Derivation

To enable machine processing of logical scenarios for the provided ontology, we derive a JSON schema of the ontology in the next step. We decided to use JSON due to its simple syntax, its option to present various data types, and its easy and fast parsing possibilities. Note that other formats such as XML are also valid options in this step. Figure 3.7 provides an excerpt of the resulting JSON schema that focuses on describing the flight area present in the logical scenarios. We depict the complete JSON schema in Appendix A. As presented in Fig. 3.7, JSON enables us to specify the needed attributes of each dimension with the *required* keyword and their possible values with the *enum* keyword. With these options, we can limit the selected possible attributes of specific logical scenarios and, thus, validate the correctness of a given logical scenario with respect to the provided ontology.

Selection of Logical Scenarios

As a final step, we need to choose specific logical scenarios for testing our SUT from the created ontology since it is infeasible to consider all possible combinations of the ontology dimensions. As mentioned earlier, we can use different defect hypotheses when selecting logical scenarios from the ontology. There exist various general defect hypotheses, e.g., pair-wise testing of the ontology elements and UAV-specific defect hypotheses describing challenging situations for UAVs. However, formulating an adequate defect hypothesis for a SUT is challenging and is currently still an open research area. Thus, for the purpose of this work and simplicity of presentation, we use two simple defect hypotheses in this work. Table 3.3 shows the derived logical scenarios for these two simple defect hypotheses about challenging situations for UAVs: (1) testing each sub-category of the ontology once, and (2) testing the high impact of regional weather effects. Defect hypothesis (1) assumes that it is sufficient to reach complete conditional coverage over all sub-categories of the ontology to provoke all faults in the SUT and selects logical scenarios accordingly. Thus, for this

```
{ ..., "flight area": {
        "type": "object",
        "properties": {
            "landform": {
                "type": "string",
                "enum": ["flat", "depression", "elevation",
                 ↪ "steep_transition"]
            },
            "surface nature": {
                "type": "string",
                "enum": ["land", "mixture", "water"]
            }
        }, "required": ["landform", "surface nature"],
        "additionalProperties": false
}, ...}
```

Figure 3.7.: Excerpt of the derived JSON schema for specifying logical scenarios for a quadcopter, which focuses on the flight area dimension. A previous version appeared in [118].

defect hypothesis, we derive, e.g., one logical scenario with a flat landform (E1) and one with strong wind (E4), as we assume each of these environmental effects to independently influence the SUT's behavior. However, we do not expect the SUT to encounter other faults when combining these specific environmental effects. The presented logical scenarios based on the second defect hypothesis capture the following scenarios inspired by [37], who state the difficulty of regional weather effects for UAVs: (R1) stormy weather with strong wind and thick fog, (R2) coastal weather at a cliff with moderate wind, (R3) mountain weather in an elevation landscape with light wind and fog, (R4) desert weather with moderate wind, (R5) weather over a flat landmass with light wind, and (R6) weather over the sea represented by a water surface and light wind. This defect hypothesis states that the SUT shows all its faults when encountering regional weather effects, e.g., stormy weather (R1) might lead to a malfunction of one of the rotors, and mountain weather with fog and an elevation landscape (R4) might corrupt the sensor data, which might delay the detection of obstacles. Further, we present logical scenarios based on the defect hypothesis that pair-wise testing of the ontology's elements is sufficient in Section 7.4.

We need to stress that the chosen logical scenarios only represent situations based on the used defect hypothesis. Thus, we need to ensure that the applied defect hypothesis represents all challenging situations for the SUT. In this work, we only show simple defect hypotheses for several dimensions of the presented ontology and, thus, do not claim

Table 3.3.: The derived logical scenarios based on two different defect hypotheses that focus on the environment-related dimensions. We present the parameters landform (flat F, elevation E, depression D, or steep transition ST), surface nature (land L, water W, or a mixture M of them), wind force (none N, light L, moderate M, or strong S), reduced visibility (none N, fog F, heavy fog HF, or thick fog TF), and various parameters about the included obstacles (static ST or dynamic DY; small S, medium M, or large L; cuboid CU, sphere SP, or cylinder CY) of these logical scenarios. A previous version appeared in [118].

	(1) Each Category Once				**(2) Regional Weather Effects**					
Scenario	E1	E2	E3	E4	R1	R2	R3	R4	R5	R6
Landform	F	D	E	ST	D	ST	E	D	F	F
Nature	L	W	M	L	L	M	L	L	L	W
Wind	N	L	M	S	S	M	L	M	L	L
Red. Visibility	N	F	HF	TF	TF	N	F	N	N	F
# Obstacles	4	1	2	3	3	1	2	1	3	1
	DY	ST	DY	ST	ST	ST	ST	ST	ST	DY
Obstacle	ST	—	ST	DY	DY	—	DY	—	ST	—
Kinds	DY	—	—	DY	DY	—	—	—	DY	—
	ST	—	—	—	—	—	—	—	—	—
	M	S	M	S	L	S	L	M	L	S
Obstacle	S	—	L	M	S	—	S	—	L	—
Sizes	S	—	—	L	S	—	—	—	M	—
	L	—	—	—	—	—	—	—	—	—
	SP	CU	SP	CY	CU	CY	CY	CU	CU	SP
Obstacle	CU	—	CY	SP	SP	—	SP	—	CU	—
Forms	CY	—	—	CU	CY	—	—	—	SP	—
	CU	—	—	—	—	—	—	—	—	—

to acquire a comprehensive list of relevant logical scenarios from these hypotheses. As mentioned before, the derived logical scenarios from the first defect hypothesis include one logical scenario with a flat landform (E1) and one with strong wind (E4). However, we might not provoke all failures of the SUT with these logical scenarios if the underlying defect hypothesis is incorrect and, e.g., the specific combination of a flat landform and strong wind leads to an unsafe behavior of the SUT. This observation indicates that only if we can ensure (1) the correctness of the applied defect hypothesis for the SUT and (2) the completeness of the created ontology, the derived logical scenarios based on this defect hypothesis represent all challenging situations for the SUT.

3.4. Conclusion

We can derive logical scenarios for testing the safe behavior of UAVs (1) automatically with clustering techniques or (2) based on mental models. Ideally, we gain logical scenarios from both approaches that complement each other and present a comprehensive set of relevant logical scenarios for the SUT. When applying the first approach, we need high amounts of diverse and relevant real-world flight data, which we currently lack for testing UAVs. Nonetheless, we show the general idea of using these clustering techniques to acquire logical scenarios with simulated data and point out their challenges in the first part of this chapter. When researchers apply these approaches, we strongly encourage them to execute an analysis for the different parameters of the collected data and the clustering approach, such as the used clustering algorithm or metric for finding an optimal number of clusters. In our experiments, we present such an analysis on a small scale and based on simulated data to outline the currently existing research challenges. We investigate the results of applying the clustering algorithms KMeans and Agglomerative Hierarchical Clustering and the two metrics Silhouette Score and distortion values for finding an optimal number of clusters on data with 7 or 20 parameters. The experimental results present the general applicability of clustering techniques to acquire logical scenarios and outline several open research challenges for applying them: (1) we need high amounts of diverse and relevant real-world flight data that we currently lack, (2) as we currently miss a quality measure for the generated clustering, we either need to define such a measure or consider the results for all suitable settings, (3) as the characteristics of the collected data heavily influence the clustering results, we need to consciously acquire this data for our use case, and (4) if we lack information about the UAV's environment, explicitly describing the derived logical scenarios gets more complicated. Due to these presented open research challenges, we set our research focus on acquiring logical scenarios based on mental models in the second part of this chapter.

In this second part, we outline the challenges of deriving logical scenarios for testing the safe behavior of UAVs: (1) finding the relevant dimensions for logical scenarios, (2) defining a suitable level of granularity for them, (3) writing them down, and (4) combining the found dimensions to derive a manageable number of logical scenarios. To address several of these challenges, we present an ontology characterizing logical scenarios for a quadcopter in this work. Further, we describe how we systematically derive such an ontology to enable test engineers to build similar ontologies for their UAVs. Note that we consider the presented ontology a "living model" that we can adapt to newly discovered challenging situations or applications. First, we collect the dimensions of logical scenarios for UAVs. After this generic step, we refine these dimensions to a suitable level of granularity for the SUT. Note the need to perform this refinement step system-specifically since different environments are challenging for various systems. Next, we derive a JSON schema that represents the ontology to enable writing down logical scenarios and verifying them with the built ontology. Finally, we can apply a selection method to derive specific logical scenarios from the created ontology. Depending on the correctness of the defect hypothesis on which we

base this selection method, we might be able to further build a completeness argument over the list of derived logical scenarios. Clearly, there is a trade-off between completeness and practicability: If we use an inadequate defect hypothesis as a basis for selecting logical scenarios, we might miss relevant situations for testing the SUT. However, if we can ensure (1) the correctness of the used defect hypothesis for the SUT and (2) the completeness of the generated ontology, we derive logical scenarios that represent all relevant situations for the SUT.

In future work, we would like to investigate the open research challenges for automatically deriving logical scenarios and investigate the correctness of various defect hypotheses for selecting specific logical scenarios from an ontology to enable deriving a comprehensive list of logical scenarios for UAVs in the future. In addition, we aim to generate ontologies for other types of UAVs and compare their similarities and differences.

4. Exploration of Bounds for the Ontology's Dimensions

This chapter presents an automated approach for finding reasonable bounds for the parameter values that each dimension of an ontology for logical scenarios for UAVs describes. Further, it illustrates the approach with the example of exploring an upper bound for the number of relevant obstacles to include in logical scenarios. Parts of this chapter previously appeared in a peer-reviewed publication [115] co-authored by the author of this thesis.

4.1. Introduction

When building an ontology that characterizes logical scenarios for testing the safe behavior of UAVs, we need to system-specifically divide the discovered dimensions into sub-categories to represent various challenging situations for the SUT. For deriving these sub-categories, we first need to define reasonable lower and upper bounds for the parameter values of each dimension of the ontology. We can acquire these lower and upper bounds (1) from expert knowledge and specifications or (2) from experimental results for decreasing or increasing parameter values. An example of the first case is the temperature dimension. The developers of a UAV often specify the temperature range in which one can operate the UAV, e.g., 0 to 35 degrees Celsius. This information provides the lower and upper bounds of the parameter values for the temperature dimension. Next, we can divide the specified range of values into suitable sub-categories for the SUT, e.g., *cold* denoting 0 to 5 degrees Celsius, *moderate* presenting 5 to 30 degrees Celsius, and *hot* describing the range 30 to 35 degrees Celsius. If we cannot acquire the lower and upper bounds from specifications or expert knowledge, we need to find them by performing experiments with different bounds for the parameters to investigate their influence on the UAV's behavior. The lower bounds for the dimensions are normally given by specifications or intuition, e.g., 0 kilometers per hour representing the lower bound of the wind force dimension, 0 centimeters per hour presenting the lower bound of the precipitation dimension, or 0 obstacles as the lower bound for the number of obstacles to consider. On the contrary, we cannot easily specify upper bounds for several dimensions, such as the upper bound for the number of relevant obstacles or the number of wind directions to consider in logical scenarios for UAVs. Thus, we need to evaluate the impact of increasing numbers of these entities on the UAV's behavior to find reasonable upper bounds for these dimensions.

The **contribution of this chapter** is an automated approach for finding bounds for the parameter values of the ontology's dimensions presented with the example of finding an upper bound for the number of relevant obstacles for our SUT. By defining reasonable lower and upper bounds for the dimensions of the ontology, we effectively limit the number of logical scenarios to test. We show the applicability of the presented approach in experiments with two optimization algorithms and when recording different parameter values to represent the UAV's behavior.

4.2. Automated Derivation of Bounds

As mentioned before, we use the exploration of a maximal number of relevant obstacles for the SUT as our example for presenting our proposed approach for finding bounds for the ontology's dimensions. When deriving a maximal number of relevant obstacles for the SUT, we focus on those obstacles that impact the UAV's behavior and trajectory planning. Thus, we can dismiss obstacles located, e.g., 100 meters away and concentrate on those residing in the surrounding area around the UAV. Since the UAV moves through its environment during its mission, the relevance of specific obstacles varies over time. However, the maximal number of relevant obstacles at each point is stable at any time. Even though we might need fewer obstacles to provoke challenging situations for the SUT in specific circumstances, we are interested in the maximum number of obstacles that generally influence the UAV's behavior here. Note that the relevance of an obstacle does not only depend on its distance to the SUT but might also depend on other factors such as the UAV's size, type, or velocity. To experimentally explore an upper bound for the number of relevant obstacles, we inspect the impact of a varying number of N obstacles on the UAV's behavior. We perform these experiments independent of a given logical scenario as we aim to investigate the impact of obstacles on the SUT and do not focus on testing the safe behavior of the SUT directly. After our approach yields a maximal number of relevant obstacles M, we can build logical scenarios with 0 to M obstacles for the SUT, including various environmental effects and missions, as presented in the previous chapter. In this way, we derive a suitable upper bound for the obstacle dimension and limit the number of logical scenarios to test by excluding those with more than M obstacles.

4.2.1. Black-Box Description of the UAV's Behavior

When inspecting the impact of obstacles on the UAV's behavior, we consider black-box descriptions of its behavior in this chapter. Thus, we concentrate on the system states of the UAV that we can observe externally. The pitch, roll, and yaw values of a UAV describe its orientation and are one example of such externally observable system states. One reason for using these parameter values in our experiments for finding an upper bound for the number of relevant obstacles is the defect hypothesis that extreme orientation values present challenging situations for the SUT. Depending on the SUT, this defect hypothesis might be

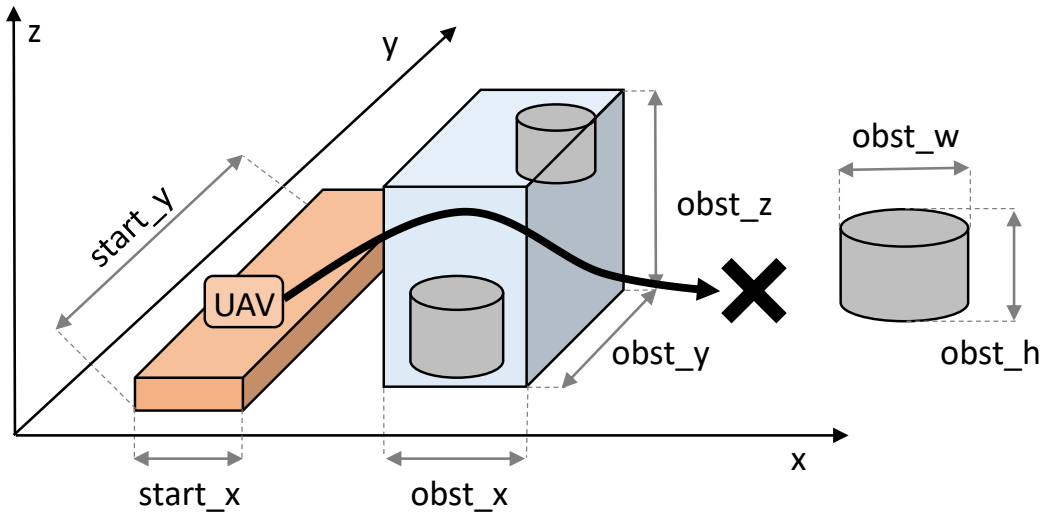
correct and suitable as (1) extreme orientations of the UAV might lead to instability and loss of control, and (2) the UAV needs to perform more extreme orientations when avoiding obstacles in its environment. However, when testing a UAV that should perform maneuvers with extreme orientations as part of its operation, such a defect hypothesis might not be convenient for representing challenging situations for this system. In such a case, other parameter values might be more suitable to discover demanding situations for the SUT, such as the UAV's linear velocity, which describes the UAV's speed along a straight line, its angular velocity, which represents the change in the angle that the UAV covers in a given time, or its acceleration, which presents the rate of change of the UAV's velocity over time. Independent of the concrete parameter values that we use to describe the UAV's state, we investigate how a varying number of obstacles influences the range of these externally observable values. If additional obstacles expand the range of the observable values, they have an impact on the UAV's behavior and might introduce challenging situations to the SUT. Thus, we explore whether additional obstacles force the UAV into, e.g., new extreme orientation or linear velocity values depending on the applied defect hypothesis. We need to emphasize that only if we use a correct defect hypothesis, which describes parameter values that represent challenging situations for the SUT, we can utilize the resulting upper bound for the number of obstacles in logical scenarios to test the safe behavior of this SUT. Otherwise, we cannot infer that the derived maximum number of relevant obstacles is suitable when we generate "good" test cases for the SUT in the next step, as described in Chapter 5. In our experiments, we present the independence of our proposed approach from the specific parameter values used by showing results for orientation and linear velocity values. However, to simplify explanations, we concentrate on orientation values as an example for externally observable values in the remainder of this section.

4.2.2. Methodology

With our proposed approach, we evaluate the impact of various starting positions for the UAV and differently located obstacles on the UAV's behavior. We assess this impact by collecting the orientation values of the UAV for an increasing number of obstacles N. In our approach, we apply an optimization algorithm to discover situations in which the UAV presents new extreme orientation values. To find these challenging situations, we first define the search problem with its search space and corresponding fitness function. Next, we present an overview of our process for finding bounds for the ontology's dimensions.

Search Space & Fitness Function

To derive a maximal number of relevant obstacles for the SUT, we need to investigate the UAV's behavior when encountering varying amounts of obstacles. The corresponding search space represents all possible situations for the UAV for each number of obstacles. In Fig. 4.1, we visualize a suitable search space for finding a maximal number of obstacles to consider in logical scenarios. For each parameter of the search space, we define a parameter



Param.	start_x	start_y	obst_x	obst_y	obst_z	obst_w	obst_h
Range	[-4.0, -2.0]	[-5.0, 5.0]	[4.0, 8.0]	[-4.0, 4.0]	[0.0, 5.0]	[0.5, 4.0]	[1.0, 5.0]

Figure 4.1.: Visualization of an exemplary search space for exploring the maximal number of relevant obstacles with our proposed approach. The UAV starts in the left area with the mission to fly to the target point marked with an *X* while avoiding the obstacles in the middle area. A previous version appeared in [115].

range to span a multi-dimensional space of possible situations that the SUT can encounter for different numbers of obstacles. The presented search space includes parameters to modify the starting position of the UAV and the position and size of the obstacles in a specified area. For simplicity of presentation, the utilized search space contains only static obstacles. However, we can extend this search space to consider also dynamic obstacles by including parameter ranges for their trajectories and velocities. To detect situations in this search space that lead to extreme orientation values and, thus, denote an impact of the obstacles on the UAV's behavior, we need to define a fitness function that finds these situations. We collect all discovered parameter values in a convex hull h to find situations that enforce new extreme parameter values. For each candidate c, we build a new convex hull $h_{new}(c)$ that includes the existing hull h and the orientation values of the UAV in this candidate. Then, we compare the volumes v of these two hulls in our fitness function f_{bounds} to discover whether the candidate forced the UAV into new extreme orientation values:

$$f_{bounds}(c) = v(h_{new}(c)) - v(h) \tag{4.1}$$

When searching for promising candidates, we strive to maximize this fitness function to discover new extreme parameter values. Note that this fitness function does not directly

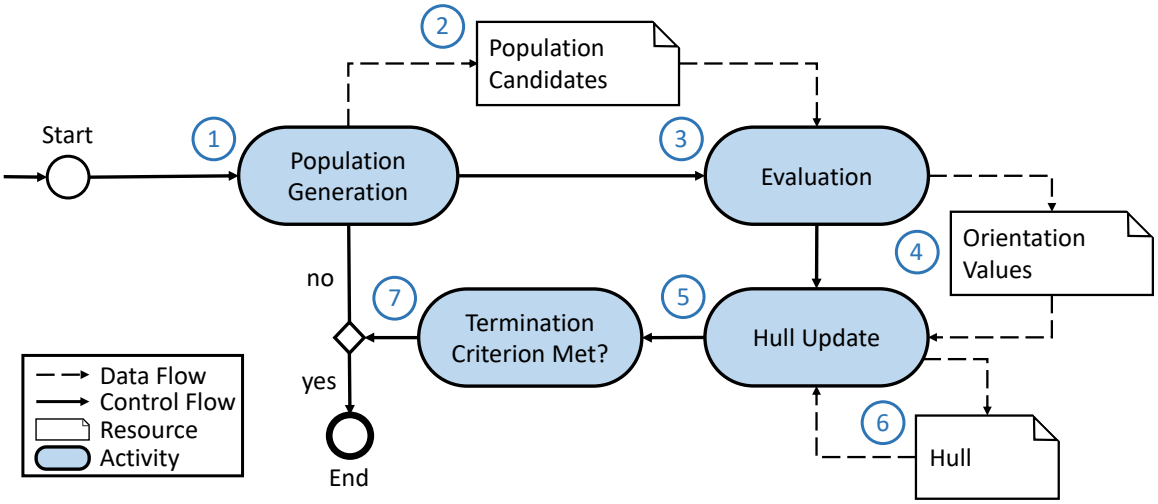


Figure 4.2.: Process overview of our proposed approach for finding bounds for the ontology's dimensions. A previous version appeared in [115].

depend on the collected parameter values and that we can use it for various parameter values such as orientation values or linear velocity values. Further note that we can use other fitness functions than the one presented in Eq. (4.1) to find challenging situations for the UAV, e.g., one that maximizes the distance of an entire population to the existing hull or one that maximizes the average distance of the parameter values to the hull.

Process Overview

To discover the impact of an increasing number of obstacles on the UAV's behavior, we follow the following process that we also depict in Fig. 4.2: for each number of obstacles N, the optimization algorithm generates in step ① a population of candidates ② that represent concrete scenarios from the search space presented in the previous subsection. Next, in step ③, we simulate the SUT in each candidate and collect its orientation throughout the simulation ④. If other parameters represent challenging situations for the SUT, we collect these other parameters in this step. As a final step of evaluating the population's candidates, we compute the quality of each candidate by calculating its fitness value with the fitness function presented in Eq. (4.1). In step ⑤, we update the convex hull ⑥ that contains the orientation values of all evaluated candidates of the last populations by adding the orientation values of the current candidates. Finally, the optimization algorithm generates a new population if the termination criterion is not yet met ⑦ or stops this process for the currently regarded number of obstacles N. In this work, we apply the following termination criterion: we stop the search for challenging situations if (1) the current population does not produce any new extreme orientation values that reside outside of the hull, (2) no crashes with obstacles occurred in the current population, and (3) we evaluated a minimum

of 500 candidates. Note that we repeat this process for each number of obstacles N to discover a convergence in the results. If we add an additional obstacle $N + 1$, we expect to observe new extreme orientations of the SUT compared with its behavior for N obstacles. However, we will observe fewer and fewer new extreme orientations for a rising number of obstacles. When we do not discover any new extreme orientations for additional obstacles, these further obstacles do not have an impact on the UAV's behavior. As an advantage of our proposed approach, we only need to apply this process once for each system version to discover appropriate bounds for the ontology's dimensions and to generate a limited number of logical scenarios for the SUT.

4.3. Experiments

In our experiments, we demonstrate the applicability of the proposed approach at the example of finding a maximal number of relevant obstacles for the open-source PX4 autopilot for UAVs [81] with the obstacle avoidance extension. We evaluate the performance of two optimization algorithms for our approach, namely NSGAII [32] and MOEA/D [143]. We assess NSGAII since it generally performs well, as presented in [1, 6], and evaluate MOEA/D as the benchmark of [45] suggests it as the best performing algorithm for constraint dynamic problems. Since our search problem is a dynamic problem, this is a suitable choice for evaluation. In addition, we show the results for collecting orientation values or linear velocity values to demonstrate the applicability of our approach independent of the chosen parameter values that represent challenging situations for the SUT.

4.3.1. Setup and Implementation

During our experiments, we perform a Software-in-the-Loop simulation of the PX4 autopilot in the simulator Gazebo [61], in which it autonomously flies to a specified target point while avoiding obstacles in its path. During these simulations, we collect the UAV's orientation by observing its roll, pitch, and yaw value or its linear velocity in x-, y-, and z-direction via the Robot Operating System (ROS). For the implementation of the optimization algorithms MOEA/D and NSGAII, we use the jMetalPy framework [15] with Tschebycheff as the aggregation function for MOEA/D and SBX Crossover and Binary Tournament as the crossover and selection operators for NSGAII. In pre-experiments, we evaluate the performance of these algorithms with population sizes of 25, 50, and 100. As the population size of 100 showed the best performance in these pre-experiments, we apply this population size in our presented experiments. During the experiments, the optimization algorithms pick concrete scenarios from the search space displayed in Fig. 4.1 and apply the fitness function in Eq. (4.1) to find challenging situations for the SUT. To enable the reproducibility of these experiments, we provide additional details on the experiment settings as, e.g., the version number of the used libraries, in Appendix B. In our evaluation, we focus on those concrete scenarios in which the UAV is in control and does not crash into any obstacle

without any environmental impact. As we aim to inspect the impact of additional obstacles on the UAV's behavior, we need to enable a clean collection of the chosen parameter values. However, when the UAV loses control or crashes, it produces a random set of parameter values that do not make a statement about the challenge of the situation necessarily. As we use an open-source UAV, this situation might occur randomly without the present obstacles influencing the control loss or crash. Note that if we test the safe behavior of UAVs directly, as we do in the following chapter, these crashes represent particularly interesting situations. However, as we instead investigate a maximum number of obstacles, we are dependent on a clean collection of parameter values to gain insightful results. Thus, we discard the parameter values from those concrete scenarios in which the UAV crashes without environmental impact. Note that for the currently regarded number of obstacles, we still collect parameter values of the UAV from other concrete scenarios and, thus, still detect their effect on the UAV's behavior. Further, we consider the crash information in the termination criterion. Note that we provide the points of all hulls generated in these experiments and their visualizations in [112].

4.3.2. Experimental Results

We show the results of our experiments in Table 4.1. In this table, we represent the performance of MOEA/D and NSGAII when collecting orientation values or linear velocity values of the UAV. For each number of obstacles $N \in \{1, 2, ..., 15\}$, we present the percentage volume increase vi of the new hull h_{new} to the previous hull h. We compute this difference in volumes v with the following formula:

$$vi = \frac{v(h_{new}) - v(h)}{v(h)} \tag{4.2}$$

We do not provide a volume increase for $N = 0$ obstacles as we create the first hull of parameter values for this number of obstacles. The results for orientation values show that MOEA/D finds a maximum of $M = 8$ relevant obstacles, whereas NSGAII does not converge and creates new extreme orientation values for higher numbers of obstacles. These observations indicate that NSGAII cannot detect an upper bound for the number of relevant obstacles when gathering orientation values with the presented methodology. When collecting linear velocity values of the UAV, both algorithms discover a maximum of $M = 5$ relevant obstacles as they produce no more new extreme parameter values for $N > 5$.

4.3.3. Discussion

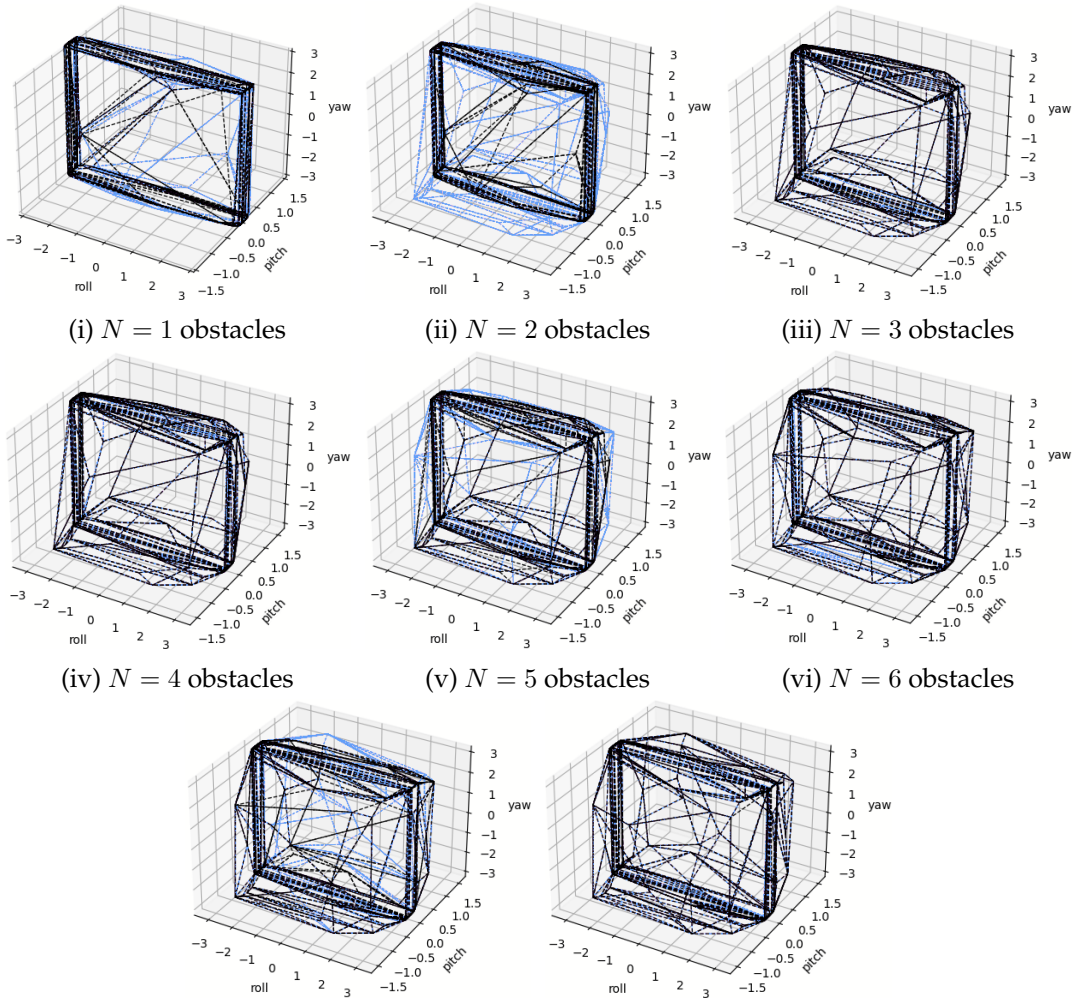
In our experiments, we compare the performance of the optimization algorithms MOEA/D and NSGAII when exploring a maximal number of relevant obstacles for the SUT. When collecting orientation values, NSGAII does not discover a maximal number of relevant obstacles M, whereas MOEA/D results in $M = 8$ relevant obstacles that influence the

Table 4.1.: The experimental result for finding a maximal number of relevant obstacles with MOEA/D and NSGAII while collecting orientation or linear velocity values of the SUT. We denote the percentage volume increases vi [%] for varying numbers of obstacles N. A previous version appeared in [115].

MOEA/D			**NSGAII**		
N	Orientation vi	Lin. Velocity vi	N	Orientation vi	Lin. Velocity vi
1	15.78	8.32	1	147.11	0.08
2	110.49	0.48	2	0.00	0.00
3	0.00	0.00	3	9.37	0.00
4	0.00	0.00	4	14.96	12.07
5	7.95	0.06	5	1.38	28.15
6	0.01	0.00	6	0.00	0.00
7	4.01	0.00	7	5.00	0.00
8	0.01	0.00	8	0.00	0.00
9	0.00	0.00	9	1.39	0.00
10	0.00	0.00	10	0.00	0.00
11	0.00	0.00	11	0.00	0.00
12	0.00	0.00	12	0.31	0.00
13	0.00	0.00	13	0.00	0.00
14	0.00	0.00	14	0.22	0.00
15	0.00	0.00	15	1.26	0.00

SUT's behavior. Note that we acquire this number of relevant obstacles when assuming that extreme orientations represent challenging situations for the SUT as they, e.g., denote potential instabilities of the UAV that might lead to loss of control or avoidance maneuvers of the UAV. To draw conclusions from this provided number, we need to ensure that the presented defect hypothesis holds for the SUT. In Fig. 4.3, we show visualizations of the convex hulls that contain all collected orientation values for $N \in \{1, 2, ..., 8\}$ obstacles created by MOEA/D in our experiments. Note that the number of obstacles impacts the UAV's behavior and, thus, implicitly influences the range of the gathered orientation values. In these plots, we further display the increase in the hull's volume by presenting the hull for $N - 1$ obstacles in black and the hull for N obstacles in blue.

When recording the UAV's linear velocity in our experiments, we discover that both algorithms do not find any new extreme parameter values representing challenging situations for more than $M = 5$ obstacles. When the underlying defect hypothesis is true that challenging situations for the SUT are presented by extreme linear velocities, these results indicate that a maximum of 5 obstacles is relevant for the SUT. Even though both algorithms converge to the same number of relevant obstacles, they reach this boundary differently. NSGAII generates new extreme linear velocity values mainly for $N \in \{4, 5\}$ obstacles,



(i) $N = 1$ obstacles (ii) $N = 2$ obstacles (iii) $N = 3$ obstacles

(iv) $N = 4$ obstacles (v) $N = 5$ obstacles (vi) $N = 6$ obstacles

(vii) $N = 7$ obstacles (viii) $N = 8$ obstacles

Figure 4.3.: Visualization of the convex hulls that present the collected orientation values for MOEA/D for $N \in \{1, 2, ..., 8\}$ obstacles, which implicitly affect the range of the orientation values. In addition, we show the increase in the hull's volume for each N by depicting the hull for $N - 1$ obstacles in black and the hull for the current N obstacles in blue. A previous version appeared in [115].

whereas MOEA/D primarily creates them for $N = 1$ obstacles.

Overall, both optimization algorithms discover challenging situations represented by extreme parameter values and converge to a maximal number of obstacles in our experiments except for NSGAII when collecting orientation values. These results indicate the applicability of the proposed approach for finding bounds for the ontology's dimensions

with the MOEA/D algorithm for the example of discovering a maximal number of relevant obstacles for the SUT. MOEA/D seems to find a maximum number of obstacles more reliably than NSGAII, which enables us to limit the number of logical scenarios we need to test for the SUT and presents a basis for defining a test-ending criterion for testing the safe behavior of UAVs. Further, as the algorithm focuses on discovering extreme parameter values for fewer obstacles, it reduces the cost of finding bounds for the ontology's dimensions, which is essential as the simulations of UAVs take quite some time. The nature of our search problem as a dynamic optimization problem might be one reason for this better performance of MOEA/D. An alternative interpretation of the results would be that there is no upper bound for the maximal number of obstacles when considering orientation values, and MOEA/D misses additional extreme values for higher numbers of obstacles. When inspecting the extreme values collected by both algorithms, we can observe that the volume of the hull built with MOEA/D is 32% larger than the one generated by NSGAII. Further, the data shows us that NSGAII gathers only a small number of extreme values of 0.003% of all discovered values that MOEA/D does not detect. For reference, we include the visualization of the convex hulls that present the collected orientation values for NSGAII for $N \in \{1, 2, ..., 15\}$ obstacles in Appendix C. These observations show that various optimization algorithms perform differently when searching for extreme parameter values and that we should investigate system-specifically which works best for this use case. We encounter similar findings when inspecting the performance of optimization algorithms for generating worst-case situations for different logical scenarios in Section 6.3.

When assuming that MOEA/D works best for our SUT, the experimental results for MOEA/D finally indicate a maximum number of $M = 5$ or $M = 8$ obstacles for the SUT that we need to consider in logical scenarios for this system depending on whether we collect orientation or linear velocity values. Note that the decision of which parameter values are suitable to represent challenging situations is system-specific. Thus, we need to ensure that we base this decision on a correct defect hypothesis for the SUT. By finding an upper bound for the number of obstacles, we demonstrate how we can use the proposed methodology to discover bounds for the ontology's dimensions and effectively limit the number of logical scenarios to test the SUT. These limited value ranges further present a basis for collecting a complete list of relevant logical scenarios for UAVs in the future.

Threats to Validity

The results of our experiments might not generalize to other UAVs as we present results for the PX4 autopilot only. It is essential to pick the collected parameter values for our proposed approach system-specifically and based on a correct defect hypothesis for the SUT to gain insightful results. Note that we focus on static obstacles in our experiments for simplicity of presentation. To reduce the threats to internal validity in our experiments, we use the implementations of MOEA/D and NSGAII from the open-source library jmetalPy and evaluate the open-source PX4 autopilot. In addition, we run all simulations in isolated Docker containers to decrease unwanted side effects.

4.4. Conclusion

When exploring bounds for the ontology's dimensions, we can acquire them (1) from expert knowledge and specifications or (2) experimental results for different bounds. In this chapter, we present an automated approach for the second option when we miss knowledge about appropriate bounds for these dimensions for the SUT. We demonstrate this approach with the example of finding a maximal number of obstacles that we need to consider in logical scenarios for the SUT. In our presented method, we explore the impact of a varying number of obstacles on the UAV's behavior. To determine the relevance of obstacles, we search for challenging situations that are represented by extreme parameter values. Note that it is essential to select these parameter values system-specifically and based on a correct defect hypothesis for challenging situations for the SUT to gain insightful results. In our experiments, we compare the performance of the optimization algorithms MOEA/D and NSGAII and demonstrate that our method can work with various parameter values depending on which ones present challenging situations for the SUT. The experimental results reveal the different performances of the investigated two optimization algorithms when searching for extreme parameter values and the corresponding need to investigate system-specifically which works best for the presented methodology. Finally, the results for MOEA/D show that we need to consider a maximal number of $M = 5$ or $M = 8$ relevant obstacles for the SUT when collecting orientation or linear velocity values. With the resulting reasonable bounds for each dimension of the ontology for the SUT, we can effectively limit the number of logical scenarios in which we need to test the SUT. They further present a basis for collecting a comprehensive list of relevant logical scenarios for UAVs, which we aim to accomplish in the future. In addition, we aim to explore the results of observing additional parameter values that describe the UAV's behavior in future work, such as its angular velocity or acceleration profiles. Finally, we would like to explore the performance of other optimization algorithms for our presented approach and show experimental results for finding a maximal number of dynamic obstacles in the future.

Part III.

Test Case Generation

5. Understanding and Assessment of the Safe Behavior of UAVs

This chapter provides a methodology for testing the safe behavior of UAVs while considering their environment and the potential challenge of explicitly defining the safe behavior of UAVs. Throughout the chapter, we explore the two cases of having a safety distance specified and working with no defined safety distance. Parts of this chapter previously appeared in a peer-reviewed publication [114] co-authored by the author of this thesis.

5.1. Introduction

After deriving logical scenarios for testing the safe behavior of UAVs, we generate test cases for each of them in the next step. Note that we first need to define logical scenarios, as the number of parameters for the search problem of generating test cases would otherwise be too high. Since we cannot provide a thorough safety argumentation by randomly generating test cases for each of these logical scenarios, we aim to create so-called "good" test cases that can reveal *potential* faults in the tested UAV. These "good" test cases represent challenging situations for the UAV, which we also call worst-case situations. In these worst-case situations, a correct UAV behaves safely, while a faulty UAV operates unsafely by, e.g., violating specified safety distances. The general idea of creating "good" test cases for each logical scenario is the following: If the UAV behaves safely in all generated test cases of a logical scenario — even in the worst-case situation — it will always behave safely when encountering this situation, given that we found the worst-case situation. However, due to missing regulations and the wide range of possible maneuvers and missions for UAVs, it is not trivial to explicitly specify the safe behavior of UAVs in all possible situations. Thus, a methodology for generating "good" test cases for testing the safe behavior of UAVs is needed, which takes into account that we might not always be able to explicitly define the UAV's safe behavior.

Related work presents a similar concept for creating worst-case situations when testing the safe behavior of ADS [48, 137]. However, we cannot directly apply their methodology for testing the safe behavior of UAVs since different situations are challenging for UAVs and ADS. Further, due to the existence of fine-grained traffic rules and rigid road structures, the definition of the safe behavior of ADS is less challenging than for UAVs. The authors of

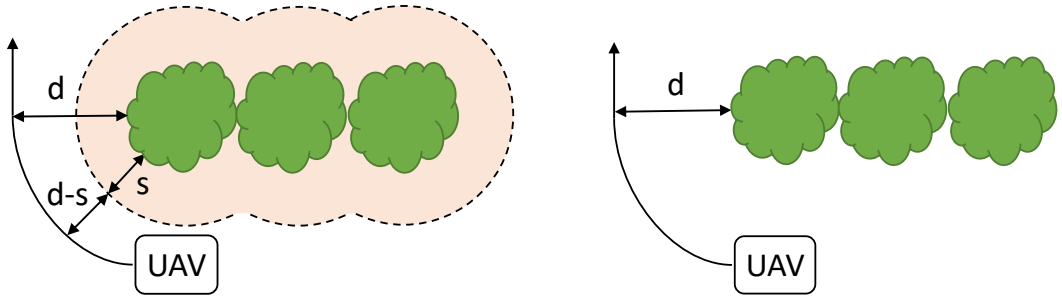
[146, 147] concentrate on testing the safe behavior of UAVs in an environment with other UAVs but no further obstacles or environmental effects. Since we believe that the operation of UAVs in urban environments will be crucial for their deployment in the future, we need to take the environment into account when testing the safe behavior of UAVs. Finally, all presented papers test against a specified safety distance and neglect the case where we might not be able to define the safe behavior of UAVs with such a safety distance.

The **contribution of this chapter** is a methodology for generating "good" test cases for testing the safe behavior of UAVs in various logical scenarios that take the UAV's environment into account. The proposed approach defines objectives to discover worst-case situations for two cases: (1) we can specify a safety distance and use it when generating test cases, and (2) we lack such a definition and still aim to find challenging situations. Experiments present the effectiveness of our proposed methodology for finding "good" test cases for four logical scenarios and both use cases.

5.2. Challenges of Defining the Safe Behavior

When assessing the safe behavior of UAVs, we first need to explicitly define such a safe behavior. However, as we do not have fine-grained traffic rules for UAVs and UAVs operate in an open field, specifying such a safe behavior is more challenging than for ADS. Current regulations forbid the operation of UAVs near people or structures. Depending on the country of operation, we need to keep a distance of 30 to 50 meters to these entities [8, 21, 35]. As UAVs should, e.g., deliver packages to our doorsteps in the near future, these regulations will change and be less restrictive. As we are unaware of how these regulations will be formulated and how fine-granular they will be, we need to consider the possibility that they might not specify a safety distance for all possible situations that the UAV might encounter in urban areas. Thus, we look at two cases throughout this chapter: (1) we have a specified safety distance, and (2) we lack such a safety distance.

Fig. 5.1 presents a first overview of how we can find worst-case situations for both of these cases. We will explain these approaches in more detail in the following subsection. We call the case with a specified safety distance Safety Distance Testing (SDT) and the one without one Boundary Analysis Testing (BAT). In the case of SDT, we search for challenging situations in which the SUT approaches the specified safety distance s. Thus, in our search for worst-case situations, we compare the UAV's distance d to any obstacle with the defined safety distance s to assess the UAV's behavior. Further, this approach presents an automatic oracle of the UAV's safe behavior. If $d < s$, the UAV behaves unsafely; if $d \geq s$, it shows a safe behavior. For BAT, we lack a safety distance due to missing requirements, specifications, or regulations. In this case, similar to SDT, we can generate worst-case situations by, e.g., minimizing the distance that the UAV keeps from any obstacle. However, there exist also other options for how we can find worst-case situations for BAT. If the UAV needs to fly through a gap between two obstacles, we can also generate challenging situations by finding the minimal gap through which the UAV still flies. The most distinct difference



(1) SDT (2) BAT

Figure 5.1.: For Safety Distance Testing (SDT), we can use a specified safety distance s to assess the UAV's safe behavior. It behaves safely if it keeps a distance of $d > s$ to any obstacle while operating. For Boundary Analysis Testing (BAT), we lack such a safety distance and instead create worst-case situations by, e.g., minimizing the UAV's distance d to all obstacles. A previous version appeared in [114].

between SDT and BAT is that we lack an automatic oracle for BAT as we have no safety distance given. Thus, an expert needs to inspect the discovered worst-case situations to evaluate the UAV's safe behavior.

5.3. Generation of "Good" Test Cases

In this section, we present our methodology for generating "good" test cases for testing the safe behavior of UAVs. The inputs for this approach are the logical scenarios in which we aim to test the SUT. While generating test cases, we do not randomly pick them from the search space but instead search for those test cases that can reveal potential faults in the SUT. To find these so-called "good" test cases, we apply search-based techniques that achieve valuable results when creating "good" test cases for ADS [48, 137]. Thus, as an output, we gain test cases for each logical scenario and worst-case situations that present the most challenging circumstances for the UAV in these logical scenarios.

5.3.1. Methodology

We present an overview of our methodology for generating "good" test cases for testing the safe behavior of UAVs with search-based techniques in Fig. 5.2. When applying these techniques, we first need to define the search space and the fitness function ①. The logical scenarios ② in which we aim to test the UAV's behavior serve as input to this step. We can derive these logical scenarios automatically or manually, as presented in Chapter 3. Further, when manually gathering logical scenarios in an ontology, we can use the methodology presented in Chapter 4 to find lower and upper bounds for the dimensions of this ontology.

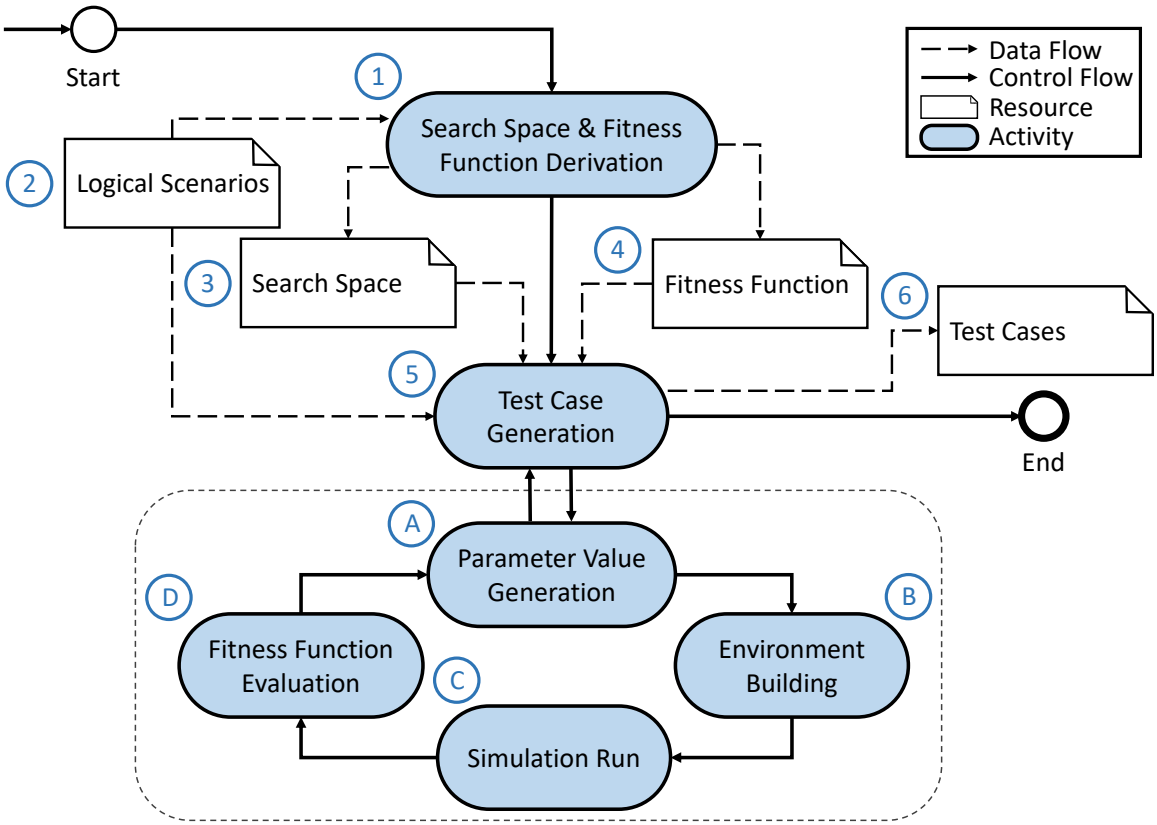


Figure 5.2.: Overview of our proposed methodology for generating "good" test cases with search-based techniques. A previous version appeared in [114].

In the next step, we can use the derived logical scenarios to define the search space for the test case generation step. This search space ③ represents all possible test cases for a given logical scenario. Thus, the search space is an n-dimensional space denoting the parameter ranges for the n parameters P of the evaluated logical scenario. We discuss exemplary search spaces in Section 5.3.2. The fitness function ④ describes the goal of the search, which is the detection of worst-case situations in our case. We present various suitable fitness functions for our search problem in Section 5.3.3. After specifying the search space and the fitness function, we generate test cases ⑤ for the given logical scenarios ②.

While creating test cases, an optimization algorithm picks concrete parameter values Ⓐ from the search space of the evaluated logical scenario that represents a candidate. Then, we build the simulation environment of the UAV Ⓑ based on the parameter values of this candidate and place the UAV at its starting position. Next, we present the UAV and dynamic obstacles with their missions and collect data about the UAV while the simulation is running Ⓒ. Finally, we compute the fitness of the currently assessed candidate by

Table 5.1.: Three exemplary search spaces describing simplified logical scenarios with one static spherical obstacle on the ground level and no environmental effects. The search spaces include parameter value ranges for the starting and landing position of the UAV and the position and the radius of the obstacle.

#	start_x	start_y	land_x	land_y	obst_x	obst_y	obst_r
1	[0.0, 2.0]	[1.0, 9.0]	[8.0, 10.0]	[1.0, 9.0]	[3.5, 6.5]	[1.5, 5.5]	[0.5, 1.0]
2	[-2.0, 0.0]	[5.0, 10.0]	[6.0, 8.0]	[0.0, 5.0]	[1.5, 4.5]	[3.0, 7.0]	[0.1, 3.5]
3	[-4.0, 4.0]	[-15.0, -5.0]	[16.0, 24.0]	[5.0, 15.0]	[8.0, 12.0]	[-3.0, 3.0]	[3.0, 8.0]

applying the specified fitness function (D). The optimization algorithm repeats these steps (A) to (D) for different candidates until it meets a termination criterion, e.g., it has evaluated a pre-defined number of candidates. When generating new candidates, the optimization algorithm tries to create candidates with better fitness values according to the specified fitness function. The output of this test case generation step (5) are the created test cases (6) that include the worst-case situations with the best fitness values. By inspecting the behavior of the UAV in these worst-case situations, we gain an understanding of its safe behavior.

5.3.2. Search Space

As mentioned before, the search space represents the set of all possible concrete scenarios for a given logical scenario. As we describe logical scenarios for UAVs with many parameters P, the search space for generating "good" test cases is high-dimensional. Since we outlined the derivation of logical scenarios for testing the safe behavior of UAVs in Chapter 3, we focus on creating "good" test cases for the given logical scenarios in this chapter.

Table 5.1 presents three simplified search spaces for a logical scenario with one static spherical obstacle on the ground level and no environmental effects. These exemplary search spaces include parameters for the starting position of the UAV on the x- and y-axis, the landing position of the UAV on the x- and y-axis, the position of the obstacle on the x- and y-axis, and the radius of the obstacle. Note the need to specify a range of parameter values for each parameter to define a specific search space. With the presented search spaces, we enable the optimization algorithm to change the starting and landing position of the UAV in a given range as well as the position and size of the obstacle when searching for "good" test cases in the provided logical scenario. As the search space for a logical scenario only depends on its parameters and is independent of whether we can define a safety distance or not, the search space is the same for SDT and BAT. However, within the same search space, different concrete scenarios might present worst-case situations for the SUT for these two cases.

5.3.3. Fitness Function

The fitness function guides the search for "good" test cases in the specified search space. As introduced in [48], the fitness function first needs to ensure that the logical scenario is presented for which we search for "good" test cases. If this is the case, the fitness function evaluates the objective o that leads to "good" test cases. Otherwise, the fitness function assigns a poor fitness value to the evaluated concrete scenario cs as it does not represent the given logical scenario. This poor fitness value is set to infinity in our case as we aim to minimize the objective in our fitness function. We present this general fitness function f in Eq. (5.1):

$$f(cs) = \begin{cases} o, & \text{if given logical scenario is represented} \\ \infty, & \text{otherwise} \end{cases} \tag{5.1}$$

As the objective o differs for SDT and BAT, we will look at specific fitness functions for these two cases in the subsequent subsections. Note that we can place a geo-fence in our simulation to restrict the flying area of the UAV if desired and, thus, do not need to check such restrictions in the fitness functions.

Fitness Function for Safety Distance Testing

In SDT, we have a specified safety distance that we can use in a fitness function to search for challenging situations for the UAV. With this safety distance, we can define a safety area around all obstacles the UAV should not enter. In a challenging situation, the UAV approaches the borders of these safety areas around the obstacles. Thus, in a fitness function for SDT, we compare the distance $d(cs, t)$ that the UAV keeps in a specific concrete scenario cs to any obstacle at time $t \in T$ with the specified safety distance for this concrete scenario $s(cs, t)$. We denote the corresponding fitness function f_{sdt} in Eq. (5.2):

$$f_{sdt}(cs) = \begin{cases} \min(\{t \in T : d(cs,t) - s(cs,t)\}), & \text{if given logical scenario is represented} \\ \infty, & \text{otherwise} \end{cases} \tag{5.2}$$

Note that we can apply the presented fitness function for various logical scenarios when we have a specified safety distance. The concrete safety distance might change over time and might be dependent on various factors such as the UAV's velocity or size or the wind speed in the currently evaluated concrete scenario. Finally, this fitness function has the advantage of providing an automatic oracle that solves the oracle problem presented in [12]. If the computed fitness value of a concrete scenario is positive, the UAV behaved safely in this situation as it kept a distance $d \geq s$ at any time during operation. However, if the calculated fitness value is negative, we know that the UAV behaved unsafely as it violated the specified safety distance since $d < s$.

Fitness Function for Boundary Analysis Testing

In BAT, we lack a given safety distance and, thus, need to adapt the fitness function compared to SDT for finding challenging situations for the SUT. As the name suggests, we search for the boundary between the safe and unsafe behavior of the UAV in the given logical scenarios. To avoid an obstacle, the UAV has four alternatives: (1) it can fly around it on the left- or right-hand side, (2) it can fly above it, (3) it can fly through a gap between two obstacles, or (4) it can fly below it, thus, through a gap between the ground and the obstacle. We can specify different objectives to guide the search for worst-case situations for these various alternatives. For (1) and (2), we search for the minimal distance $d(cs, t)$ that the UAV keeps to any obstacle at time $t \in T$, as denoted in Eq. (5.3):

$$f_{bat,1}(cs) = \begin{cases} \min(\{t \in T : d(cs,t)\}), & \text{if given logical scenario is represented} \\ \infty, & \text{otherwise} \end{cases} \tag{5.3}$$

For (3) and (4), we can find challenging situations by minimizing the width of the gap $w(cs, t)$ between the obstacles or the obstacle and the ground at time $t \in T$. In these cases, it is essential that we ensure that the given logical scenario is presented and the UAV flies through the gap. Thus, a concrete scenario with no gap will receive a poor fitness value of infinity. Equation (5.4) presents the corresponding fitness function $f_{bat,2}$:

$$f_{bat,2}(cs) = \begin{cases} \min(\{t \in T : w(cs,t)\}), & \text{if given logical scenario is represented} \\ \infty, & \text{otherwise} \end{cases} \tag{5.4}$$

Even though the fitness function presented in Eq. (5.3) is similar to Eq. (5.2) for SDT, it does not provide an automatic oracle. Instead, an expert needs to investigate the discovered worst-case situations to evaluate the safe or unsafe behavior of the UAV. However, the expert only needs to inspect the worst-case situations and does not need to look at all generated concrete scenarios. The same applies when using the fitness function presented in Eq. (5.4).

5.4. Experiments

In our experiments, we investigate the effectiveness and applicability of our proposed methodology for generating "good" test cases for testing the safe behavior of UAVs. We perform these experiments in four logical scenarios, which present all alternatives to avoid an obstacle, and for the two cases of SDT and BAT.

5.4.1. Setup and Implementation

In these experiments, we generate test cases for the obstacle detection and avoidance extension of the open-source PX4 autopilot [81], which we simulate in the Gazebo simulator [61].

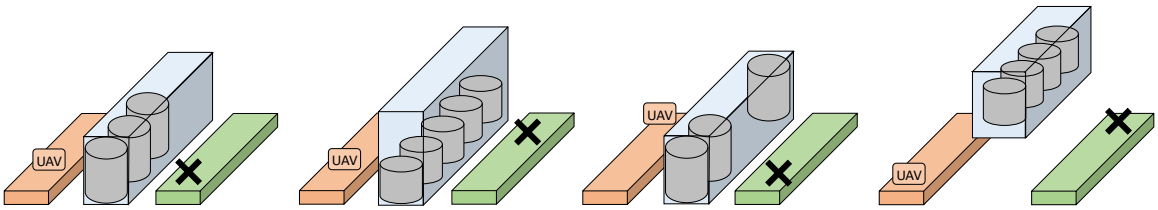


Figure 5.3.: In our experiments, we evaluate four logical scenarios in which the UAV flies (1) around obstacles, (2) above obstacles, (3) through a gap between two obstacles, and (4) below an obstacle to reach its destination point. The UAV starts in the area on the left and lands in the area on the right in each of these logical scenarios. A previous version appeared in [114].

We forward a mission to fly to a specified target point to the UAV via the MAVSDK-Python library [92]. During the simulation, we collect data about the UAV via ROS [100] to calculate a fitness value for each evaluated candidate. In the experiments for this chapter, we use the optimization algorithm NSGAII [32], following the guidance of [1, 6]. We present a technical comparison of various optimization algorithms and their performance for our search for worst-case situations in Chapter 6. We utilize the algorithm's implementation from the jMetalPy framework [15] with its default parameter settings. Since we often face multi-objective fitness functions when testing autonomous systems and aim to enable their use in future work, we apply this multi-objective algorithm to our single-objective fitness functions. In our experiments, this algorithm searches for "good" test cases following the methodology presented in Fig. 5.2 for 200 evaluations for each logical scenario. Even though this number is relatively low, we can already discover insights into the safe behavior of the tested UAV. To enable the reproducibility of these experiments, we provide additional details on the experiment settings as, e.g., the version number of the used libraries, in Appendix B. Further, we present the parameter values of the concrete scenarios generated in these experiments and their fitness values in [112].

5.4.2. Logical Scenarios and Search Spaces

We present the four logical scenarios for the experiments in Fig. 5.3, in which the UAV flies (1) around a tree row, (2) above a grocery store, (3) through a gap between two tree rows, and (4) below a bridge to reach its destination point. In our experiments, we use the search spaces shown in Table 5.2 for these logical scenarios. Note that these are exemplary search spaces for the presented logical scenarios and that experts need to choose them system-specifically. In these search spaces, the optimization algorithm can adapt the position of the obstacles on the x- and y-axis with the parameters $obst\_1\_x$, $obst\_1\_y$, $obst\_2\_x$, and $obst\_2\_y$, which also changes the width of the gap between the tree rows in scenario (3). Further, the algorithm can modify the starting and landing position of the UAV in a specified area with

Table 5.2.: The search spaces for the four logical scenarios in our experiments.

Scenario	**start_x**	**start_y**	**land_x**	**land_y**		
All	[-1.0, 1.0]	[-10.0, 10.0]	[12.0, 14.0]	[-10.0, 10.0]		
Scenario	**obst_1_x**	**obst_1_y**	**obst_1_z**	**obst_1_h**	**obst_2_x**	**obst_2_y**
(1)	[4.0, 5.0]	[-10.0, 15.0]	—	—	—	—
(2)	—	—	—	[1.0, 10.0]	—	—
(3)	[4.0, 5.0]	[-10.0, 15.0]	—	—	[4.0, 5.0]	[-10.0, 15.0]
(4)	—	—	[0.0, 6.0]	—	—	—

the parameters $start\_x$, $start\_y$, $land\_x$, and $land\_y$. For scenarios (2) and (4), we fix the position of the obstacles on the x- and y-axis. Instead, we alter as additional parameters in the search space the obstacle's height $obst\_1\_h$ in (2) and the position on the z-axis $obst\_1\_z$ in (4) to modify the width of the gap between the bridge and the ground. Note that we use only static obstacles in these experiments for simplicity of presentation. However, we can easily extend the search space to include dynamic obstacles by adding their trajectories and velocities.

5.4.3. Experimental Results for Safety Distance Testing

For simplicity of presentation, we use a fixed safety distance $s(cs, t) = 1.0$ meters for our experiments for SDT. Note that we usually use a non-static safety distance that might change over time and depend on various characteristics of the UAV and its environment. In our experiments for SDT, we aim to discover concrete scenarios from the search space in which the SUT approaches the specified safety area around the obstacles and potentially violates it. Note that the presented fitness function in Eq. (5.2) yields a negative fitness value when the UAV violates the specified safety distance and, thus, represents an automatic oracle about the UAV's safe behavior. We present the results of our experiments for SDT in Table 5.3. This table shows for each investigated logical scenario the number of concrete scenarios in which the SUT shows an unsafe behavior by violating the defined safety distance. In addition, we depict the highest of these violations for each logical scenario. In our experiments, we find 4 to 62 violations of the defined safety distance in each logical scenario. Fig. 5.4 presents examples of these detected violations by depicting the UAV and the safety areas around the obstacles in red.

5.4.4. Experimental Results for Boundary Analysis Testing

In the experiments for BAT, we lack a defined safety distance due to missing regulations or specifications. In these experiments, we aim to discover concrete scenarios in the search space that present challenging situations for the SUT. In such worst-case situations, the UAV

Table 5.3.: For the logical scenarios (1) - (4), we denote the number of concrete scenarios in which the SUT shows an unsafe behavior by violating the defined safety distance and the largest of these violations. A previous version appeared in [114].

	Scenario (1)	**Scenario (2)**	**Scenario (3)**	**Scenario (4)**
# Discovered Violations	38	4	4	62
Highest Violation	-0.95	-0.18	-0.82	-0.84

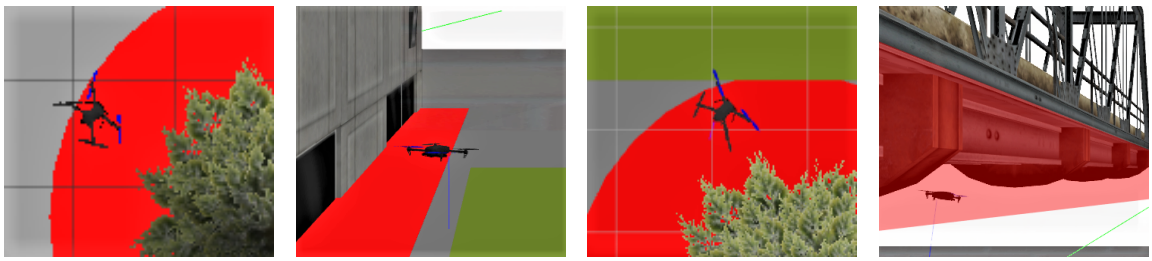
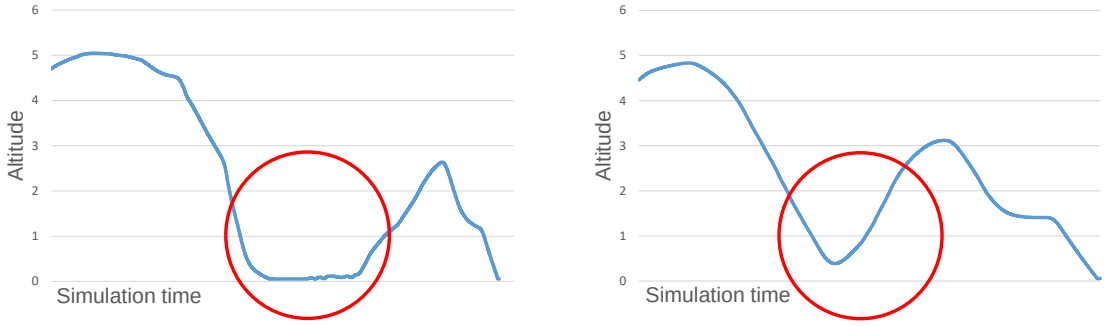


Figure 5.4.: Examples of the detected safety distance violations in our experiments for SDT. We mark the safety areas around the obstacles that the UAV should not enter in red. A previous version appeared in [114].

might select an unsafe trajectory due to the challenging circumstances. In our experiments, we search for these worst-case situations with the fitness functions presented in Eq. (5.3) for the logical scenarios (1) and (2) that minimizes the UAV's distance to the obstacles and Eq. (5.4) for the logical scenarios (3) and (4) that minimizes the width of the gap. In Table 5.4, we show the results of these experiments. For the logical scenarios (1) and (2), we present the best fitness value, which represents the minimal distance in meters the UAV keeps to any obstacle in all evaluated concrete scenarios and the number of concrete scenarios in which this minimal distance is smaller than 1.0 meters. For the logical scenarios (3) and (4), we present the minimal width of the gap through which the UAV flies in meters and the width of the gap for which the UAV shows a presumably safe behavior. Note that to assess the safe behavior of the UAV for BAT, experts need to inspect the discovered worst-case situations and judge the UAV's behavior in them. However, we reduce this manual effort as the experts only need to investigate the worst-case situations instead of all generated test cases for each logical scenario. While inspecting the worst-case situations for the logical scenarios (3) and (4), we discovered a questionable behavior of the SUT. In these situations, the UAV seems to recognize that the gap at the level of the tree trunks is broader than the one at the level of the tree tops and decreases its altitude accordingly. The UAV behaves in a similar way when encountering the bridge in the logical scenario (4). While lowering its altitude in both worst-case situations, the UAV unintentionally lands on the ground and rotates there before taking off again. This action represents a questionable behavior that we do not expect from a safely behaving UAV. By investigating additional concrete scenarios

Table 5.4.: Characteristics of the worst-case situations that we discovered in our experiments for BAT. Presented is the minimal distance the UAV keeps to all obstacles, the number of concrete scenarios in which this distance is below 1.0 meters, the minimal width of the gap through which the UAV flies, and the width of the gap in which the UAV shows a presumably safe behavior. A previous version appeared in [114].

	Scenario (1)	**Scenario (2)**	**Scenario (3)**	**Scenario (4)**
Minimal Distance	0.53	0.35	—	—
# Below 1.0	22	7	—	—
Minimal Gap Width	—	—	3.76	3.02
Gap Width Safe Behavior	—	—	4.77	4.53



(a) $w = 3.76$ meters (b) $w = 4.77$ meters

Figure 5.5.: The change in the UAV's altitude in (a) the worst-case situation for the logical scenario (3), in which the SUT unintentionally lands on the ground, and (b) for a broader gap w, for which the UAV shows a presumably safe behavior. We highlight the differences with the red circles. A previous version appeared in [114].

that have a slightly worse fitness value, we discovered a presumably safe behavior of the UAV for gaps of 4.77 and 4.53 meters and broader, in which the UAV stays in the air while decreasing its altitude and keeps decent distances to the obstacles. In Fig. 5.5, we visualize the behavior of the UAV in the worst-case situation for the logical scenario (3) and a broader gap by depicting the change of the UAV's altitude in the evaluated concrete scenarios.

5.4.5. Discussion

In our experiments, we aim to investigate the applicability and effectiveness of our proposed approach for generating "good" test cases for testing the safe behavior of UAVs. Therefore, we perform experiments in four logical scenarios representing all alternatives to avoid

obstacles for SDT and BAT. The results for SDT show the effectiveness of the proposed approach as we find safety distance violations in all four logical scenarios. These safety distance violations present unsafe behaviors of the SUT, which experts can inspect in the next step to discover their underlying fault. Further, the results demonstrate the advantage of having an automatic oracle for SDT since we can directly derive the safe or unsafe nature of the UAV's behavior from the computed fitness values. The results for BAT show how we can generate worst-case situations for the SUT that can reveal potential unsafe behaviors of the SUT, even without a defined safety distance. For BAT, an expert needs to inspect the discovered worst-case situations and manually analyze the UAV's behavior in these situations. However, we minimize the experts' effort as they only need to investigate the worst-case situations and not all generated test cases. The experimental results not only reveal questionable behaviors of the UAV in the worst-case situations but also indicate that the gap between two tree rows needs to be 4.77 meters or broader for the SUT to safely fly through it. In addition, a bridge needs to be located at least 4.53 meters above the ground for the SUT to safely fly below it. Overall, the experimental results show the applicability and effectiveness of the proposed approach for discovering potential unsafe behaviors of the SUT by generating worst-case situations for the SUT. Further, the experiments demonstrate how we can apply this methodology with and without a given safety distance and create "good" test cases for both of these cases.

Threats to Validity

As we aim to gain an understanding of how we can test the safe behavior of UAVs with search-based techniques, our experimental results in this chapter focus on showing the applicability of our proposed approach and do not present results for various SUTs or optimization algorithms. However, we present experimental results for different optimization algorithms in the subsequent chapter. In our experiments, we execute only a limited number of 200 evaluations for each logical scenario. However, as we discover unsafe behaviors of the SUT with this limited number of test cases, it shows the effectiveness of the proposed approach all the more. As we concentrate on understanding how we can generally test the safe behavior of UAVs, we do not execute 30 runs for our experiments in this chapter, as proposed by [5]. However, we present further experiments with more runs in the following chapters. To reduce the threats to internal validity in our experiments, we use the implementation of NSGAII from the open-source library jmetalPy and evaluate the open-source PX4 autopilot for UAVs.

5.5. Conclusion

When testing the safe behavior of UAVs in urban areas, we face the challenge of ensuring their safe behavior in all situations for a logical scenario, even in the worst-case situation, and the challenge of explicitly defining this safe behavior. To tackle these challenges, we

present a methodology for generating so-called "good" test cases that can reveal potential unsafe behaviors of the SUT. In addition, we demonstrate how we can apply this methodology when we (1) have a specified safety distance and (2) lack such a safety distance due to missing regulations or specifications. In our experiments, we show the applicability and effectiveness of our proposed approach in four logical scenarios that present all alternatives to avoid obstacles. When we have a given safety distance, we detect several violations of this safety distance in all evaluated logical scenarios. In addition, we discovered questionable behaviors of the SUT when testing our approach without a specified safety distance. Finally, we minimize the experts' manual effort needed when we cannot define a safety distance by presenting worst-case situations that the experts solely need to inspect.

In future work, we aim to expand our experiments to assess the UAV's behavior in logical scenarios that present real-world situations in more detail. In addition, we would like to explore additional fitness functions that might be applicable when we have no specified safety distance. Finally, we aim to investigate search spaces and fitness functions suitable for our proposed approach when testing the safe behavior of cooperative UAVs.

6. Evaluation of Optimization Algorithms for Testing the Safe Behavior of UAVs

This chapter introduces the problem of a missing guarantee for finding worst-case situations with heuristic optimization algorithms. Further, it presents a case study to explore the quality of generated test cases for three optimization algorithms and their sequential combinations when testing the safe behavior of an open-source UAV. Parts of this chapter previously appeared in a peer-reviewed publication [116] co-authored by the author of this thesis.

6.1. Introduction

As mentioned in the previous section, instead of randomly picking test cases for each logical scenario, we aim to find the most challenging concrete scenarios for the UAV for each logical scenario. Various papers [48, 114, 147] propose using heuristic search-based techniques to find these challenging situations that we also call worst-case situations. Even though the presented papers show that these techniques can effectively detect potential faults in the SUTs, they lack the *guarantee* of discovering the most challenging worst-case situation for a logical scenario due to their heuristic nature. However, when building a thorough safety argumentation for testing the safe behavior of UAVs, we need to reliably find worst-case situations to assess the UAV's safe behavior. Thus, as a first step toward ensuring that we tested each logical scenario sufficiently, we need to evaluate the quality of the worst-case situations that different optimization algorithms generate. In such a case study, we aim to explore which optimization algorithm reliably produces more challenging worst-case situations for a given SUT. In this chapter, we present such a case study and focus on the following three aspects: (1) we evaluate three given optimization algorithms, (2) we explore whether sequential combinations of optimization algorithms perform better than their base algorithms, and (3) we inspect which of the evaluated algorithms performs best for our use case.

In related work, there exist various case studies about optimization algorithms for problems in different domains such as hydro-powered plant management, wireless sensor networks clustering, or multi-objective land allocation [29, 64, 125]. The authors of [60] present an evaluation of different optimization algorithms for generating test cases for ADS that focuses on the convergence rate of the algorithms. In this chapter, we aim to

instead concentrate on the quality of the created worst-case situations. To the best of our knowledge, no work that evaluates the performance of various optimization algorithms for testing the safe behavior of UAVs with scenario-based testing exists. Since the performance of optimization algorithms is highly context-specific and cannot be generalized over several domains, there is a need to conduct a case study about the quality of worst-case situations generated by various optimization algorithms for UAVs. When combining optimization algorithms sequentially, related work presents diverse use cases. The authors of [85, 102] use different optimization algorithms to solve distinct and separate sub-problems of their search problem. [130] evaluates the performance of two optimization algorithms for their search problem before choosing one for the overall evaluation. Finally, several works exist [24, 33, 77] that combine optimization algorithms to exploit their advantages, e.g., their good exploration or exploitation characteristics. In this chapter, we concentrate on this last use case and explore the effectiveness of such combinations for generating worst-case situations for testing the safe behavior of UAVs.

The **contribution of this chapter** is a case study that demonstrates one of the crucial problems of testing the safe behavior of UAVs with scenario-based testing: various optimization algorithms generate different worst-case situations without a guarantee of finding the worst one. While it is well-known that heuristic search does not provide optimality guarantees, our results show that non-optimality does not seem to be an exceptional case. In addition, the results demonstrate that safety violations are instead likely to be missed by some algorithms and that the outcome of different algorithms differs substantially, by up to 20%. For these reasons, scenario-based testing comes at the extra cost of having to run multiple optimizers to find worst-case situations, which challenges the current widespread application of scenario-based testing for ensuring the safe behavior of autonomous systems such as ADS or UAVs.

6.2. Optimization Algorithms

When searching for "good" test cases, the search space presented by the parameter ranges of a logical scenario denotes a high dimensional space. Due to this high dimensionality, we cannot use exact algorithms such as integer linear programming, dynamic programming, or branch-and-bound [98] to compute optimal solutions for our problem. Instead, we can use heuristic methods that produce good solutions in a reasonable amount of time but cannot guarantee finding the optimal solution for a given problem. There exists a large variety of heuristic optimization algorithms that all have advantages and disadvantages, i.a., genetic algorithms, particle swarm optimization, and ant colony optimization [17]. As an alternative, we can use surrogate optimization algorithms such as Bayesian optimization [42], which additionally provide the option to compute the credibility of the results.

To gain an understanding of the quality of the produced "good" test cases, we aim to investigate whether some optimization algorithms achieve better results than others in the search for worst-case situations. In this work, we explore the performance of three

well-known and diverse optimization algorithms: NSGAII [32], PSO [56], and BO [96]. All three algorithms use different approaches for finding solutions for the given problem and, thus, represent a good mix of algorithms, in our opinion. Further, these three algorithms showed a good performance in several small pre-experiments that we executed before performing the case study presented in this chapter. One can find an overview of the process of each of these algorithms in Section 2.4.

In this work, when combining optimization algorithms, we consider collaborative combinations of these algorithms, as explained in [98]. In collaborative combinations, algorithms supplement each other by exchanging information but are not integrated into each other. In our case study, we execute the optimization algorithms sequentially and pass on information about their best solutions. Several papers on seeding strategies [4, 26, 41] emphasize the beneficial effect of using already good performing candidates compared to new randomly generated ones for additional populations or optimization algorithms. According to [4], the amount of seeded candidates seems to be less important than the existence of them in the initial data for the next algorithm. Based on these papers, we decided to create the initial candidates for the subsequent algorithm in a two-fold way. One half consists of the best candidates from the previous algorithms, following the guidance of [26]. The other half consists of randomly generated candidates from areas not yet heavily covered by the previous algorithms. To determine the candidates for this second half, we investigate the coverage of the previously evaluated candidates in the search space. Since the search space for our problem is too high dimensional to reasonably investigate the coverage of all possible sub-parts of the search space, we decided to inspect the coverage of each dimension separately. Therefore, we divide each dimension of the search space into 10 sub-parts and investigate the number of evaluated candidates with parameter values in each of these parts. The smaller the number of candidates with values in a sub-part, the less the optimization algorithm covered this sub-part. Thus, to not neglect these areas, we generate candidates for the second half of the initial set of candidates for the subsequent algorithm in the three sub-parts with the lowest number of evaluated candidates. We consider all possible combinations of the three optimization algorithms NSGAII, PSO, and BO for three executions with the exception of the direct repetition of the same algorithm. Table 6.1 presents an overview of the sequential execution of the resulting 12 combinations of these algorithms. We give every combination a short name for further reference that includes the first letter of the executed algorithms in their order. The resulting best fitness value of a combined algorithm is the minimal fitness value that one of the algorithms in the combination achieves.

6.3. Case Study

When searching for "good" test cases for testing the safe behavior of UAVs with heuristic methods, we lack a guarantee of finding the worst-case situations. Since the fitness function for this search is highly complex, we cannot assume a simple distribution that would be

Table 6.1.: The combinations of the optimization algorithms NSGAII, PSO, and BO that we investigate in this work.

Short Name	Algorithm 1	Algorithm 2	Algorithm 3
NPN	NSGAII	PSO	NSGAII
NPB	NSGAII	PSO	BO
NBN	NSGAII	BO	NSGAII
NBP	NSGAII	BO	PSO
PNP	PSO	NSGAII	PSO
PNB	PSO	NSGAII	BO
PBN	PSO	BO	NSGAII
PBP	PSO	BO	PSO
BNP	BO	NSGAII	PSO
BNB	BO	NSGAII	BO
BPN	BO	PSO	NSGAII
BPB	BO	PSO	BO

necessary for computing guarantees with stochastic analysis. Instead, to explore the performance of several optimization algorithms and their combinations for this search problem, we perform an empirical case study following the guidance of [5, 101] that investigates the safe behavior of the open-source PX4 autopilot [81] in a simulation environment.

6.3.1. Setup and Implementation

During this case study, we generate test cases for the obstacle detection and avoidance extension of the open-source PX4 autopilot [81]. To investigate the behavior of this UAV, we simulate the system with Gazebo [61] and assess the recorded simulation data. We use this recorded data and the fitness function described in Eq. (5.2) to assess the UAV's behavior:

$$f_{sdt}(cs) = \begin{cases} \min(\{t \in T : d(cs,t) - s(cs,t)\}), & \text{if given logical scenario is represented} \\ \infty, & \text{otherwise} \end{cases}$$

For simplicity of presentation, we set the safety distance $s(cs,t)$ to a fixed value of 1.0 meters. In previous experiments, we could create various worst-case situations for the PX4 autopilot in which it violates specified safety distances when including dynamic obstacles in its environment. On the other hand, for several logical scenarios with only static obstacles, the UAV shows a *presumably* safe behavior by not violating any safety distances. There exist two possible causes for this safe behavior: (1) the UAV does behave safely in all alternatives of the given logical scenario, even in the worst-case situation, or (2) the optimization algorithm did not find the worst-case situation in which the UAV might still behave unsafely. Due to these two possible reasons, we aim to investigate the

performance of different optimization algorithms, especially in logical scenarios in which the UAV *presumably* behaves safely in all generated test cases. Since the PX4 autopilot shows such a behavior for logical scenarios with only static obstacles, we focus on these in this case study. Table 6.2 presents the evaluated logical scenarios that include different landforms, surface natures, obstacles, wind forces, and fog thicknesses. Note that our simulation setup limits our choices for denoting characteristics of logical scenarios so that the presented logical scenarios do not provide a complete description of real-world situations. However, as we discuss a general problem of applying optimization algorithms for finding worst-case situations for autonomous systems independently of the evaluated concrete logical scenarios, such a comprehensive description is not needed in this chapter.

In our case study, we investigate the performance of the optimization algorithms NSGAII, PSO, BO, and their sequential combinations, as presented in Table 6.1. To implement these algorithms, we utilize the jMetalPy framework [15] for the optimization algorithms NSGAII and an improved version of PSO, called Speed-constraint Multi-objective PSO, and the BayesianOptimization framework [91] for implementing BO. As proposed in [60], we run 500 evaluations for assessing a given optimization algorithm. When combining algorithms, we perform 300 evaluations for the first algorithm to enable it to find attractive areas in the search space, followed by 100 evaluations for the second and third algorithms to refine the search in these areas. These numbers add up to 500 evaluations to enable comparability with the given optimization algorithms. For BO, we use the Expected Improvement acquisition function and a balanced value for xi with 0.05 to allow an equal exploration and exploitation. To enable a fair comparison, we select the same value for the population size and offspring population size for NSGAII, the swarm size for PSO, and the number of initial evaluations for BO. We set this value to 50 for the given algorithms and first iterations of the combinations and reduce it to 20 for the second and third iterations of the combinations as these iterations encompass fewer candidates. To explore the robustness of the optimization algorithms, we follow the guidance of [5, 13, 101] and perform several runs of generating "good" test cases with a given algorithm for each logical scenario, namely fifteen. Note that we limit ourselves to evaluating each logical scenario fifteen times in this case study instead of thirty times as proposed by [5] due to limited computation power and time. Since we need to assess the UAV's behavior in each generated test case, we need to perform a simulation of about two minutes for each test case. Since we evaluate 15 algorithm combinations in 5 logical scenarios for 500 evaluations each, we need $15 * 5 * 500 * 2$ minutes $= 52$ days per run when executing the algorithms sequentially. Nonetheless, we achieve decent median absolute deviation values with these fifteen runs, as presented in Section 6.3.3. To compare the optimization algorithms fairly, we use the same initial populations for all algorithms for each run of a logical scenario, as proposed in [13, 77, 101]. However, note that we use a different initial population for each of the 15 runs. When combining algorithms, we use the process described in Section 6.2 to share information between them. To enable the reproducibility of these experiments, we provide additional details on the experiment settings as, e.g., the version number of the used libraries, in Appendix B. Further, we present all fitness values created during these

Table 6.2.: The five logical scenarios evaluated in the case study include various landforms, surface natures, wind forces, types of reduced visibility as well as different sizes (small S, medium M, and large L) and forms of obstacles (cuboid CU, sphere SP, and cylinder CY).

Scenario	1	2	3	4	5
Landform	depression	steep transition	flat	depression	elevation
Nature	water	water	land	land	mixture
Wind	moderate	light	strong	light	light
Red. Visibility	fog	thick fog	none	heavy fog	fog
# Obstacles	1	2	2	3	4
	S	M	S	L	S
Obstacle	—	M	L	M	M
Sizes	—	—	—	M	S
	—	—	—	—	M
	CU	CU	CY	SP	CY
Obstacle	—	SP	CU	CY	CY
Forms	—	—	—	CU	CU
	—	—	—	—	CY

experiments for the evaluated logical scenarios and runs and the convergence of the minimal fitness values in [112].

6.3.2. Evaluation Objectives

When evaluating optimization algorithms, common objectives found in the literature are the success rate [77, 102], the best and average fitness values [77, 85, 125], and the convergence speed of the fitness values [64, 88]. Since the success rate depends on knowing the optimal solution, we cannot use it in our case study for generating "good" test cases for UAVs as these "good" test cases are not known beforehand. In addition, as mentioned earlier, finding an optimal solution corresponding to a worst-case situation is more important in our use case than having a quick convergence rate of the algorithms. For safety argumentations, running the fastest algorithm is insufficient as long as it cannot find the worst-case situation in this shorter run-time. When generating "good" test cases for testing the UAV's behavior, we commonly only execute a small number of runs per logical scenario and system version due to time constraints. Thus, the high average performance of an optimization algorithm is more important for our use case than high peak performance [38]. Therefore, following the guidance of the presented literature, we explore (1) the overall best fitness value, which corresponds to the worst-case situation found in all runs for a logical scenario. In addition, we investigate (2) the median of the detected best fitness values found in all runs to explore

the average performance of an optimization algorithm. Finally, we inspect (3) the median absolute deviation from the median best fitness value detected in all runs to compute how heavily the best fitness values deviate from their median. We decided to use this metric instead of the standard deviation since it is more robust to outliers. In our case study, an optimization algorithm achieves high average performance with a low median value of the found minimal fitness values and a low median absolute deviation from this median value.

6.3.3. Evaluation Results

In Table 6.3, we present the results of our evaluation of three optimization algorithms and their combinations for testing the safe behavior of UAVs. For each algorithm and each logical scenario, we show the best fitness value achieved overall runs, the median of these minimal fitness values in all runs, and the median absolute deviation of these values. For each algorithm, we further present the median value of each objective considering all logical scenarios. Finally, we highlight the minimal value for each objective representing the minimal distance of the UAV to the obstacles or the minimal median deviation. PBP achieves the best minimal fitness value with -0.22, while PNP provides the best median of the minimal fitness values with 1.24 and the minimal median absolute deviation with 0.36. To better understand the relevance of the results, we present the performance of the algorithms compared to a base algorithm in Table 6.4. We use NSGAII, which is currently applied in literature [114, 147], as a base algorithm for sub-categories (1) and (3) of our problem, which we presented in the introduction. Further, we inspect the performance of the combinations compared to their base algorithm for sub-category (2). To investigate the significance of the results of our case study, we follow the suggestion of [5] and perform a Mann-Whitney U test that explores the difference between all minimal fitness values found in the fifteen runs by each of the algorithms in all logical scenarios. We present the resulting P-values of these tests in Table 6.4 next to the performance of the algorithms. Two algorithms perform significantly differently for a confidence interval of 95% if the P-value is below 0.05. For better visualization, we highlight these occurrences in bold in Table 6.4. In addition, we compute Vargha and Delaney's A_{12} measure [131], which calculates the probability of algorithm 1 yielding better results than algorithm 2. The advantage of this statistical measure is that it is independent of specific data distributions. The resulting effect size of this measure ranges between 0 and 1, with 0.5 indicating that the performance of both algorithms is stochastically equal. A result of $A_{12} > 0.5$ suggests that the first algorithm produces higher results than the second algorithm. However, as smaller fitness values indicate better performance in our use case, a result of $A_{12} > 0.5$ indeed indicates a better performance of the second algorithm in our use case. The authors of [131] further present guidelines for interpreting the effect size of this measure by categorizing it as negligible, small, medium, or large depending on the achieved A_{12} value. In Table 6.4, we also present Vargha and Delaney's A_{12} measure for the optimization algorithms' performances and its corresponding effect size category. In this measure, we compare the performance of the base algorithms as algorithm 1 with the performance of the other algorithms as algorithm 2.

Table 6.3.: The resulting minimal fitness values, the median minimal fitness values, and their median absolute deviation discovered in all runs for the evaluated five logical scenarios. We underline the best median values per objective.

Minimal Fitness Value						
Scenario	1	2	3	4	5	Median
NSGAII	1.06	1.47	0.30	0.38	-0.02	0.38
PSO	1.05	1.33	0.36	0.72	0.10	0.72
BO	1.24	1.34	-0.03	1.18	0.86	1.18
NPN	0.98	1.54	-0.15	0.65	-0.27	0.65
NPB	1.15	1.54	-0.03	0.70	0.01	0.70
NBN	0.19	1.38	-0.05	0.67	-0.14	0.19
NBP	0.14	1.53	-0.82	0.71	-0.65	0.14
PNP	0.89	1.31	0.21	-0.22	0.19	0.21
PNB	0.89	1.34	0.24	-0.22	0.19	0.24
PBN	0.65	1.33	-0.14	-0.22	-0.17	-0.14
PBP	0.66	1.46	-0.62	-0.22	-0.56	<u>-0.22</u>
BNP	-0.04	1.27	0.16	-0.73	-0.21	-0.04
BNB	0.25	1.32	0.16	0.66	-0.21	0.25
BPN	0.01	1.01	0.20	0.48	0.45	0.45
BPB	0.01	1.48	0.34	0.48	0.69	0.48
Median Minimal Fitness Value						
Scenario	1	2	3	4	5	Median
NSGAII	1.71	1.67	0.69	1.49	1.55	1.55
PSO	2.05	1.90	0.89	1.26	1.58	1.58
BO	1.76	2.17	1.44	2.36	1.88	1.88
NPN	1.76	1.68	0.37	1.39	0.77	1.39
NPB	2.02	1.77	0.71	1.74	1.13	1.74
NBN	1.95	1.68	0.55	1.36	1.20	1.36
NBP	1.90	1.77	0.75	1.47	1.06	1.47
PNP	1.58	1.52	0.69	1.24	1.18	<u>1.24</u>
PNB	1.81	1.53	0.69	1.48	1.24	1.48
PBN	1.65	1.63	0.73	1.30	1.19	1.30
PBP	1.74	1.66	0.77	1.41	1.25	1.41
BNP	1.38	1.52	0.83	1.22	1.61	1.38
BNB	1.54	1.63	0.81	1.92	1.73	1.63
BPN	1.39	1.51	0.92	1.89	1.42	1.42
BPB	1.46	1.79	1.03	2.16	1.72	1.72

continued on next page

Table 6.3 – continued from previous page

Median Absolute Deviation of Minimal Fitness Values						
Scenario	1	2	3	4	5	Median
NSGAII	0.41	0.07	0.40	0.77	0.66	0.41
PSO	0.49	0.50	0.48	0.47	0.26	0.48
BO	0.42	0.54	0.40	0.37	0.37	0.40
NPN	0.36	0.20	0.56	0.64	0.42	0.42
NPB	0.35	0.24	0.50	0.62	0.39	0.39
NBN	0.45	0.32	0.29	0.70	0.58	0.45
NBP	0.47	0.21	0.29	0.62	0.42	0.42
PNP	0.51	0.12	0.35	0.92	0.36	0.36
PNB	0.38	0.24	0.34	0.57	0.41	0.38
PBN	0.65	0.32	0.49	0.52	0.41	0.49
PBP	0.72	0.30	0.46	0.78	0.30	0.46
BNP	0.53	0.26	0.49	0.82	0.42	0.49
BNB	0.74	0.25	0.41	0.59	0.19	0.41
BPN	0.85	0.31	0.49	0.66	0.70	0.66
BPB	0.53	0.27	0.39	0.48	0.66	0.48

6.3.4. Discussion

As mentioned before, in our use case of generating "good" test cases for testing the safe behavior of UAVs, high average performance is more important than a high peak performance of an optimization algorithm. Further, finding a minimal fitness value and, thus, a worst-case situation is the most influential factor in our use case. Thus, an algorithm with the lowest median of minimal fitness values found in various runs and logical scenarios presents the most suitable optimization algorithm for our use case. Further, a low median absolute deviation shows that the evaluated algorithm can reliably produce the presented results. When investigating the performance of the evaluated optimization algorithms, we focus on three sub-categories of the problem, as mentioned earlier.

Category (1) - Comparison of NSGAII, PSO, and BO

First, we inspect the performance of the three given optimization algorithms NSGAII, PSO, and BO for our use case of generating "good" test cases for UAVs. Compared with PSO and BO, NSGAII creates test cases with the lowest median value of minimal fitness values overall logical scenarios with 1.55, which is 2% better than PSO and 21% better than BO. NSGAII further generates test cases with the lowest minimal fitness value of a median

Table 6.4.: For each sub-category of the presented problem, we show the performance of the optimization algorithms compared to one base algorithm and the P-value of a Mann-Whitney-U test for this comparison in brackets. We flag the base algorithm of each comparison with "Base". In addition, we highlight the P-values that show a significant difference for a 95% confidence interval in bold. Further, we present Vargha and Delaney's A_{12} measure below the performances and the category of their effect size with "-", "S", "M", or "L" presenting negligible, small, medium, or large. A previous version appeared in [116].

	Category (1)	**Category (2)**	**Category (2)**	**Category (2)**	**Category (3)**
NSGAII	Base	Base	—	—	Base
PSO	-2% (0.19)	—	Base	—	-2% (0.19)
	$A = 0.46$ (-)	—	—	—	$A = 0.46$ (-)
BO	-21% (**0.00**)	—	—	Base	-21% (**0.00**)
	$A = 0.26$ (**L**)	—	—	—	$A = 0.26$ (**L**)
NPN	—	+10% (0.05)	—	—	+10% (0.05)
	—	$A = 0.56$ (-)	—	—	$A = 0.56$ (-)
NPB	—	-12% (0.82)	—	—	-12% (0.82)
	—	$A = 0.49$ (-)	—	—	$A = 0.49$ (-)
NBN	—	+12% (0.41)	—	—	+12% (0.41)
	—	$A = 0.52$ (-)	—	—	$A = 0.52$ (-)
NBP	—	+5% (0.80)	—	—	+5% (0.80)
	—	$A = 0.51$ (-)	—	—	$A = 0.51$ (-)
PNP	—	—	+22% (**0.00**)	—	+20% (**0.01**)
	—	—	$A = 0.61$ (**S**)	—	$A = 0.57$ (**S**)
PNB	—	—	+6% (**0.01**)	—	+5% (0.25)
	—	—	$A = 0.56$ (**S**)	—	$A = 0.53$ (-)
PBN	—	—	+18% (**0.00**)	—	+16% (**0.04**)
	—	—	$A = 0.59$ (**S**)	—	$A = 0.56$ (**S**)
PBP	—	—	+11% (**0.00**)	—	+9% (0.13)
	—	—	$A = 0.58$ (**S**)	—	$A = 0.54$ (-)
BNP	—	—	—	+27% (**0.00**)	+11% (0.72)
	—	—	—	$A = 0.75$ (**L**)	$A = 0.51$ (-)
BNB	—	—	—	+13% (**0.00**)	-5% (0.05)
	—	—	—	$A = 0.69$ (**M**)	$A = 0.44$ (-)
BPN	—	—	—	+24% (**0.00**)	+8% (0.88)
	—	—	—	$A = 0.74$ (**L**)	$A = 0.50$ (-)
BPB	—	—	—	+9% (**0.00**)	-11% (**0.00**)
	—	—	—	$A = 0.64$ (**S**)	$A = 0.39$ (**S**)

of 0.38. However, this difference in the performance is only significant for NSGAII and BO according to our executed Mann-Whitney U test. Vargha and Delaney's A_{12} measure shows the same result and demonstrates that the difference in the performance of NSGAII and BO is large in effect size. These results indicate that NSGAII is more suitable than BO for generating test cases for our system and the tested logical scenarios. Even though NSGAII further presents better results than PSO, their performances are not significantly different. In future work, we aim to investigate the performance of these algorithms in additional logical scenarios and for various UAV systems to strengthen the significance and generalizability of the presented results.

Category (2) - Comparison of Combinations with Base Algorithms

After evaluating the performance of the given optimization algorithms individually, we are interested in exploring whether a sequential combination of algorithms can improve their performance. The combinations based on NSGAII are NPN, NPB, NBN, and NBP. The algorithms NPN, NBN, and NBP produce test cases with, on average, better minimal fitness values than NSGAII and achieve better performances of $5 - 12\%$. However, none of the results are significantly better than NSGAII, according to the Mann-Whitney U test and Vargha and Delaney's A_{12} measure. We can explore the reasons for this when investigating the logical scenarios individually. For the first and second logical scenarios, none of the combinations generates better median minimal fitness values than NSGAII. In contrast, all combined algorithms generate lower median minimal fitness values than NSGAII for scenario 5. These opposite observations show that the performance of the optimization algorithms varies for the different logical scenarios. They further emphasize the importance of focusing on the average performance of the algorithms. For the combinations based on PSO — PNP, PNB, PBN, PBP — we discover that all combined algorithms produce better median minimal fitness values than PSO, with an increased performance of $6 - 22\%$. This finding also holds when inspecting the logical scenarios separately, except for scenario 4, in which only PNP performs better. Overall, the Mann-Whitney U tests show that the performances of all four combinations are significantly better than the performance of PSO in our case study. The corresponding A_{12} measures show the same results with a small effect size. These results indicate that the combined versions are more suitable for finding "good" test cases for testing the UAV's safe behavior than PSO itself. BNP, BNB, BPN, and BPB are the evaluated combinations based on BO. On average, all of these combinations show significantly better results for the median minimal fitness value overall logical scenarios than BO, with an increased performance of $9 - 27\%$. Further, the A_{12} measures describe the severity of this significance with the comparisons with BNP and BPN having a large effect size, the one with BNB having a medium effect size, and the comparison with BPB having a small effect size. As for PSO, these observations strongly advise us to use one of the combined versions of BO instead of BO itself when generating "good" test cases for UAVs.

Category (3) - Comparison of All Algorithms

Finally, we are interested in comparing all evaluated optimization algorithms, including given and combined ones. Focusing on the median best fitness values reached throughout all logical scenarios, PNP performs best with a median value of 1.24 and a performance increase of 20% compared with NSGAII. The algorithm combination further achieves finding a worst-case situation in scenario 4, in which the tested UAV behaves unsafely. We can discover this unsafe behavior directly from the fitness value since the fitness value presents the remaining distance to an obstacle until the UAV violates the specified safety distance. Since the best fitness value generated is negative with -0.22, the UAV violates the safety distance by 0.22 meters in the created test case. This finding shows that PNP could create a worst-case situation that reveals unsafe behavior of the SUT, even though the results of NSGAII suggested that the SUT behaves safely in all situations of this logical scenario. These results strongly indicate that we should use various optimization algorithms when searching for worst-case situations for the SUT. In addition, PNP shows the lowest median absolute deviation of the minimal fitness values overall logical scenarios with 0.36, which presents its reliability in finding "good" test cases for the SUT. When considering the performance of PNP and NSGAII, a P-value of 0.01 shows a significant difference for a 95% confidence interval and the A_{12} measure presents a small effect size. Therefore, the results of our case study indicate that the combined optimization algorithm PNP is a reasonable choice for reliably generating "good" test cases for the SUT. Compared with NSGAII, which is currently used in literature, PNP shows a significantly increased performance of 20%, considering the quality of the created test cases.

Even though the algorithm combination PNP shows the best results in our case study, it nonetheless does not provide us a guarantee that it can find all worst-case situations. In addition, we can observe a variation in the quality of the generated test cases for different logical scenarios for all optimization algorithms, including PNP, which shows their heuristic and non-deterministic behavior. To gain more reliable results, we advise test engineers to execute various runs of generating "good" test cases for each logical scenario for their SUT to increase the chance of finding the most challenging worst-case situation. However, our case study also presents that this might still not be sufficient. Even though we perform fifteen runs per logical scenario, we can discover that the best-performing algorithm PNP does not detect several worst-case situations found by other optimization algorithms that reveal the unsafe behavior of the tested UAV. Table 6.3 shows that, e.g., NPN finds safety distance violations of the SUT in the logical scenarios 3 and 5 while PNP cannot discover these situations. These results demonstrate that we cannot blindly trust optimization algorithms to find worst-case situations when testing the safe behavior of UAVs. Even though it is well-known that heuristic optimization algorithms do not provide a guarantee to detect an optimal solution, our results show that non-optimality does not seem to be an exceptional case. In addition, the results demonstrate that safety violations are instead likely to be missed by some of the evaluated algorithms and that the outcome of different algorithms differs substantially, by up to 20%.

As presented in this work, a case study about the performance of optimization algorithms and their combinations indicates which algorithm generates "better" test cases for the SUT in relevant logical scenarios. However, it is crucial to note that even if we perform such a comparison to increase the possibility of finding worst-case situations, the resulting best-performing algorithm might still miss several of these worst-case situations. Having such a missing guarantee of finding the worst-case situation is problematic as we cannot ensure a safe behavior of the SUT in the tested logical scenarios. However, for the certification of autonomous systems, we should be able to present such a guarantee. Thus, to enhance the chance of finding worst-case situations, we should execute multiple optimization algorithms and combinations for all logical scenarios when testing the safe behavior of autonomous systems with scenario-based testing. We feel that the scenario-based testing community currently does not consider the cost and consequences of this essential insight that presents the need to execute multiple optimization algorithms to improve the search for worst-case situations. Further note that this crucial problem of generating worst-case situations for scenario-based testing is not restricted to our use case of UAVs but also applies to other domains such as autonomous driving. With the presented case study, we demonstrate that testing the safe behavior of autonomous systems with scenario-based testing faces the crucial challenge that optimization algorithms cannot provide a guarantee of finding worst-case situations.

Threats to Validity

In this case study, we evaluate the performance of different optimization algorithms for generating "good" test cases for five logical scenarios. Note that we do not claim that the best-performing optimization algorithm found in our experiments also operates best for other logical scenarios and SUTs, as we only evaluate the various optimization algorithms for generating "good" test cases for the PX4 autopilot. Instead, we emphasize the need to perform a similar case study as the presented one for relevant logical scenarios for each SUT specifically. Further, our evaluation results show the general problem of applying optimization algorithms for finding worst-case situations for autonomous systems. To acknowledge the randomness of the evaluated optimization algorithms and gain robust evaluation results, we perform fifteen runs for each logical scenario and investigate the median absolute deviation. However, we do not execute 30 runs as proposed by [5] due to limited computation power and time, as discussed in Section 6.3.1.

To reduce the threats to internal validity in our case study, we use existing libraries to execute the optimization algorithms and an open-source UAV. Further, we base the information-sharing process in the combination of algorithms on recommendations from the literature. In addition, we run all simulations in isolated Docker containers to decrease unwanted environmental effects. Finally, to achieve a fair comparison between the optimization algorithms, we use the same initial candidates for all algorithms per run and set the size parameters equally.

6.4. Conclusion

When applying heuristic optimization algorithms for generating "good" test cases for testing the safe behavior of UAVs, we encounter the problem that these algorithms cannot provide a *guarantee* to find the worst-case situation for the SUT. Due to this missing guarantee, we aim to evaluate the performance of various optimization algorithms and their sequential combinations when generating test cases for our use case. In our case study, we investigate the performance of three optimization algorithms — NSGAII, PSO, and BO — and their combinations when testing an open-source UAV in five different logical scenarios. We focus on inspecting the performance of (1) these three optimization algorithms, (2) the sequentially combined optimization algorithms compared with the algorithm we base them on, and (3) all evaluated optimization algorithms, including the given and combined ones. The results of our case study indicate that NSGAII is more suitable for our use case than BO when considering the three evaluated optimization algorithms individually for generating "good" test cases for our evaluated system and logical scenarios. Further results advise us to use the combined algorithms instead of PSO and BO directly as they, on average, produce "better" test cases. Finally, the results show that PNP presents, on average, the best performance overall runs and logical scenarios. The algorithm achieves an increased average performance of 20% compared to NSGAII in our case study. With PNP, we even generate test cases that reveal the unsafe behavior of the tested UAV in one of the tested logical scenarios. However, our case study also reveals the high variation of quality of the generated test cases for different logical scenarios for all optimization algorithms, including PNP. To increase the chance of finding the worst-case situation, we encourage test engineers to repeat the generation of "good" test cases for each logical scenario several times. However, our case study demonstrates that this might still not be sufficient for finding all unsafe behaviors of the SUT. Even though it is known that heuristic search algorithms do not provide optimality guarantees, our results show that non-optimality does not seem to be an exceptional case. In addition, the results demonstrate that safety violations are instead likely to be missed by some of the optimization algorithms and that the outcome of different algorithms differs substantially. If we miss worst-case situations to reveal potential faults in the SUT, we cannot guarantee a safe behavior of this system in the evaluated logical scenarios. However, for the development and certification of autonomous systems, a guarantee of the safe behavior of these systems is essential. For these reasons, scenario-based testing comes at the *extra cost of having to run multiple optimizers* to increase the chance of detecting worst-case situations. We feel that this insight and its consequences are somewhat lost in the scenario-based testing community and that they challenge the current widespread application of scenario-based testing for ensuring the safe behavior of autonomous systems such as ADS or UAVs.

In future work, we plan to extend the case study with further runs per logical scenario to more reliably explore the statistical significance of our results. Also, we aim to evaluate the performance of other optimization algorithms for different UAVs and other autonomous systems to strengthen the significance and generalizability of the presented results.

7. StellaUAV: A Tool for Testing the Safe Behavior of UAVs

This chapter presents the tool StellaUAV that implements the proposed approach of testing the safe behavior of UAVs with scenario-based testing and an evaluation of its applicability and effectiveness for generating test cases that can reveal potential faults in UAVs. Parts of this chapter previously appeared in a peer-reviewed publication [116] co-authored by the author of this thesis.

7.1. Introduction

When we aim to test the safe behavior of UAVs with scenario-based testing and the presented methodology in this work, we need a tool to generate "good" test cases and evaluate various optimization algorithms. With such a tool, we can systematically apply the shown methodology and (1) define logical scenarios to test the UAV's safe behavior, (2) generate worst-case situations for each of these logical scenarios for the SUT, and (3) compare the performance of various optimization algorithms for the presented use case.

Related work presents frameworks [66, 141] that concentrate on testing path planning algorithms as one part of UAVs or focus on testing the reliability of these systems instead of their safety. The authors of [142] present a tool that uses belief state machines to create test cases for testing the reliability of cyber-physical systems. Even though all three tools are essential for testing UAVs, they do not focus on testing the safe behavior of UAVs, which is another crucial aspect when testing autonomously operating UAVs. In [71], the authors present a framework that applies metamorphic and model-based testing to test the stability of a controller for UAVs. However, when using metamorphic testing, we need to specify the corresponding correct and complete metamorphic relations. As we cannot easily accomplish this, we focus on applying scenario-based testing in this work as the specification of logical scenarios for a SUT is more often performed and more intuitive. The authors of [30] present a framework that uses fault-injection methods to test the safe behavior of their UAV autopilot. However, with such fault-injection methods, we can only detect known faults. To ensure the safe behavior of UAVs, it seems favorable to discover unknown faults likewise. Finally, note that we cannot directly apply tools and frameworks for testing ADS as presented in [65, 76] due to the mentioned differences between using scenario-based testing for testing ADS and UAVs.

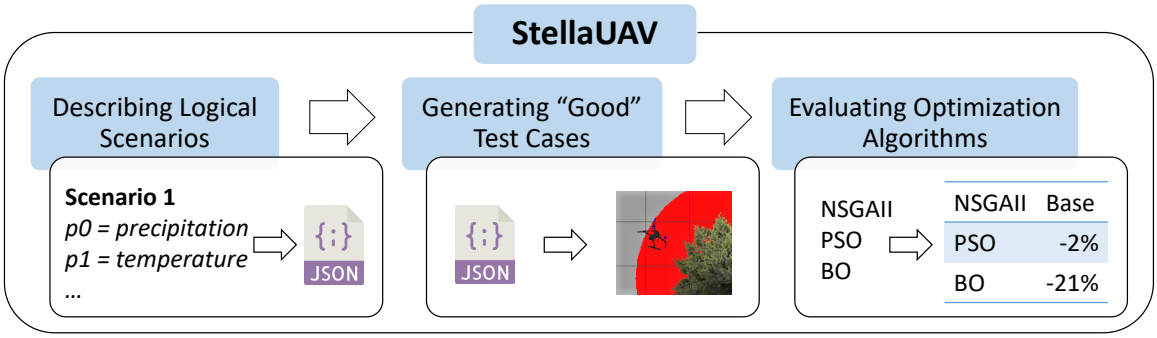


Figure 7.1.: For testing the safe behavior of UAVs, StellaUAV includes three primary use cases: describing logical scenarios to test, generating "good" test cases, and evaluating the performance of various optimization algorithms. A previous version appeared in [116].

The **contribution of this chapter** is a tool called StellaUAV, which enables us to generate "good" test cases for testing the safe behavior of UAVs in various logical scenarios. The tool offers the possibility to define these different logical scenarios before creating "good" test cases for each of them. In addition, the user can specify the optimization algorithms to apply for generating worst-case situations and, thus, perform a case study of different algorithms, as we have shown in the previous chapter.

7.2. Methodology

As presented in the introduction, StellaUAV has three essential use cases, which we explain in more detail in the remainder of this section: (1) defining logical scenarios to test the UAV's safe behavior, (2) generating worst-case situations for each of these logical scenarios, and (3) comparing the performance of various optimization algorithms. We also depict these use cases in Fig. 7.1. When executing StellaUAV, the user can select several parameters in the presented GUI: the logical scenarios in which the user aims to test the UAV, the maximal number of evaluations that the optimization algorithms perform to find "good" test cases, the size parameter for the optimization algorithms, and the algorithms themselves.

In StellaUAV, we present the manually defined logical scenarios to test in a JSON format, which offers various advantages, as described in Section 3.3.2. The tool can work with logical scenarios that represent simulation worlds with different landforms, surface natures, kinds, sizes, and forms of obstacles, wind forces, and reduced visibilities. We showed examples of these simulation worlds in Fig. 3.6. For each logical scenario, the tested UAV gets the mission to fly to a specified waypoint, which depends on the landform of the logical scenario. In Fig. 7.2, we present an example of a logical scenario defined for StellaUAV. Note that this example includes parameter values that implicitly denote parameter ranges, e.g., moderate wind force stands for a wind force of $4.0 - 8.0$ kilometers per hour, and a

```
{
    "name": "Test Scenario #1",
    "system": {
        "UAVs": [{"maneuvers": ["move to waypoint"]}]
    },
    "environment": {
        "flight area": {
            "landform": "flat",
            "surface nature": "land"
        },
        "obstacles": [
            {
                "kind": "dynamic",
                "size": "medium",
                "form": "sphere"
            },
            {
                "kind": "static",
                "size": "large",
                "form": "cuboid"
            }
        ],
        "weather": {
            "wind force": "moderate",
            "reduced visibility": "heavy_fog",
            "lighting": "normal",
            "temperature": "moderate",
            "precipitation": "none",
            "lightning": "none",
            "cloud cover": "none"
        }
    }
}
```

Figure 7.2.: Examplary JSON file that defines a logical scenario for evaluating the UAV's safe behavior in StellaUAV. A previous version appeared in [116].

medium-sized obstacle is $5.0 - 10.0$ meters large. Thus, this example presents a logical and not a concrete scenario. Further, the presented logical scenario adheres to the ontology presented in Section 3.3.2 and the JSON schema derived from this ontology that we depict in Appendix A.

Before StellaUAV generates "good" test cases for the manually specified logical scenarios, it first derives the search space from these logical scenarios and the corresponding fitness function. The value ranges of the parameters P of each logical scenario present

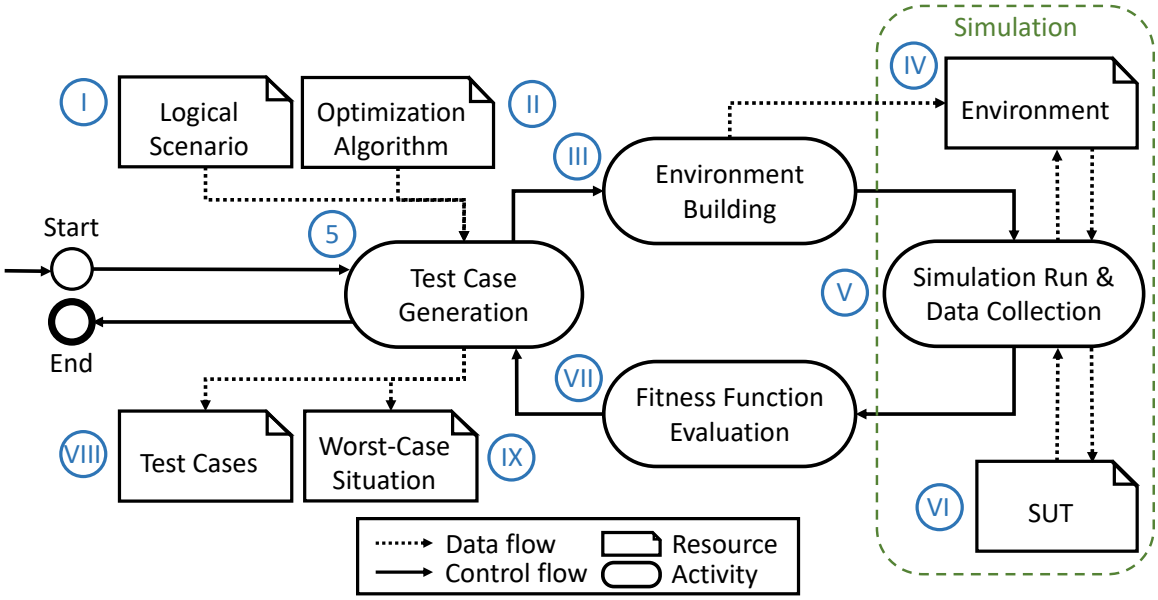


Figure 7.3.: The methodology for generating "good" test cases applied in StellaUAV that denotes step (5) in Fig. 5.2. A previous version appeared in [116].

the dimensions of the search space. In addition, we add the positions and velocities of the obstacles to this search space. As the fitness function, StellaUAV uses the function presented in Eq. (5.2). Note that this fitness function offers an automatic oracle as a negative fitness value represents a safety distance violation to an obstacle in the UAV's environment. When searching for worst-case situations, StellaUAV follows the methodology introduced in Chapter 5. We also present a focused view on this applied methodology in Fig. 7.3 and visualize which steps are executed in simulation. The logical scenario (I) to evaluate and the optimization algorithm (II) to execute are input to the test case generation step (5). For each generated test case, StellaUAV builds in step (III) the simulation environment (VI) before providing the SUT with its mission. During the simulation (V), StellaUAV collects data about the UAV (VI) to evaluate its safe behavior in the fitness function afterwards (VII). After assessing the initial candidates, the optimization algorithm generates further test cases with the goal of discovering "better" ones that present worst-case situations. Finally, StellaUAV returns the generated test cases (VIII) and the detected worst-case situations (IX) for the evaluated logical scenarios.

Finally, StellaUAV presents several optimization algorithms to compare their performance for testing the safe behavior of UAVs. In the previous chapter, we presented such a case study for testing the safe behavior of the PX4 autopilot. StellaUAV includes implementations of NSGAII, PSO, and BO and their sequential combinations that we introduced in the previous chapter: NPN, NPB, NBN, NBP, PNP, PNB, PBN, PBP, BNP, BNB, BPN, BPB; To gain reliable results and reduce the effect of randomness on the performance of the

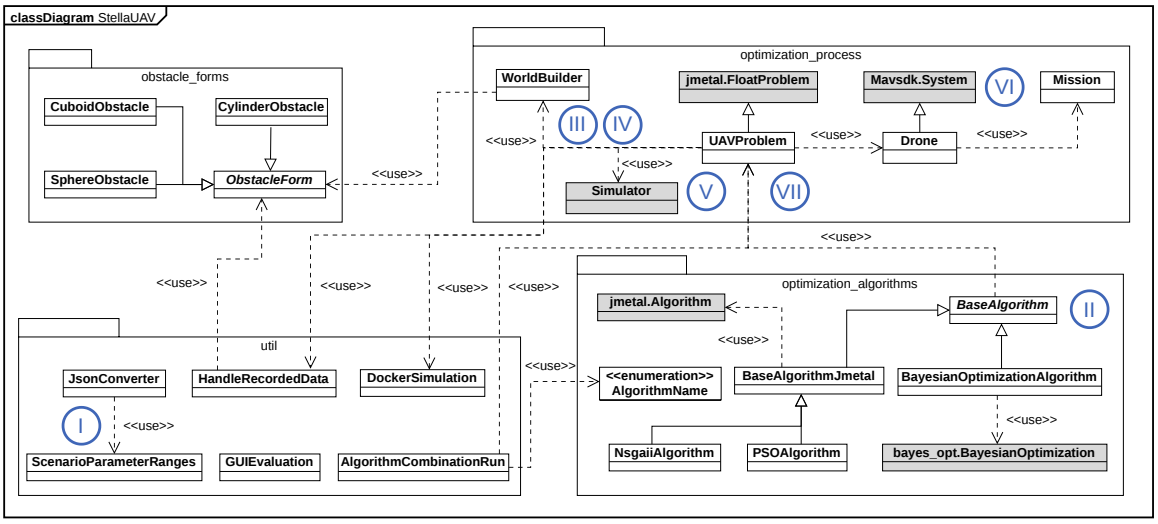


Figure 7.4.: The architecture of StellaUAV with steps from Fig. 7.3 and third-party frameworks and libraries marked in gray. A previous version appeared in [116].

optimization algorithms, we encourage test engineers to perform several runs of generating "good" test cases for each logical scenario and optimization algorithm.

7.3. Architecture

In Fig. 7.4, we present the architecture of our tool StellaUAV and corresponding steps from Fig. 7.3. It is composed of four main packages: *obstacle_forms*, *optimization_process*, *optimization_algorithms*, and *util*; The first package *obstacle_forms* includes implementations of the various obstacles in StellaUAV. In the current implementation, we focus on cuboid, spherical, and cylindrical obstacles which inherit attributes and methods from the abstract class ObstacleForm. When adding another obstacle form, one needs to implement a new class that inherits from this abstract class and includes methods for updating its position and calculating the distance to this new form of obstacle.

The package *optimization_process* encompasses classes for evaluating the safe behavior of a UAV in simulation. In StellaUAV, we use the open-source simulator Gazebo [61], which enables us to simulate various kinds of robots, including terrestrial, aerial, and underwater ones. In addition, we can set different environmental effects in Gazebo, such as the wind force or fogginess. In the *optimization_process* package, we specify the simulation world (IV) for Gazebo in a .world file. The class WorldBuilder generates this file (III) based on the definition of the logical scenarios (I) that we currently evaluate. In the current implementation, this class adapts the landform, the water level, the type, size, and form of obstacles, the wind force coming from one direction, and the fog to reduce the UAV's visibility. To adjust

additional elements of the world, one can extend the *adapt()* function of the WorldBuilder class. In the class UAVProblem, we formulate the search problem for our search for "good" test cases with the search space and the fitness function. We inherit from the FloatProblem class of the jmetalpy framework [15] to enable the usage of their optimization algorithms. This framework presents implementations of various optimization algorithms and search problems to evaluate with these algorithms. The applied optimization algorithms call the *evaluate()* function of this class to start the simulation of the UAV (V) and evaluate its safe behavior (VII). Finally, we define a Drone class that inherits from the MAVSDK-Python library [92] to enable communication with the simulated UAV (VI) via MAVSDK as well as upload and start missions on the UAV. The MAVSDK library offers APIs for communicating with MAVLink systems by sending them missions or controlling their movements.

In the package *optimization_algorithms*, we define the various optimization algorithms (II) that the user can select to generate "good" test cases. The current implementation of StellaUAV provides the optimization algorithms NSGAII, an enhanced version of PSO called Speed-constraint Multiobjective PSO, and BO. For NSGAII and PSO, we use the implementations of the jmetalpy framework, and for BO's implementation, we utilize the BayesianOptimization framework [91]. All optimization algorithms inherit from the abstract class BaseAlgorithm. To add another optimization algorithm, we need to create a new class that inherits from this abstract class and specifies methods for parameterizing and executing this algorithm.

Finally, package *util* includes several utility files that, i.a., convert the JSON files in which we specify logical scenarios into parameter ranges for the definition of the search space or evaluate the safe behavior of the UAV by investigation data that we recorded during the simulation.

Limitations and Extensibility With StellaUAV, we can generate "good" test cases for testing autonomously operating UAVs. To apply StellaUAV for testing a specific UAV, this UAV needs to provide communication interfaces via the MAVSDK library to allow transmission of the UAV's mission and ROS to enable data collection during the simulation. For exchanging the currently implemented PX4 autopilot [81] in StellaUAV that we use in our experiments and will describe in Section 7.4.1, we have to modify the script that starts the simulation of the UAV and the roslaunch files that launch the simulation environment with the UAV. Except for these two files, we do not need to alter the source code of StellaUAV to exchange the SUT. As a simulator, StellaUAV currently applies the open-source simulator Gazebo, which enables us to define various landforms, surface natures, obstacles, wind forces, and fogginess for the UAV's environment. However, we are limited to these capabilities and lack the option to represent, e.g., ambient temperature, cloudiness, or various degrees of rain. To exchange Gazebo with another simulator, we need to adjust the roslaunch files to launch the other simulator and adapt the class WorldBuilder to adhere to specifying the simulation world for the new simulator. Finally, note that we apply our methodology in this work in the context of Software-in-the-Loop testing but can also utilize it for Hardware-in-the-Loop or Model-in-the-Loop testing.

7.4. Evaluation

In this section, we evaluate the presented tool StellaUAV by investigating the safe behavior of the PX4 autopilot [81] in a Software-in-the-Loop simulation in 32 different logical scenarios. We share the source code for our presented tool StellaUAV in [117].

7.4.1. System Under Test

In StellaUAV, we generate test cases for testing the safe behavior of the obstacle and avoidance extension of the PX4 autopilot [81], which is an open-source project with a large community that works on it. Note that the authors of this work did not implement this autopilot. We can operate this autopilot autonomously by presenting it with a mission, e.g., to fly to a defined waypoint. The tested autopilot autonomously plans its trajectory to fulfill its mission and automatically adapts it to upcoming obstacles or environmental conditions that affect its behavior.

7.4.2. Setup and Implementation

As mentioned before, StellaUAV uses the simulator Gazebo [61] to simulate the PX4 autopilot. During simulation, we forward the mission to fly to a specified target point to the UAV via the MAVSDK-Python library [92] and gather data about the UAV via ROS [100]. To evaluate the safe behavior of the SUT in our experiments, we apply the fitness function presented in Eq. (5.2), where we fix the safety distance $s(cs, t)$ to 1.0 meters for simplicity of presentation. Accordingly, we set the corresponding safety distance parameter of the tested PX4 autopilot to the same value. As we present a case study that evaluates the performance of various optimization algorithms in Chapter 6, we focus on the optimization algorithm NSGAII in the experiments of this chapter. We selected NSGAII as it is currently widely used for testing the safe behavior of autonomous systems [48, 114, 147]. In our experiments, we set the maximum number of evaluations to 500 for each logical scenario, as suggested in [60], and the population size of NSGAII to 100. We apply SBX Crossover, Binary Tournament Selection, and polynomial mutation operators with a crossover rate of 0.9 and a mutation rate of one divided by the number of variables in the search space, as proposed by [19]. Due to the heuristic nature of the optimization algorithms, we execute three runs to search for "good" test cases for each logical scenario to reduce the randomness of the results. Note that we do not perform 30 runs as proposed by [5] due to time constraints. However, as we do not present an empirical assessment of different optimization algorithms in these experiments, three runs seem sufficient to demonstrate the general applicability of StellaUAV. Note that we run all simulations in Docker containers to decrease unwanted environmental effects. To evaluate the safe behavior of the UAV in realistic concrete scenarios that enable the UAV to react reasonably, we exclude concrete scenarios with dynamic obstacles with very high velocities. If the obstacles move too fast and the SUT cannot respond to them, we cannot blame the SUT when violating any safety

distances. Thus, we need to ensure that the SUT can theoretically keep the specified safety distance by limiting the velocity of the obstacles to a reasonable value. This limitation presents one of our constraints to ensure the realism of the generated concrete scenarios. Further, we assure that the UAV performs the predefined mission of the logical scenarios in the created concrete scenarios. Due to the limitations of our applied simulator, which we discussed in Section 7.3, we only represent a few environmental effects relevant to UAVs and represent real-world obstacles in an approximated form. Regardless, StellaUAV still enables us to define and test the safe behavior of UAVs in logical scenarios with static and dynamic obstacles and relevant environmental effects, such as wind and fog. Finally, note that the dynamic obstacles in StellaUAV currently move straight between two points with a certain velocity. In future work, we would like to extend StellaUAV with more sophisticated maneuvers for the dynamic obstacles. To enable the reproducibility of these experiments, we provide additional details on the experiment settings as, e.g., the version number of the used libraries, in Appendix B and present the generated minimal fitness values for all logical scenarios and runs for these experiments in [112].

7.4.3. Logical Scenarios

In Tables 7.1 and 7.2, we present the logical scenarios that we evaluate in our experiments. We derived these logical scenarios manually from the ontology introduced in Section 3.3.2. For selecting logical scenarios from this ontology, we apply the simple defect hypothesis that a pair-wise combination of the ontology's dimensions is sufficient to provoke all relevant faults in the SUT. However, note that we chose this defect hypothesis for simplicity of presentation and that it does not present a complete and sufficient defect hypothesis for the tested system. The resulting logical scenarios include different landforms, surface natures, wind forces, reduced visibilities, and a varying number of obstacles in various forms, sizes, and kinds.

7.4.4. Experimental Results & Discussion

We present the results of our experiments to show the applicability of StellaUAV in Table 7.3. In this table, we depict the number of created test cases in which the SUT shows an unsafe behavior by violating the defined safety distance for each logical scenario and each of the three runs. Further, we denote the average and median values over all runs. For five of the 32 logical scenarios, we cannot find any safety distance violations in our experiments. In the other logical scenarios, we generate several test cases with negative fitness values that represent safety distance violations in the experiments. An inspection of these logical scenarios reveals that all of them include dynamic obstacles, whereas those in which the UAV only shows a safe behavior consist solely of static obstacles. This observation indicates that the SUT can safely handle the static obstacles in the tested logical scenarios but encounters problems when faced with dynamically moving ones. In addition, the

Table 7.1.: The parameter values for the landform, surface nature, wind force, and reduced visibility of the 32 logical scenarios in our experiments.

Scenario	Landform	Surface Nature	Wind Force	Reduced Visibility
1	flat	mixture	strong	thick fog
2	depression	land	light	heavy fog
3	elevation	water	moderate	fog
4	steep transition	mixture	none	none
5	flat	land	strong	none
6	elevation	water	none	fog
7	depression	water	moderate	heavy fog
8	steep transition	land	moderate	thick fog
9	flat	water	light	fog
10	flat	mixture	none	thick fog
11	steep transition	land	strong	heavy fog
12	depression	land	strong	heavy fog
13	elevation	mixture	light	fog
14	steep transition	water	moderate	none
15	flat	land	light	thick fog
16	depression	mixture	light	none
17	depression	land	none	fog
18	depression	water	strong	none
19	elevation	mixture	moderate	heavy fog
20	elevation	land	none	none
21	flat	mixture	strong	fog
22	steep transition	water	light	thick fog
23	flat	land	none	heavy fog
24	steep transition	mixture	strong	none
25	flat	land	moderate	thick fog
26	depression	water	light	thick fog
27	depression	land	light	heavy fog
28	elevation	mixture	moderate	thick fog
29	elevation	water	strong	thick fog
30	steep transition	water	light	fog
31	flat	land	moderate	heavy fog
32	steep transition	mixture	none	none

Table 7.2.: The parameter values for the number of obstacles, their kinds (static ST or dynamic DY), sizes (small S, medium M, or large L), and forms (cuboid CU, sphere SP, or cylinder CY) of the 32 logical scenarios in our experiments.

Scenario	# Obst.	Obstacle Kinds				Obstacle Sizes				Obstacle Forms			
1	4	DY	DY	DY	DY	S	S	S	S	CY	SP	CY	CY
2	3	ST	ST	ST	—	L	M	M	—	SP	CY	CU	—
1	4	DY	DY	DY	DY	S	S	S	S	CY	SP	CY	CY
2	3	ST	ST	ST	—	L	M	M	—	SP	CY	CU	—
3	1	DY	—	—	—	M	—	—	—	CU	—	—	—
4	4	ST	DY	ST	ST	M	L	L	M	CU	CU	SP	CU
5	2	ST	ST	—	—	S	L	—	—	CY	CU	—	—
6	4	DY	ST	DY	ST	L	M	L	L	SP	SP	CY	SP
7	4	DY	DY	DY	DY	M	L	M	M	CY	CY	CU	SP
8	4	ST	ST	ST	DY	L	S	S	L	CU	CY	SP	CY
9	4	DY	DY	ST	DY	S	M	M	S	SP	CU	SP	CU
10	1	ST	—	—	—	L	—	—	—	SP	—	—	—
11	1	DY	—	—	—	S	—	—	—	CY	—	—	—
12	4	ST	ST	DY	ST	M	S	L	S	CU	SP	CU	CU
13	4	ST	ST	ST	ST	S	M	S	M	CY	CY	CU	CY
14	3	ST	DY	DY	—	M	S	S	—	SP	SP	CY	—
15	4	ST	DY	DY	ST	M	L	M	L	CU	CU	CY	SP
16	1	DY	—	—	—	S	—	—	—	CU	—	—	—
17	2	DY	DY	—	—	L	S	—	—	CY	CY	—	—
18	4	ST	ST	ST	DY	L	L	L	M	SP	CY	CY	CY
19	4	DY	DY	DY	ST	S	S	M	L	CY	CU	SP	CU
20	4	ST	DY	ST	DY	L	L	S	S	CY	CU	CU	SP
21	3	DY	ST	ST	—	S	L	L	—	CY	SP	SP	—
22	2	ST	ST	—	—	M	M	—	—	CU	SP	—	—
23	4	DY	ST	DY	ST	S	S	M	M	SP	SP	CY	CY
24	4	DY	ST	DY	DY	S	M	M	L	SP	SP	CU	SP
25	4	ST	DY	DY	DY	L	M	L	S	CU	CY	CU	CU
26	4	ST	ST	DY	ST	S	S	L	M	CY	CU	SP	SP
27	4	DY	DY	DY	DY	S	M	S	L	CY	SP	CY	CU
28	2	DY	ST	—	—	S	M	—	—	SP	SP	—	—
29	3	DY	ST	DY	—	L	S	S	—	CU	CU	CY	—
30	4	DY	DY	DY	DY	M	L	M	S	CU	CU	CY	CY
31	2	DY	DY	—	—	M	M	—	—	SP	CY	—	—
32	3	DY	ST	ST	—	L	L	S	—	SP	SP	CU	—

Table 7.3.: The number of test cases in which the SUT violates the specified safety distance in our experiments for three runs and their average and median values.

Logical Scenario	# Safety Distance Violations				
	Run 1	Run 2	Run 3	Average	Median
1	104	116	61	93.67	104
2	0	0	0	0.00	0
3	97	69	16	60.67	69
4	12	22	5	13.00	12
5	0	0	0	0.00	0
6	112	163	151	142.00	151
7	18	7	14	13.00	14
8	34	25	25	28.00	25
9	209	222	234	221.67	222
10	0	0	0	0.00	0
11	37	31	31	33.00	31
12	9	18	2	9.67	9
13	0	0	0	0.00	0
14	10	12	14	12.00	12
15	128	124	133	128.33	128
16	7	18	37	20.67	18
17	58	17	50	41.67	50
18	6	3	1	3.33	3
19	198	211	169	192.67	198
20	153	180	135	156.00	153
21	23	1	10	11.33	10
22	0	0	0	0.00	0
23	177	177	143	165.67	177
24	24	39	10	24.33	24
25	123	185	149	152.33	149
26	36	132	83	83.67	83
27	12	8	6	8.67	8
28	167	180	160	169.00	167
29	61	53	12	42.00	53
30	38	42	20	33.33	38
31	229	260	239	242.67	239
32	126	140	111	125.67	126

results demonstrate the effectiveness and applicability of StellaUAV for creating "good" test cases that reveal potential faults in the SUT.

Threats to Validity

Our experimental results focus on showing the applicability of the presented tool for the integrated PX4 autopilot by generating "good" test cases for this system in several logical scenarios. Note that we do not present results for other systems and limit ourselves to the presented logical scenarios in these experiments. However, we evaluate the tool's effectiveness in 32 logical scenarios that represent various environmental conditions for the UAV. Note that we do not claim to demonstrate a complete list of relevant logical scenarios for the SUT with these 32 logical scenarios. Since our tool uses existing open-source libraries, tests the safe behavior of the open-source PX4 autopilot, and runs all simulations in isolated Docker containers, we reduce the threats to internal validity.

7.5. Conclusion

For systematically applying the presented methodology in this work for testing the safe behavior of UAVs with scenario-based testing, we require a tool that enables (1) the definition of logical scenarios to test the UAV's safe behavior, (2) the generation of worst-case situations for each of these logical scenarios, and (3) the comparison of the performance of various optimization algorithms for the presented use case. As a contribution, this chapter presents such a tool named StellaUAV. The tool describes the logical scenarios to test in JSON files before deriving the search space and fitness function for these logical scenarios. Afterward, StellaUAV applies a specified optimization algorithm to search for worst-case situations in the defined search space. The tool enables the user to specify the logical scenarios to test, the optimization algorithms to use for the search, and different parameters for these algorithms. Finally, we show an evaluation of the applicability and effectiveness of the presented tool by searching for safety distance violations in 32 logical scenarios. These experiments show that the tested PX4 autopilot can handle static obstacles very well, whereas it encounters problems with dynamically moving ones. Thus, these results show the effectiveness of StellaUAV when creating "good" test cases that can reveal potential faults in the SUT.

In future work, we would like to extend StellaUAV to enable the possibility of testing various kinds of UAV autopilots, including additional environmental effects, and introducing more complex maneuvers for the included dynamic obstacles.

Part IV.

Related Work and Conclusion

8. Related Work

This chapter discusses related work about testing the safe behavior of UAVs with its challenges of deriving relevant logical scenarios for these systems and generating worst-case situations for them. Parts of this chapter previously appeared in peer-reviewed publications [114, 115, 116] and a peer-reviewed submission under review [118] co-authored by the author of this thesis.

8.1. Testing the Safe Behavior of UAVs

For testing the safe behavior of UAVs, literature presents various methods. One approach is to identify hazards and risks for UAVs, as proposed by [14, 27, 80, 93, 134]. The authors either derive the safety hazards from expert knowledge, a failure analysis, or experimental flight tests in specific conditions, such as occurring motor or GPS failures. Primary hazards presented in these papers are midair collisions [27, 80, 134], collisions with objects or persons located on the ground [14, 80, 134], severe weather conditions [134], and loss of control [14, 93, 134]. Addressing the detected hazards should be the first but not the last step when testing the safe behavior of UAVs. The derived hazards or logical scenarios present a limited subset of all relevant logical scenarios for the SUT. Further, the demonstrated approach does not include a methodology to generate "good" test cases for each of the detected logical scenarios to reveal the potentially faulty behavior of the SUT. In this work, we present an ontology to characterize relevant logical scenarios for the SUT and outline a methodology for generating "good" test cases for each of them with search-based techniques.

Reachable set analysis is another approach to evaluate the safe behavior of UAVs, as suggested by [43, 70, 144]. The idea of this approach for safety evaluations of systems is the following: we consider a SUT to behave safely if it does not enter a set of undesirable states during its operation. Literature defines these undesired states as the reachable set of other UAVs and obstacles [70, 144] or by setting boundaries for relevant parameter values [43]. We need to perform this partitioning into safe and unsafe states correctly for all possible configurations to ensure the safety of the SUT. Even though reachable set analysis has the advantage of ensuring a safe behavior of the regarded system in real-time, the verification of the correct partitioning between safe and unsafe states is not easily accomplished. We need to perform this additional verification step also for several other approaches, such as simulation-based testing. When using simulations, we rely on the correctness of the simulations, which present an abstraction of the SUT. However, there exists an extensive

amount of research about the correctness of simulations [58, 108, 109]. Thus, we decided to work with simulations instead of using reachable set analysis since we do not aim to test the behavior of UAVs in real-time. Due to the existing high effort in verifying and improving the quality of simulations in the research community, we use existing tools for simulating UAVs and do not incorporate research about further improvements of simulations into this thesis. Thus, we can focus on finding "good" test cases with the help of those simulations instead of having to verify the abstraction of the system ourselves.

Another advantage of simulation-based testing compared with real-world flight tests, as proposed by [55, 57, 84], is that they cause no damage or injury when the SUT behaves unsafely and collides with persons or objects in its environment. Since we are interested in testing UAVs in safety-critical situations, simulation-based techniques are, thus, well suited for our approach. When only applying real-world flight tests, we face the problem of testing only specific concrete scenarios that might not present the most challenging ones. Thus, we need to develop and apply additional methods for ensuring that the UAVs behave safely, even in the worst-case situation of each logical scenario. As scenario-based testing [23] has shown to be an insightful approach for testing the safe behavior of ADS [48, 82, 89, 126], we propose to apply the same method for testing the safe operation of UAVs.

Summary: Related work on testing the safe behavior of UAVs focuses on testing the system in specific concrete scenarios or logical scenarios without basing their selection on suitable defect hypotheses. For other approaches, we need an additional verification step of the abstraction of the SUT that is not easily accomplished.

8.2. Logical Scenario Derivation

In the literature, several works present use case scenarios for UAVs [36, 72, 110]. However, their main focus lies on the mission of the UAVs without concentrating on their environment. If the use case scenarios include descriptions of the environment, they often focus on specific areas, such as coastal settings [36] or desert environments [72]. To define logical scenarios for testing the safe behavior of UAVs in more detail, we present an ontology that describes the UAVs, their missions, and detailed information about their environment in this thesis.

The authors of [10, 90] apply a similar approach by modeling the logical scenarios for ADS with ontologies. However, [10] focuses on the temporal and spatial causalities of logical scenarios for ADS, and [90] concentrates on describing the objects and their relationships in the logical scenarios as well as road and weather conditions. Both works present meta-models of logical scenarios for ADS without concrete dimensions that prevent us from directly deriving specific logical scenarios. In addition, both papers discuss logical scenarios which present challenging situations for ADS. Due to missing fine-grained traffic rules and rigid road structures, demanding situations for UAVs look different than those for ADS. Further, to the best of our knowledge, no test-ending criterion for testing the safe behavior of UAVs with scenario-based testing exists that would inherently present a complete list of all relevant logical scenarios. To close these gaps in the literature, we introduce an

ontology in this work that includes concrete dimensions to enable a direct derivation of logical scenarios for testing the safe behavior of UAVs.

When creating an ontology that represents logical scenarios for UAVs, we first need to find the relevant dimensions of these scenarios. As the existing ontologies for UAVs [22, 51, 124] present dimensions for specific use cases such as UAV video content analysis, available aircraft resources, and UAV flight control and management systems, we use them as an inspiration for our ontology. In addition, we gathered ideas from ontologies of other domains, such as [11, 59].

When dividing the collected dimensions of the ontology into suitable sub-categories, several works [46, 47] emphasize the importance of achieving the correct level of granularity. Contrary to [90], we aim to find this granularity level for logical scenarios themselves instead of distinguishing between scenes, situations, scenarios, and use cases. We believe that we need to derive the sub-categories and, thus, the correct level of granularity system-specifically, as different situations are challenging for various UAVs. To describe all relevant logical scenarios for UAVs, we need to find suitable lower and upper bounds for the dimensions of our ontology before dividing each of them into sub-categories. Thus, we describe an automated approach for detecting these bounds in this work to present a limited number of test scenarios. Other methods for finding a complete list of logical scenarios for autonomous systems focus on gathering them from collected real-world data [31, 49, 135]. To state whether we discovered all relevant logical scenarios for the SUT, these approaches either analyze the density or diversity of the gathered data or use an instance of the Coupon Collector's Problem to detect whether additional data needs to be collected. However, in the context of UAVs, we lack the high amounts of data necessary to apply these approaches. Thus, we focus on an ontology based on mental models that represent logical scenarios in this work instead of concentrating on statements about the completeness of the derived logical scenarios.

Next, we aim to describe specific logical scenarios in a machine-readable. The authors of [7, 99] present languages to denote concrete or logical scenarios for ADS, whereas the Aviation Scenario Definition Language [53] introduces a language to define aircraft landing scenarios. As all of the presented languages do not include the needed attributes to describe all relevant logical scenarios for testing the safe behavior of UAVs, we propose to define these logical scenarios in JSON.

Finally, we need a method to select specific logical scenarios from the presented ontology. The authors of [68] suggest the use of combinatorial testing for an ontology that describes the environment of autonomous vehicles. However, we believe that such a selection method should be chosen system-specifically and based on a defect hypothesis of challenging situations for the SUT.

Summary: Related work on logical scenarios for UAVs focuses on specific use cases and the mission of the UAVs and misses an in-depth description of its environment. As different situations are challenging for ADS and UAVs, we cannot directly use existing work about meta-models of logical scenarios from the automotive domain that furthermore lack concrete dimensions.

8.3. Generating "Good" Test Cases

When testing the safe behavior of ADS, related work suggests the use of search-based techniques to find challenging situations for the SUT [48, 137]. In [137], the authors present the application of this approach for testing the safe behavior of an automated parking system, whereas [48] focuses on the fitness functions needed to create "good" test cases for ADS. Since we can encounter different challenging situations for ADS and UAVs, we cannot directly apply these methods. In this work, we propose a methodology for generating "good" test cases for testing the safe behavior of UAVs, which is based on the presented literature but tackles the issues specific to UAVs. In addition, since we lack fine-grained traffic rules and rigid road structures for UAVs, the explicit specification of a safe behavior for UAVs gets more challenging. To acknowledge this challenge, we consider two cases in this work. In the first case, we assume that we have a specified safety distance that we can use in our fitness function to explicitly define the safe behavior of the SUT. In the second case, we work without a specified safety distance and adapt our search for challenging situations accordingly.

The authors of [146, 147] apply search-based techniques to find collisions between multiple UAVs in an environment with no further obstacles. In addition, the authors test against a specified safety distance to evaluate the safe behavior of the UAVs. However, the environment of UAVs has a high impact on their behavior, and assuring a safe behavior of UAVs in urban areas will be an essential part of their deployment. Thus, we provide a methodology for generating "good" test cases for testing the safe behavior of UAVs that considers different environmental effects. In addition, we also investigate the case when we cannot specify a safety distance, as mentioned before.

Summary: Related work on testing the safe behavior of autonomous systems with search-based techniques focuses on testing ADS that encounter different challenging situations than UAVs or on testing UAVs in an environment with no obstacles. In addition, the literature concentrates on testing against a specified safety distance and neglects the case when we cannot define such a safety distance.

8.4. Evaluation of Optimization Algorithms

In the literature, several works [9, 29, 64, 125] evaluate the performance of optimization algorithms in different contexts, such as managing hydro-powered plants, allocating land multi-objectively, solving kinematic equations for robot manipulators, or clustering in wireless sensor networks. The authors of [60] focus on the convergence rate of different optimization algorithms when creating "good" test cases for testing the safe behavior of ADS. Contrary to this work, we aim to concentrate our evaluation on the algorithms' ability to find worst-case situations. In addition, we cannot easily generalize the results from case studies from other domains as the performance of optimization algorithms is highly context-specific.

We base our empirical case study of the performance of optimization algorithms for finding worst-case situations on the good practices presented in [13, 101]. To assess the performance of each algorithm, we need to specify objectives relevant to our case study. As we are not aware of the "good" test cases beforehand, we cannot use objectives based on knowing the optimal solution, such as the success rate or deviation from the optimal solution [9, 24, 77]. Since the simulation of the tested UAV in a concrete scenario exceeds the execution time of the algorithms significantly, we also do not focus on the execution time of the algorithms, as mentioned in [9, 24, 85, 125]. Instead, we concentrate on the best and average fitness values found by the algorithms, as presented in [60, 85, 101, 125, 130], and the deviation from the averaged values, as suggested by [85, 101, 125, 130].

In this work, we evaluate the performance of optimization algorithms and their combinations. We focus on so-called *collaborative* combinations, as introduced by [74, 98], that implement an information-sharing process between the individual executions of the different algorithms. Related work uses collaborative combinations for various use cases, such as solving distinct parts of the search problem [85], executing pre-experiments to decide which algorithm to utilize for further evaluations [130], or exploiting all their advantages [24, 33, 77], such as good exploration or exploitation characteristics. In this work, we explore the third use case by comparing the performance of collaborative combinations with their base algorithm to understand whether the combined version can create "better" test cases. To share information between the execution of the different optimization algorithms, we follow the guidance of [24, 33, 77] and pass the best candidates from the previous algorithms on to the subsequent algorithm. Further, we enhance the initial candidates for the subsequent algorithm with candidates from areas of the search space that were not yet heavily covered to increase the variability of the candidates. Such an increased variability leads to better results according to [139]. We can add these additional candidates as [4, 104] indicate that having seeded candidates that previously performed well is more important than the number of these candidates.

Summary: Related work on evaluating optimization algorithms presents their results for other domains and focuses on the convergence rate of the algorithms instead of finding their optimal solution. In addition, no evaluation of collaborative combinations for testing the safe behavior of UAVs exists yet.

8.5. Tools and Frameworks for Testing UAVs

The literature presents various tools and frameworks for testing autonomous systems such as ADS, UAVs, or cyber-physical systems [30, 65, 66, 71, 76, 141, 142]. The authors of [142] concentrate on minimizing test suites concerning uncertainty and cost-effectiveness and testing the reliability of cyber-physical systems while considering their uncertainty. Similarly, the authors of [66, 141] focus on testing the reliability of UAVs. Contrary to their works, we present a tool for testing the safe behavior of UAVs, which presents another aspect of testing autonomous systems.

Related work in the automotive domain [65, 76] applies fuzzing techniques to discover unsafe behaviors. However, as autonomous cars present different systems with other challenges than UAVs, we cannot utilize their tools directly.

[71] applies a model-based approach that uses metamorphic testing to assess the safe behavior of a controller for UAVs. This approach presents one alternative for testing the safe behavior of UAVs, which is based on the correct and comprehensive specification of all relevant metamorphic relations. However, deriving such a complete set of metamorphic relations presents various challenges. In this work, we instead concentrate on testing the safe behavior of UAVs in logical scenarios for which we aim to find worst-case situations. Even though we also face different challenges when deriving a complete list of logical scenarios, we find it more intuitive as test engineers define these logical scenarios implicitly when testing their systems.

The authors of [30] use fault-injection techniques to test the safe behavior of their simulated UAV. One disadvantage of these techniques is that we need to know the faults beforehand and can only detect previously specified faults. Since we want to find potential faults in our SUT even if we do not know them yet, we use search-based techniques to find worst-case situations for the SUT in relevant logical scenarios.

Summary: Related work about tools and frameworks for testing autonomous systems focuses on testing autonomous cars, testing other aspects than the UAVs' safe behavior, applying metamorphic testing, which presents the challenge of defining all relevant metamorphic relations, or using fault-injection techniques which can only detect known faults in the SUT.

9. Conclusion and Outlook

This chapter concludes by summarizing the concepts and results for testing the safe behavior of UAVs presented in this thesis. In addition, it includes limitations of our work, lessons learned, and ideas for future work.

9.1. Summary of Results and Limitations

In this work, we present a methodology for testing the safe behavior of UAVs with scenario-based testing. We focus on the two problems of (1) deriving relevant logical scenarios for a SUT and (2) generating test cases that can reveal potential faults in the SUT.

Addressing Problem 1: As related work concentrates on the mission of the UAV or contains a limited number of environmental effects, there is a need for an in-depth description of the relevant dimensions of logical scenarios for UAVs. In addition, different situations are challenging for ADS and UAVs due to, e.g., the influence of the environment and having a road versus an open field to operate. Further, there are no fine-grained traffic rules and rigid road structures for UAVs that ease the definition of logical scenarios. Thus, we cannot directly apply logical scenarios from the automotive domain and instead need to build an ontology with concrete dimensions that characterizes relevant logical scenarios for UAVs. We can derive the dimensions for an ontology (1) automatically by clustering real-flight data that we previously collected or (2) manually from mental models presented in expert knowledge, specifications, and the literature. Even though we currently lack high amounts of flight data needed for the first approach, we show the general concept of using these clustering techniques to acquire logical scenarios with simulated data in this work. In these experiments, we discover the following open research challenges for applying these techniques: (1) we currently lack the necessary high amounts of diverse and relevant real-world flight data, (2) as we currently lack a measure for assessing the quality of the generated clustering, we either need to specify one or inspect the results for all suitable settings, (3) as the characteristics of the gathered data heavily influence the clustering results, we need to consciously acquire this data for our use case, and (4) if we lack information about the UAV's environment, an explicit description of the derived logical scenarios gets more complicated. Due to these challenges, we focus on deriving logical scenarios based on mental models in the remainder of this work. Therefore, we present a methodology for systematically acquiring dimensions for an ontology based on mental models and show a

resulting ontology that characterizes logical scenarios for a quadcopter as one kind of UAV. Note that we consider the presented ontology a "living model" that we can adapt to newly discovered challenging situations or applications. In addition, we outline the necessary conditions for acquiring a complete list of logical scenarios from the presented ontology: (1) the created ontology needs to be complete, and (2) the defect hypothesis we apply for selecting specific logical situations from the ontology needs to represent all challenging situations for the SUT.

When creating the ontology, we need to find reasonable lower and upper bounds for each dimension before dividing it into system-specific categories. We can acquire these bounds from (1) expert knowledge and specifications or (2) experimental results for different bounds. In this work, we present an automated approach for the second alternative and demonstrate its applicability with the example of finding a maximal number of obstacles that we need to consider in logical scenarios for the SUT. Our experimental results for the optimization algorithm MOEA/D show that we need to include a maximal number of 5 or 8 relevant obstacles for the SUT, depending on the applied defect hypothesis used for data collection. In addition, the experiments reveal the different performances of the two investigated optimization algorithms MOEA/D and NSGAII when searching for extreme parameter values and the corresponding need to investigate system-specifically which works best for the presented methodology. With the resulting bounds for the ontology's dimensions, we can effectively limit the number of logical scenarios in which we need to test the SUT. Thus, these bounds further present a basis for collecting a comprehensive list of relevant logical scenarios for UAVs in future work.

Limitations: In this work, we present an ontology that characterizes logical scenarios for a quadcopter as one kind of UAV that might not generalize to other types of UAVs. Note the need for deriving the categories of each dimension of the ontology system-specifically as different situations are challenging for various types of UAVs. Even though we base the dimensions of this ontology on expert knowledge, the literature, and specifications, there might arise additional relevant dimensions in the future. Thus, we consider the ontology a "living model" that we can easily adapt. Concerning the experiments for deriving reasonable bounds for the ontology's dimensions, we focus on deriving them for the open-source PX4 autopilot and do not evaluate other SUTs. Further note that it is essential to base the selection of the parameter values to collect during the experiments on a correct defect hypothesis about challenging situations for the SUT to gain insightful results. Otherwise, we cannot infer that the derived maximum number of relevant obstacles is suitable when we generate "good" test cases for the SUT in the next step Finally, we concentrate on static obstacles in our experiments for simplicity of presentation but present guidelines for applying our approach to dynamic ones.

Addressing Problem 2: As for the definition of logical scenarios, we cannot directly use the methodology found in the literature about testing the safe behavior of ADS for testing UAVs. Further, since there are no fine-grained traffic rules for UAVs, the challenge of clearly defining a safe behavior for UAVs increases. Related work on testing the safe behavior of

UAVs focuses on testing against a specified safety distance and in an environment with no obstacles and environmental effects. Thus, there is a need for a methodology for testing the safe behavior of UAVs that considers the environment and acknowledges the challenge of clearly defining a safe behavior for UAVs. In this work, we present such a methodology that includes in-depth descriptions of the environment in logical scenarios and generates so-called "good" test cases that represent worst-case situations for the SUT with the help of optimization algorithms. Further, we demonstrate how we can apply this methodology when we (1) have a specified safety distance or (2) lack such a safety distance due to missing regulations or specifications. In our experiments, we detect 4 to 62 safety distance violations when generating test cases for the open-source PX4 autopilot [81] for four logical scenarios that denote all alternatives to avoid obstacles and a specified safety distance. In addition, we discover questionable behaviors of the SUT with our proposed methodology when no safety distance is defined. These experimental results show the effectiveness and applicability of our proposed approach for generating worst-case situations that reveal potential faults in the SUT. Even though experts still need to inspect the behavior of the UAV manually when we cannot explicitly specify the safe behavior of the UAV, we minimize the experts' efforts by presenting worst-case situations that they solely need to investigate.

As current literature [48, 147] suggests the use of heuristic optimization algorithms for generating these worst-case situations, we cannot ensure that we find an optimal solution. Thus, to gain confidence in the quality of the discovered worst-case situations, we need to perform a system-specific evaluation of various optimization algorithms. Further, as the combination of these optimization algorithms might result in "better" test cases, we also investigate these combinations. Related work in this area presents such empirical assessments in others domains or focuses on the convergence rate of the evaluated algorithms. To the best of our knowledge, no work on evaluating optimization algorithms and their combinations for scenario-based testing for UAVs exists. In the case study presented in this work, we assess the performance of the three optimization algorithms NSGAII, PSO, and BO and their sequential combinations when testing an open-source UAV in five different logical scenarios. The results of our case study indicate that NSGAII performs significantly better than BO and that we should use the combined algorithms instead of PSO and BO directly as they, on average, produce "better" test cases. Finally, the results show that PNP presents, on average, the best performance overall runs and logical scenarios with an increased performance of 20% compared to NSGAII, considering the quality of the generated test cases. In addition, several of the evaluated algorithms even create test cases that reveal the unsafe behavior of the SUT in the tested logical scenarios. However, our case study also reveals the high variation of quality of the generated test cases for different logical scenarios for all optimization algorithms, including PNP. Even if we perform multiple runs per logical scenario, we might still miss worst-case situations even with the best-performing optimization algorithm. Having such a missing guarantee of finding the worst-case situation is problematic as we cannot ensure the safe behavior of the SUT in the tested logical scenarios, which is needed for the certification of autonomous systems. Even though we know that heuristic search algorithms do not provide optimality guarantees,

our results show that non-optimality does not seem to be an exceptional case. For these reasons, scenario-based testing comes at the *extra cost of having to run multiple optimizers and their combinations* to increase the chance of detecting worst-case situations. We feel that this insight and its consequences are somewhat lost in the scenario-based testing community and that they challenge the current widespread application of scenario-based testing for ensuring the safe behavior of autonomous systems such as ADS or UAVs.

To systematically apply the presented methodology, we introduce the tool StellaUAV that enables (1) the definition of logical scenarios to test the UAV's safe behavior, (2) the generation of worst-case situations for each of these logical scenarios, and (3) the comparison of the performance of various optimization algorithms for the presented use case. We evaluate the applicability and effectiveness of StellaUAV by searching for safety distance violations in 32 logical scenarios. The results show that the tested PX4 autopilot can handle static obstacles very well, whereas it encounters problems with dynamically moving ones. Thus, these results show the effectiveness of StellaUAV when creating "good" test cases that can reveal potential faults in the SUT.

Limitations: In our experiments for generating "good" test cases with search-based techniques, we focus on testing the safe behavior of the PX4 autopilot. Even though we execute only a limited number of evaluations per logical scenario, we detect unsafe behaviors of the SUT with them, which shows the effectiveness of the proposed approach. When evaluating the performance of various optimization algorithms, we focus on five logical scenarios and do not claim that the best-performing optimization algorithm found in our experiments also operates best for other logical scenarios and SUTs. Instead, we emphasize the need to perform a similar case study as the presented one for relevant logical scenarios for each SUT specifically. In this empirical assessment, we execute fifteen runs for each logical scenario to acknowledge the randomness of the evaluated optimization algorithms and gain robust evaluation results. However, we do not perform 30 runs as suggested by [5] due to limited computation power and time, as discussed in Section 6.3.1. Finally, our experimental results for StellaUAV focus on showing the applicability of the presented tool for the integrated PX4 autopilot and the tested logical scenarios. Even though we evaluate the tool's effectiveness in 32 logical scenarios that represent various environmental conditions, we do not claim to present a complete list of relevant logical scenarios for the SUT with these logical scenarios.

9.2. Lessons Learned

While working on this thesis, we gained several insights into the topic of testing the safe behavior of UAVs with scenario-based testing, from which we share the most significant ones in this section.

We need to derive logical scenarios for testing the safe behavior of UAVs system-specifically. As many different kinds of UAVs exist that operate in various environments,

diverse situations are relevant and challenging for these systems. In particular, these differences are significant between UAVs flying at low or high altitudes. Thus, to enable thorough testing of the safe behavior of a SUT, we need to derive logical scenarios that present challenging situations for the SUT and cannot use acquired logical scenarios for all UAV systems.

Collecting a complete list of logical scenarios presents various open challenges. We can acquire logical scenarios based on (1) collected real-world flight data, or (2) mental models represented in, e.g., an ontology. To acquire a complete list of logical scenarios based on (1), we rely on high amounts of diverse and relevant real-world data for testing the safe behavior of the SUT. This data does currently not exist for UAVs. Further, we present various open research challenges for applying this approach to derive logical scenarios for UAVs in this work. We can gather a complete list of logical scenarios based on (2) if we have sound and comprehensive defect hypotheses about challenging situations for the SUT. Such defect hypotheses are, however, not easily specified.

Explicitly defining the safe behavior of UAVs is not easily accomplished due to missing regulations. Implicitly, we can describe the safe behavior of UAVs by stating that they should not harm anybody or anything while operating and should pose no unreasonable risk to their environment. However, defining a concrete measure for this implicit description in all situations is challenging since we lack strict regulations for UAVs. Thus, a methodology for testing the safe behavior of UAVs needs to take this into account and enable us to generate test cases with and without an explicit specification of such a safe behavior.

We need to execute multiple optimization algorithms to reliably find worst-case situations for UAVs. Our case study reveals that even the best-performing optimization algorithm from a system-specific assessment of various optimization algorithms might miss several worst-case situations. These results strongly indicate the need to execute multiple optimization algorithms to reliably detect worst-case situations for the SUT. In addition, we encourage test engineers to perform multiple runs of generating "good" test cases for each logical scenario and optimization algorithm to reduce the randomness of the results.

Scenario-based testing increases the confidence in the safe behavior of UAVs. Since we search for challenging situations for the SUT in relevant logical scenarios, scenario-based testing provides valuable insights into the safe behavior of UAVs. Further, we can increase the confidence in the safe behavior of the SUT by performing various runs per logical scenarios and executing different optimization algorithms for detecting worst-case situations for our SUT. Even though scenario-based testing probably won't be able to provide a guarantee of the safe behavior of UAVs, it presents, in our opinion, one of our best efforts to produce high confidence in their safe behavior.

9.3. Future Work

This thesis presents solutions to various problems and gaps when testing the safe behavior of UAVs. In future work, we would like to address additional open challenges for the proposed methodology.

Investigation of open research challenges for automatically deriving logical scenarios. In this work, we outlined various open challenges of applying clustering techniques to collected real-flight data to automatically derive logical scenarios for UAVs. In future work, we would like to work on solutions for these challenges and enable utilization of the presented approach for deriving logical scenarios for UAVs that can complement manually derived ones.

Evaluation of the correctness of various defect hypotheses for selecting specific logical scenarios. We need to apply a correct defect hypothesis about challenging situations for the SUT to acquire relevant logical scenarios from the presented ontology in this work. For simplicity of presentation, we show only simple defect hypotheses such as pair-wise testing or the impact of regional weather effects. In the future, we aim to investigate the correctness of more sophisticated defect hypotheses to enable the derivation of a comprehensive list of logical scenarios for UAVs.

Extension of the case study about the performance of various optimization algorithms. In our case study, we evaluate the performance of three optimization algorithms and their combinations for generating "good" test cases for testing the safe behavior of the PX4 autopilot for five different logical scenarios in fifteen runs. In future work, we would like to extend this case study by evaluating additional optimization algorithms and testing the safe behavior of other UAV autopilots to strengthen the significance and generalizability of the presented results. Further, we aim to perform additional runs per logical scenario to more reliably explore the statistical significance of our computed results.

Extension of StellaUAV to test various kinds of UAV autopilots and present more refined real-world situations. Our tool StellaUAV tests the PX4 autopilot in the current implementation. In future work, we aim to include other autopilots into StellaUAV to test their safe behavior and an easy selection procedure for choosing them. Further, the tool uses the open-source simulator Gazebo to simulate the UAV and its environment. With this simulator, we can adapt the landform, water level, kinds, forms, and sizes of obstacles, the wind force, and the fogginess. However, we cannot represent more refined real-world situations with, e.g., different heaviness of rain or varying ambient temperature. Thus, we would like to extend the current setup in StellaUAV to add additional environmental effects that we encounter in the real world. Further, we aim to extend the logical scenarios and the methodology for generating "good" test cases for these logical scenarios by including parameters that represent functions instead of constant values, e.g., to represent wind gusts.

Testing the safe behavior of cooperative UAVs. In this work, we focus on testing the safe behavior of a single UAV of autonomy level 4 following the terminology presented in Section 2.1, where the UAV receives a mission to perform completely autonomously without human interaction. In future work, we aim to extend our presented methodology to test the safe behavior of autonomously behaving cooperative UAVs, which denote an autonomy level of 5. For cooperatively operating UAVs, we need to adapt the search space and the fitness function to acknowledge the additional challenges of the cooperation. Note that the presented ontology in this work already includes the possibility to specify logical scenarios with cooperative UAVs that use various types of cooperation strategies.

Bibliography

[1] Shaukat Ali, Lionel C Briand, Hadi Hemmati, and Rajwinder Kaur Panesar-Walawege. A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Transactions on Software Engineering*, 36(6):742–762, 2009.

[2] Mostafa Aliyari, Behrooz Ashrafi, and Yonas Zewdu Ayele. Hazards identification and risk assessment for uav–assisted bridge inspections. *Structure and Infrastructure Engineering*, pages 1–17, 2020.

[3] American Meterological Society. Rain. Available online: `https://glossary.ametsoc.org/wiki/Rain`. Accessed December 20, 2022.

[4] C Aranha and Hitoshi Iba. Modelling cost into a genetic algorithm-based portfolio optimization system by seeding and objective sharing. In *2007 IEEE Congress on Evolutionary Computation*, pages 196–203. IEEE, 2007.

[5] Andrea Arcuri and Lionel Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 1–10. IEEE, 2011.

[6] Aitor Arrieta et al. Search-based test case generation for cyber-physical systems. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 688–697. IEEE, 2017.

[7] ASAM. Asam openscenario. Available online: `https://www.asam.net/standards/detail/openscenario/`. Accessed December 20, 2022.

[8] Civil Aviation Authority. The air navigation order 2016 and regulations (cap393), 2016, 2016.

[9] Mustafa Ayyıldız and Kerim Çetinkaya. Comparison of four different heuristic optimization algorithms for the inverse kinematics solution of a real 4-dof serial robot manipulator. *Neural Computing and Applications*, 27(4):825–836, 2016.

[10] Johannes Bach, Stefan Otten, and Eric Sax. Model based scenario specification for development and test of automated driving functions. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 1149–1155. IEEE, 2016.

[11] Gerrit Bagschik, Till Menzel, and Markus Maurer. Ontology based scene creation for the development of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1813–1820. IEEE, 2018.

[12] Earl T Barr et al. The oracle problem in software testing: A survey. *Transactions on Software Engineering*, 41(5):507–525, 2014.

[13] Vahid Beiranvand, Warren Hare, and Yves Lucet. Best practices for comparing optimization algorithms. *Optimization and Engineering*, 18(4):815–848, 2017.

[14] Christine M Belcastro, David H Klyde, Michael J Logan, Richard L Newman, and John V Foster. Experimental flight testing for assessing the safety of unmanned aircraft system safety-critical operations. In *17th AIAA Aviation Technology, Integration, and Operations Conference*, page 3274, 2017.

[15] Antonio Benitez-Hidalgo et al. jmetalpy: a python framework for multi-objective optimization with metaheuristics. *Swarm and Evolutionary Computation*, 51:100598, 2019.

[16] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD Workshop*, volume 10, pages 359–370. Seattle, WA, USA:, 1994.

[17] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

[18] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.

[19] Alessandro Calò, Paolo Arcaini, Shaukat Ali, Florian Hauer, and Fuyuki Ishikawa. Generating avoidable collision scenarios for testing autonomous driving systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 375–386. IEEE, 2020.

[20] Y Uny Cao, Alex S Fukunaga, and Andrew Kahng. Cooperative mobile robotics: antecedents and directions. *Autonomous robots*, 4(1):7–27, 1997.

[21] CASA. Civil aviation safety regulations part 101, 2016.

[22] Danilo Cavaliere, Vincenzo Loia, and Sabrina Senatore. Towards an ontology design pattern for uav video content analysis. *IEEE Access*, 7:105342–105353, 2019.

[23] JD Cem Kaner. An introduction to scenario testing. *Florida Institute of Technology, Melbourne*, pages 1–13, 2013.

[24] Rachid Chelouah and Patrick Siarry. Genetic and nelder–mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions. *European Journal of Operational Research*, 148(2):335–348, 2003.

[25] Mo Chen, Qie Hu, Casey Mackin, Jaime F Fisac, and Claire J Tomlin. Safe platooning of unmanned aerial vehicles via reachability. In *2015 54th IEEE conference on decision and control (CDC)*, pages 4695–4701. IEEE, 2015.

[26] Tao Chen, Miqing Li, and Xin Yao. On the effects of seeding strategies: a case for search-based multi-objective service composition. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1419–1426, 2018.

[27] Reece A Clothier and Rodney A Walker. Determination and evaluation of uav safety objectives. In *Proceedings of the 21st International Unmanned Air Vehicle Systems Conference*, 2006.

[28] Bruce T Clough. Metrics, schmetrics! how the heck do you determine a uav's autonomy anyway. Technical report, Air Force Research Lab Wright-Patterson AFB OH, 2002.

[29] Pascal Côté and Robert Leconte. Comparison of stochastic optimization algorithms for hydropower reservoir operation with ensemble streamflow prediction. *Journal of Water Resources Planning and Management*, 142(2), 2016.

[30] Xunhua Dai, Chenxu Ke, Quan Quan, and Kai-Yuan Cai. Rflysim: Automatic test platform for uav autopilot systems with fpga-based hardware-in-the-loop simulations. *Aerospace Science and Technology*, 114:106727, 2021.

[31] Erwin de Gelder, Jan-Pieter Paardekooper, Olaf Op den Camp, and Bart De Schutter. Safety assessment of automated vehicles: how to determine whether we have collected enough field data? *Traffic injury prevention*, 20(sup1):S162–S170, 2019.

[32] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[33] Xavier Delorme, Xavier Gandibleux, and Fabien Degoutin. Evolutionary, constructive and hybrid procedures for the bi-objective set packing problem. *European Journal of Operational Research*, 204(2):206–217, 2010.

[34] Patrick Doherty and Piotr Rudol. A uav search and rescue scenario with human body detection and geolocalization. In *Australasian Joint Conference on Artificial Intelligence*, pages 1–13. Springer, 2007.

[35] Les Dorr and A Duquette. Fact sheet–small unmanned aircraft regulations (part 107). *Federal Aviation Administration*, 2016.

[36] Christopher D Drummond, Mitchell D Harley, Ian L Turner, A Nashwan A Matheen, William C Glamore, et al. Uav applications to coastal engineering. In *Australasian Coasts & Ports Conference 2015: 22nd Australasian Coastal and Ocean Engineering Conference and the 15th Australasian Port and Harbour Conference*, page 267. Engineers Australia and IPENZ, 2015.

[37] EASA. Easy access rules for unmanned aircraft systems. Available online: `https://www.easa.europa.eu/document-library/easy-access-rules/easy-access-rules-unmanned-aircraft-systems-regulation-eu`. Accessed December 20, 2022.

[38] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[39] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[40] Alessandro Farinelli, Luca Iocchi, and Daniele Nardi. Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2015–2028, 2004.

[41] Gordon Fraser and Andrea Arcuri. The seed is strong: Seeding strategies in search-based software testing. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 121–130. IEEE, 2012.

[42] Peter Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[43] Jeremy H Gillula, Haomiao Huang, Michael P Vitus, and Claire J Tomlin. Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice. In *2010 IEEE International Conference on Robotics and Automation*, pages 1649–1654. IEEE, 2010.

[44] Government of Canada. Beaufort wind scale table. Available online: `https://www.canada.ca/en/environment-climate-change/services/general-marine-weather-information/understanding-forecasts/beaufort-wind-scale-table.html`. Accessed December 20, 2022.

[45] PA Grudniewski and AJ Sobey. Benchmarking the performance of genetic algorithms on constrained dynamic problems. *Natural Computing*, pages 1–17, 2020.

[46] Magnus Gyllenhammar, Carl Zandén, and Martin Törngren. Defining fundamental vehicle actions for the development of automated driving systems. In *SAE Technical Paper Series*. SAE International, 2020.

[47] Florian Hauer, Ilias Gerostathopoulos, Tabea Schmidt, and Alexander Pretschner. Clustering traffic scenarios using mental models as little as possible. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1007–1012. IEEE, 2020.

[48] Florian Hauer, Alexander Pretschner, and Bernd Holzmüller. Fitness functions for testing automated and autonomous driving systems. In *International Conference on Computer Safety, Reliability, and Security*, pages 69–84. Springer, 2019.

[49] Florian Hauer, Tabea Schmidt, Bernd Holzmüller, and Alexander Pretschner. Did we test all scenarios for automated and autonomous driving systems? In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2950–2955. IEEE, 2019.

[50] Philipp Helle, Wladimir Schamai, and Carsten Strobel. Testing of autonomous systems–challenges and current state-of-the-art. In *INCOSE international symposium*, volume 26, pages 571–584. Wiley Online Library, 2016.

[51] Xuan Hu and Jie Liu. Ontology construction and evaluation of uav fcms software requirement elicitation considering geographic environment factors. *IEEE Access*, 8:106165–106182, 2020.

[52] Hui-Min Huang, Kerry Pavek, Brian Novak, James Albus, and E Messin. A framework for autonomy levels for unmanned systems (alfus). *Proceedings of the AUVSI's unmanned systems North America*, pages 849–863, 2005.

[53] Shafagh Jafer, Bharvi Chhaya, Umut Durak, and Torsten Gerlach. Formal scenario definition language for aviation: aircraft landing case study. In *AIAA Modeling and Simulation Technologies Conference*, page 3521, 2016.

[54] Stefan Jesenski, Jan Erik Stellet, Wolfgang Branz, and J Marius Zöllner. Simulation-based methods for validation of automated driving: A model-based analysis and an overview about methods for implementation. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 1914–1921. IEEE, 2019.

[55] Eric N Johnson, Michael A Turbe, Allen D Wu, Suresh K Kannan, and James C Neidhoefer. Flight test results of autonomous fixed-wing uav transitions to and from stationary hover. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference Exhibit, Monterey, CO*, 2006.

[56] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[57] Tamás Keviczky and Gary J Balas. Flight test of a receding horizon controller for autonomous uav guidance. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 3518–3523. IEEE, 2005.

[58] Jack PC Kleijnen. Verification and validation of simulation models. *European journal of operational research*, 82(1):145–162, 1995.

[59] Florian Klück, Yihao Li, Mihai Nica, Jianbo Tao, and Franz Wotawa. Using ontologies for test suites generation for automated and autonomous driving functions. In *2018 IEEE International symposium on software reliability engineering workshops (ISSREW)*, pages 118–123. IEEE, 2018.

[60] Florian Klück, Martin Zimmermann, Franz Wotawa, and Mihai Nica. Performance comparison of two search-based testing strategies for adas system validation. In *IFIP International Conference on Testing Software and Systems*, pages 140–156. Springer, 2019.

[61] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154. IEEE, 2004.

[62] Nicola Kolb, Claudius Jordan, Florian Huber, and Alexander Pretschner. Automatic evaluation of automatically derived semantic scenario instance descriptions. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1565–1571. IEEE, 2022.

[63] Jimmy Krozel, William McNichols, Joseph Prete, and Tenny Lindholm. Causality analysis for aviation weather hazards. In *The 26th Congress of ICAS and 8th AIAA ATIO*, page 8914, 2008.

[64] Nurul Abdul Latiff, Charalampos Tsimenidis, and Bayan Sharif. Performance comparison of optimization algorithms for clustering in wireless sensor networks. In *2007 IEEE International Conference on Mobile Adhoc and Sensor Systems*, pages 1–4. IEEE, 2007.

[65] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. Av-fuzzer: Finding safety violations in autonomous driving systems. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 25–36. IEEE, 2020.

[66] Rui Li, Huai Liu, Guannan Lou, Xi Zheng, Xiao Liu, and Tsong Yueh Chen. Metamorphic testing on multi-module uav systems. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1171–1173. IEEE, 2021.

[67] Xin Li, Sonia Bilbao, Tamara Martín-Wanton, Joaquim Bastos, and Jonathan Rodriguez. Swarms ontology: a common information model for the cooperation of underwater robots. *Sensors*, 17(3):569, 2017.

[68] Yihao Li, Jianbo Tao, and Franz Wotawa. Ontology-based test generation for automated and autonomous driving functions. *Information and Software Technology*, 117:106200, 2020.

[69] T Warren Liao. Clustering of time series data—a survey. *Pattern recognition*, 38(11):1857–1874, 2005.

[70] Yucong Lin and Srikanth Saripalli. Collision avoidance for uavs using reachable sets. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 226–235. IEEE, 2015.

[71] Mikael Lindvall, Adam Porter, Gudjon Magnusson, and Christoph Schulze. Metamorphic model-based testing of autonomous systems. In *2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET)*, pages 35–41. IEEE, 2017.

[72] Giuseppe Loianno, Vojtech Spurny, Justin Thomas, Tomas Baca, Dinesh Thakur, Daniel Hert, Robert Penicka, Tomas Krajnik, Alex Zhou, Adam Cho, et al. Localization, grasping, and transportation of magnetic objects by a team of mavs in challenging desert-like environments. *IEEE Robotics and Automation Letters*, 3(3):1576–1583, 2018.

[73] Mark David Lower. Unique aspects of unmanned aerial vehicle testing, 2004.

[74] Manuel Lozano and Carlos García-Martínez. Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report. *Computers & Operations Research*, 37(3):481–497, 2010.

[75] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[76] Rupak Majumdar, Aman Mathur, Marcus Pirron, Laura Stegner, and Damien Zufferey. Paracosm: A test framework for autonomous driving simulations. In *International Conference on Fundamental Approaches to Software Engineering*, pages 172–195. Springer, Cham, 2021.

[77] Hasan Makas and NEJAT YUMUŞAK. Balancing exploration and exploitation by using sequential execution cooperation between artificial bee colony and migrating birds optimization algorithms. *Turkish Journal of Electrical Engineering & Computer Sciences*, 24(6):4935–4956, 2016.

[78] Adriano Mancini, Andrea Cesetti, A Iuale, Emanuele Frontoni, Primo Zingaretti, and Sauro Longhi. A framework for simulation and testing of uavs in cooperative scenarios. In *Unmanned Aircraft Systems*, pages 307–329. Springer, 2008.

[79] David M Mark and Barry Smith. A science of topography: from qualitative ontology to digital representations. *Geographic Information Science and Mountain Geomorphology*, pages 75–100, 2004.

[80] T McGeer, L Newcome, and Juris Vagners. Quantitative risk management as a regulatory approach to civil uavs. In *Proceedings of the International Workshop on UAV Certification*, 1999.

[81] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. Px4: a node-based multithreaded open source robotics framework for deeply embedded platforms. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6235–6240. IEEE, 2015.

[82] Till Menzel, Gerrit Bagschik, and Markus Maurer. Scenarios for development, test and validation of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1821–1827. IEEE, 2018.

[83] Francesco Montanari, Christoph Stadler, Jörg Sichermann, Reinhard German, and Anatoli Djanatliev. Maneuver-based resimulation of driving scenarios based on real driving data. In *2021 IEEE Intelligent Vehicles Symposium (IV)*, pages 1124–1131. IEEE, 2021.

[84] Andrew Moore, Swee Balachandran, Steven D Young, Evan T Dill, Michael J Logan, Louis J Glaab, Cesar Munoz, and Maria Consiglio. Testing enabling technologies for safe uas urban operations. In *2018 Aviation Technology, Integration, and Operations Conference*, page 3200, 2018.

[85] Mohammad Hasan Moradi and M Abedini. A combination of genetic algorithm and particle swarm optimization for optimal dg location and sizing in distribution systems. *International Journal of Electrical Power & Energy Systems*, 34(1):66–74, 2012.

[86] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[87] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[88] Prasanth Nair, A Keane, and R Shimpi. Combining approximation concepts with genetic algorithm-based structural optimization procedures. In *39th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 1998.

[89] Demin Nalic, Tomislav Mihalj, Maximilian Bäumler, Matthias Lehmann, Arno Eichberger, and Stefan Bernsteiner. Scenario based testing of automated driving systems: A literature survey. In *FISITA web Congress*, 2020.

[90] Takuya Nanri, Fang Fang, and Abdelaziz Khiat. Use-case generation and analysis for autonomous driving in urban areas. *International Journal of Automotive Engineering*, 12(2):54–61, 2021.

[91] Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python. Available online: `https://github.com/fmfn/BayesianOptimization`, 2014. Accessed December 20, 2022.

[92] Julian Oes. Mavsdk-python. Available online: `https://github.com/mavlink/MAVSDK-Python`, 2019. Accessed December 20, 2022.

[93] Isaac Olson and Ella M Atkins. Qualitative failure analysis for a small quadrotor unmanned aircraft system. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, page 4761, 2013.

[94] Lynne E Parker, Daniela Rus, and Gaurav S Sukhatme. Multiple mobile robot systems. In *Springer Handbook of Robotics*, pages 1335–1384. Springer, 2016.

[95] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[96] Martin Pelikan, David E Goldberg, Erick Cantú-Paz, et al. Boa: The bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume 1, pages 525–532. Citeseer, 1999.

[97] Alexander Pretschner. Defect-based testing. *Dependable Software Systems Engineering*, 84, 2015.

[98] Jakob Puchinger and Günther Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *International Work-Conference on the Interplay Between Natural and Artificial Computation*, pages 41–53. Springer, 2005.

[99] Rodrigo Queiroz, Thorsten Berger, and Krzysztof Czarnecki. Geoscenario: an open dsl for autonomous driving scenario representation. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 287–294. IEEE, 2019.

[100] Morgan Quigley et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[101] Ronald Rardin and Reha Uzsoy. Experimental evaluation of heuristic optimization algorithms: A tutorial. *Journal of Heuristics*, 7(3):261–304, 2001.

[102] Waseem Rawat and Zenghui Wang. Hybrid stochastic ga-bayesian search for deep convolutional neural network model selection. *JUCS-Journal of Universal Computer Science*, 25:647, 2019.

[103] Lennart Ries, Philipp Rigoll, Thilo Braun, Thomas Schulik, Johannes Daube, and Eric Sax. Trajectory-based clustering of real-world urban driving sequences with multiple traffic objects. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 1251–1258. IEEE, 2021.

[104] José Miguel Rojas, Gordon Fraser, and Andrea Arcuri. Seeding strategies in search-based unit test generation. *Software Testing, Verification and Reliability*, 26(5):366–401, 2016.

[105] Jonathan W Rosen. Zipline's ambitious medical drone delivery in africa. *MIT Technology Review, June*, 8:2017, 2017.

[106] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[107] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.

[108] Robert G Sargent. Verification and validation of simulation models. In *Proceedings of the 2010 Winter Simulation Conference*, pages 166–183. IEEE, 2010.

[109] Robert G Sargent and Osman Balci. History of verification and validation of simulation models. In *2017 Winter Simulation Conference (WSC)*, pages 292–307. IEEE, 2017.

[110] Mrinmoy Sarkar et al. Pie: a tool for data-driven autonomous uav flight testing. *Journal of Intelligent & Robotic Systems*, 98(2):421–438, 2020.

[111] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st international conference on distributed computing systems workshops*, pages 166–171. IEEE, 2011.

[112] Tabea Schmidt. Supplementary material for 'Testing the Safe Behavior of Unmanned Aerial Vehicles with Scenario-Based Testing'. Available online: `https://figshare.com/s/1fe3f53e42dd13dd21da`, 2022. Accessed December 20, 2022.

[113] Tabea Schmidt, Florian Hauer, and Alexander Pretschner. Automated anomaly detection in cps log files. In *International Conference on Computer Safety, Reliability, and Security*, pages 179–194. Springer, 2020.

[114] Tabea Schmidt, Florian Hauer, and Alexander Pretschner. Understanding safety for unmanned aerial vehicles in urban environments. In *2021 IEEE Intelligent Vehicles Symposium (IV)*, pages 638–643. IEEE, 2021.

[115] Tabea Schmidt, Florian Hauer, and Alexander Pretschner. Exploring a maximal number of relevant obstacles for testing uavs. In *International Conference on Computer Safety, Reliability, and Security*, pages 335–349. Springer, 2022.

[116] Tabea Schmidt and Alexander Pretschner. StellaUAV: A tool for testing the safe behavior of uavs with scenario-based testing. In *IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, pages 37–48. IEEE, 2022.

[117] Tabea Schmidt and Alexander Pretschner. Supplementary material for 'StellaUAV: A Tool for Testing the Safe Behavior of UAVs with Scenario-Based Testing'. Available online: `https://doi.org/10.6084/m9.figshare.19666311`, 2022. Accessed December 20, 2022.

[118] Tabea Schmidt and Alexander Pretschner. Ontology-based collection of scenarios for testing uavs. In *IEEE Robotics and Automation Letters*. IEEE, 2022, under review.

[119] Ruwen Schnabel, Raoul Wessel, Roland Wahl, and Reinhard Klein. Shape recognition in 3d point-clouds. In *The 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*. Václav Skala-UNION Agency, 2008.

[120] Hazim Shakhatreh, Ahmad H Sawalmeh, Ala Al-Fuqaha, Zuochao Dou, Eyad Almaita, Issa Khalil, Noor Shamsiah Othman, Abdallah Khreishah, and Mohsen Guizani. Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. *Ieee Access*, 7:48572–48634, 2019.

[121] Frank D Shepard. *Reduced visibility due to fog on the highway*, volume 228. Transportation Research Board, 1996.

[122] Saeid Shokri, Mohammad Taghi Sadeghi, and Mahdi Ahmadi Marvast. High reliability estimation of product quality using support vector regression and hybrid metaheuristic algorithms. *Journal of the Taiwan Institute of Chemical Engineers*, 45(5):2225–2232, 2014.

[123] Shiva Ram Reddy Singireddy and Tugrul U Daim. Technology roadmap: drone delivery–amazon prime air. In *Infrastructure and Technology Management*, pages 387–412. Springer, 2018.

[124] Denis Smirnov and Peter Stutz. Use case driven approach for ontology-based modeling of reconnaissance resources on-board uavs using owl. In *2017 IEEE Aerospace Conference*, pages 1–17. IEEE, 2017.

[125] Mingjie Song and DongMei Chen. A comparison of three heuristic optimization algorithms for solving the multi-objective land allocation (mola) problem. *Annals of GIS*, 24(1):19–31, 2018.

[126] Jian Sun, He Zhang, Huajun Zhou, Rongjie Yu, and Ye Tian. Scenario-based test automation for highly automated vehicles: A review and paving the way for systematic safety assurance. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[127] Alexander Tenbrock, Alexander König, Thomas Keutgens, and Hendrik Weber. The conscend dataset: Concrete scenarios from the highd dataset according to alks regulation unece r157 in openx. In *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*, pages 174–181. IEEE, 2021.

[128] Amila Thibbotuwawa, Grzegorz Bocewicz, Banaszak Zbigniew, and Peter Nielsen. A solution approach for uav fleet mission planning in changing weather conditions. *Applied Sciences*, 9(19):3972, 2019.

[129] T. Tozer, D. Grace, J. Thompson, and P. Baynham. Uavs and haps - potential convergence for military communications. *IEE Colloquium on Military Satellite Communications (Ref. No. 2000/024)*, 2000.

[130] Fevrier Valdez, Patricia Melin, and Oscar Castillo. An improved evolutionary method with fuzzy logic for combining particle swarm optimization and genetic algorithms. *Applied Soft Computing*, 11(2):2625–2632, 2011.

[131] András Vargha and Harold D Delaney. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[132] Amedeo Rodi Vetrella, Giancarmine Fasano, Alfredo Renga, and Domenico Accardo. Cooperative uav navigation based on distributed multi-antenna gnss, vision, and mems sensors. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1128–1137. IEEE, 2015.

[133] Walther Wachenfeld and Hermann Winner. The release of autonomous vehicles. In *Autonomous driving*, pages 425–449. Springer, 2016.

[134] Kay Wackwitz and Hendrick Boedecker. Safety risk assessment for uav operation. *Drone Industry Insights, Safe Airspace Integration Project, Part One, Hamburg, Germany*, 2015.

[135] Wenshuo Wang, Chang Liu, and Ding Zhao. How much data are enough? a statistical approach with case study on longitudinal driving behavior. *IEEE Transactions on Intelligent Vehicles*, 2(2):85–98, 2017.

[136] Nico Weber, Christoph Thiem, and Ulrich Konigorski. Unscene: Toward unsupervised scenario extraction for automated driving systems from urban naturalistic road traffic data. *arXiv preprint arXiv:2202.06608*, 2022.

[137] Joachim Wegener and Oliver Bühler. Evaluation of different fitness functions for the evolutionary testing of an autonomous parking system. In *Genetic and Evolutionary Computation Conference*, pages 1400–1412. Springer, 2004.

[138] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10(12):399, 2013.

[139] Shin Yoo and Mark Harman. Test data regeneration: generating new test data from existing test data. *Software Testing, Verification and Reliability*, 22(3):171–201, 2012.

[140] Larry Young, Jeffrey Yetter, and Mark Guynn. System analysis applied to autonomy: Application to high-altitude long-endurance remotely operated aircraft. *IAA Infotech@Aerospace Conference*, 2005.

[141] Jiantao Zhang, Zheng Zheng, Beibei Yin, Kun Qiu, and Yang Liu. Testing graph searching based path planning algorithms by metamorphic testing. In *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 158–15809. IEEE, 2019.

[142] Man Zhang, Shaukat Ali, and Tao Yue. Uncertainty-wise test case generation and minimization for cyber-physical systems. *Journal of Systems and Software*, 153:1–21, 2019.

[143] Qingfu Zhang and Hui Li. Moea/d: a multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[144] Yuchen Zhou and John S Baras. Reachable set approach to collision avoidance for uavs. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 5947–5952. IEEE, 2015.

[145] Xueping Zhu, Zhengchun Liu, and Jun Yang. Model of collaborative uav swarm toward coordination and control mechanisms study. In *ICCS*, pages 493–502, 2015.

[146] Xueyi Zou, Rob Alexander, and John McDermid. Safety validation of sense and avoid algorithms using simulation and evolutionary search. In *International Conference on Computer Safety, Reliability, and Security*, pages 33–48. Springer, 2014.

[147] Xueyi Zou, Rob Alexander, and John McDermid. Testing method for multi-uav conflict resolution using agent-based simulation and multi-objective search. *Journal of Aerospace Information Systems*, 13(5):191–203, 2016.

Glossary

ADS Automated and Autonomous Driving Systems. 4–7, 18, 25, 26, 59–61, 73, 74, 86, 87, 102–105, 107, 108, 110

BAT Boundary Analysis Testing. 60, 61, 63–65, 67–70, 132, 136

BO Bayesian Optimization. 18, 20, 21, 75–77, 81, 83, 86, 90, 92, 109, 131, 136, 146

DTW Dynamic Time Warping. 27, 29, 145

JSON JavaScript Object Notation. 34, 40, 41, 43, 88, 89, 92, 98, 103, 131, 132, 139

MOEA/D Multiobjective Evolutionary Algorithm Based on Decomposition. 11, 50–55, 108, 132, 135, 145

NSGAII Non-dominated Sorting Genetic Algorithm II. 11, 18–20, 50–55, 66, 70, 75–77, 79, 81, 83, 84, 86, 90, 92, 93, 108, 109, 131, 133, 135, 136, 145, 147, 148

PCA Principal Component Analysis. 27, 29, 145

PSO Particle Swarm Optimization. 18–20, 75–77, 81, 83, 86, 90, 92, 109, 131, 136, 145

ROS Robot Operating System. 50, 66, 93

SDT Safety Distance Testing. 60, 61, 63–65, 67, 68, 70, 132

SUT System Under Test. 4–12, 15–21, 25–27, 33, 34, 36, 40–52, 54, 55, 60, 61, 63, 65, 67–71, 73, 84–87, 93, 94, 97, 98, 101–104, 106–112, 131, 132, 135–137, 145

UAV Unmanned Aerial Vehicle. xi, xii, 3–18, 21, 25–29, 32–37, 39–41, 43–52, 54, 55, 59–71, 73–79, 81, 83–94, 98, 101–113, 131, 132, 135, 136, 145

XML Extensible Markup Language. 34, 40

List of Figures

List of Tables

A. JSON Schema

In this part of the appendix, we provide the entire JSON schema for specifying logical scenarios for a quadcopter, for which we present an excerpt in Section 3.3.2. We derived this schema from the ontology shown in Section 3.3.2.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://logical_scenarios.com/scenario.schema.json",
  "title": "Logical Scenario",
  "description": "A logical scenario that can be used for testing the
   ↪  safe behavior of UAVs",
  "type": "object",
  "properties": {
    "name": {
      "description": "The name of the logical scenario",
      "type": "string"
    },
    "system": {
      "description": "The system dimensions of the logical scenario
       ↪  describing the UAVs and, if applicable, their cooperation
       ↪  mechanism",
      "type": "object",
      "properties": {
        "UAVs": {
          "description": "The UAVs that are present in the logical
           ↪  scenario",
          "type": "array",
          "items": {
            "description": "A single UAV in the logical scenario with
             ↪  its maneuvers and, if applicable, failures",
            "type": "object",
            "properties": {
              "maneuvers": {
                "description": "A list of maneuvers for the single UAV
                 ↪  in the logical scenario",
                "type": "array",
                "items": {
                  "description": "A single maneuver of a UAV in the
                   ↪  logical scenario",
```

```
                "type": "string",
                "enum": ["take off", "hover", "landing", "move to
                 ↪ waypoint", "exploration mission"]
              }
            },
            "failures": {
              "description": "A list of failures for the single UAV in
               ↪ the logical scenario",
              "type": "array",
              "items": {
                "description": "A single failure of a UAV in the
                 ↪ logical scenario with its type and degree",
                "type": "object",
                "properties": {
                  "type": {
                    "description": "The type of failure of the single
                     ↪ UAV in the logical scenario",
                    "type": "string",
                    "enum": ["motor", "rudder", "aileron", "GPS",
                     ↪ "control loss"]
                  },
                  "degree": {
                    "description": "The degree of the failure of the
                     ↪ single UAV in the logical scenario",
                    "type": "string",
                    "enum": ["none", "light", "moderate", "serious"]
                  }
                },
                "required": ["type", "degree"],
                "additionalProperties": false
              }
              }
          },
          "required": ["maneuvers"],
          "additionalProperties": false
        },
        "additionalProperties": false
      },
      "cooperation": {
        "description": "The cooperation mechanism in the logical
         ↪ scenario describing the communication, coordination,
         ↪ organization, and awareness of the UAVs",
        "type": "object",
        "properties": {
          "communication": {
            "description": "The communication dimension of the
             ↪ cooperation mechanism in the logical scenario",
```

```
            "type": "string",
            "enum": ["direct", "indirect", "none"]
          },
          "coordination": {
            "description": "The coordination dimension of the
             ↪ cooperation mechanism in the logical scenario",
            "type": "string",
            "enum": ["with protocol", "without protocol", "none"]
          },
          "organization": {
            "description": "The organization dimension of the
             ↪ cooperation mechanism in the logical scenario",
            "type": "string",
            "enum": ["centralized", "decentralized", "hybrid"]
          },"knowledge": {
            "description": "The knowledge dimension of the cooperation
             ↪ mechanism in the logical scenario",
            "type": "string",
            "enum": ["aware", "unaware"]
          }
        },
        "required": ["communication", "coordination", "organization",
         ↪ "knowledge"],
        "additionalProperties": false
      }
    },
    "required": ["UAVs"],
    "additionalProperties": false
  },
  "environment": {
    "description": "The environment dimensions of the logical
     ↪ scenario, including flight area, weather, and, if applicable,
     ↪ obstacles",
    "type": "object",
    "properties": {
      "flight area": {
        "description": "The flight area in the logical scenario with
         ↪ its landform and surface nature",
        "type": "object",
        "properties": {
          "landform": {
            "description": "The landform of the flight area in the
             ↪ logical scenario",
            "type": "string",
            "enum": ["flat", "depression", "elevation",
             ↪ "steep_transition"]
          },
```

```
      "surface nature": {
        "description": "The nature of the surface of the flight
         ↪ area in the logical scenario",
        "type": "string",
        "enum": ["land", "mixture", "water"]
      }
    },
    "required": ["landform", "surface nature"],
    "additionalProperties": false
  },
  "obstacles": {
    "description": "The obstacles in the logical scenario, if
     ↪ existing",
    "type": "array",
    "items": {
      "description": "A single obstacle in the logical scenario
       ↪ describing its kind, size, and form",
      "type": "object",
      "properties": {
        "kind": {
          "description": "The kind of a single obstacle in the
           ↪ logical scenario",
          "type": "string",
          "enum": ["static", "dynamic"]
        },
        "size": {
          "description": "The size of a single obstacle in the
           ↪ logical scenario",
          "type": "string",
          "enum": ["small", "medium", "large"]
        },
        "form": {
          "description": "The form of a single obstacle in the
           ↪ logical scenario",
          "type": "string",
          "enum": ["cuboid", "sphere", "cylinder", "cone",
           ↪ "torus"]
        }
      },
      "required": ["kind", "size", "form"],
      "additionalProperties": false
    },
    "additionalProperties": false
  },
  "weather": {
```

```
"description": "The weather in the logical scenario with its
 ↪  lighting, wind force, temperature, precipitation,
 ↪  lightning, reduced visibility, and cloud cover",
"type": "object",
"properties": {
  "lighting": {
    "description": "The lighting in the logical scenario",
    "type": "string",
    "enum": ["none", "dim", "normal", "bright"]
  },
  "wind force": {
    "description": "The wind force in the logical scenario",
    "type": "string",
    "enum": ["none", "light", "moderate", "strong"]
  },
  "temperature": {
    "description": "The temperature in the logical scenario",
    "type": "string",
    "enum": ["cold", "moderate", "hot"]
  },
  "precipitation": {
    "description": "The precipitation in the logical
     ↪  scenario",
    "type": "string",
    "enum": ["none", "light", "moderate", "heavy"]
  },
  "cloud cover": {
    "description": "The cloud cover in the logical scenario",
    "type": "string",
    "enum": ["none", "moderate", "heavy", "complete"]
  },
  "reduced visibility": {
    "description": "The reduced visibility in the logical
     ↪  scenario",
    "type": "string",
    "enum": ["none", "fog", "heavy_fog", "thick_fog"]
  },
  "lightning": {
    "description": "If lightning exists in the logical
     ↪  scenario",
    "type": "string",
    "enum": ["none", "existing"]
  }
},
"required": ["lighting", "wind force", "temperature",
 ↪  "precipitation", "cloud cover", "reduced visibility",
 ↪  "lightning"],
```

```
            "additionalProperties": false
          }
        },
        "required": ["flight area", "weather"],
        "additionalProperties": false
      }
    },
    "required": ["name", "system", "environment"],
    "additionalProperties": false
}
```

B. Details on Experimental Settings

In this chapter, we provide detailed settings for our experiments in this work to enable the reproducibility of the presented results. Further, we share the source code for our tool StellaUAV, presented in Chapter 7, in [117].

Settings for the experiments in Chapter 3:

- fastdtw library [107] version 0.3.4 for computing the DTW distances
- Scikit-learn framework [95] version 1.0.2 for implementing the clustering algorithms, the metrics for finding an optimal number of clusters k, the min-max-normalization, and the PCA
- kneed library [111] version 0.7.0 for the implementation of the Kneedle algorithm

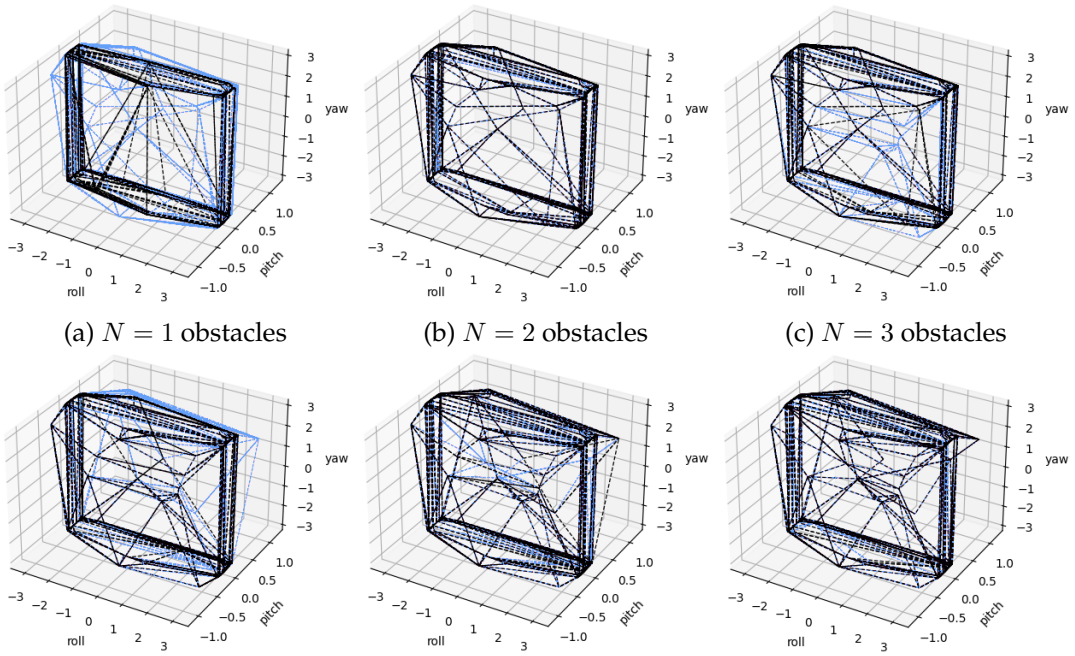
Settings for the experiments in Chapters 4, 5, 6, and 7:

- Used operating system: Ubuntu 18.04
- System under test (SUT): PX4 autopilot for UAVs [81] version 1.11.3 with the obstacle avoidance extension version 0.3.1
- Applied simulator: Gazebo [61] version 9.0.0
- Additional libraries:
  - ROS Melodic Morenia [100] for collecting the SUT's data
  - MAVSDK-Python library [92] version 0.6.1 for forwarding a mission to the SUT
  - jMetalPy framework [15] version 1.5.5 for the implementation of the optimization algorithms NSGAII, PSO, and MOEA/D:
    * NSGAII: SBX crossover operator (crossover rate $CR = 0.9$), polynomial mutation operator (mutation rate $MR = 1/num\_variables$), and binary tournament selection operator
    * PSO: Implementation of the speed-constraint Multi-objective PSO with a polynomial mutation operator ($MR = 1/num\_variables$), and a crowding distance archive
    * MOEA/D: Differential evolution crossover operator ($CR = 1.0$), polynomial mutation operator ($MR = 1/num\_variables$), Tschebycheff as the aggregation function, and a neighborhood selection probability of 0.9

    - For the evaluated search space and specific values of additional parameters, such as the population/swarm size, please refer to the corresponding setup section presented for each experiment in this work.
  - BayesianOptimization framework [91] version 1.2.0 for the implementation of the optimization algorithm BO:
    - BO: Expected improvement as the acquisition function and $xi = 0.05$
    - For the evaluated search space and specific values of additional parameters, such as the initial population size, please refer to the corresponding setup section presented for each experiment in this work.

C. Visualization of Convex Hulls for NSGAII

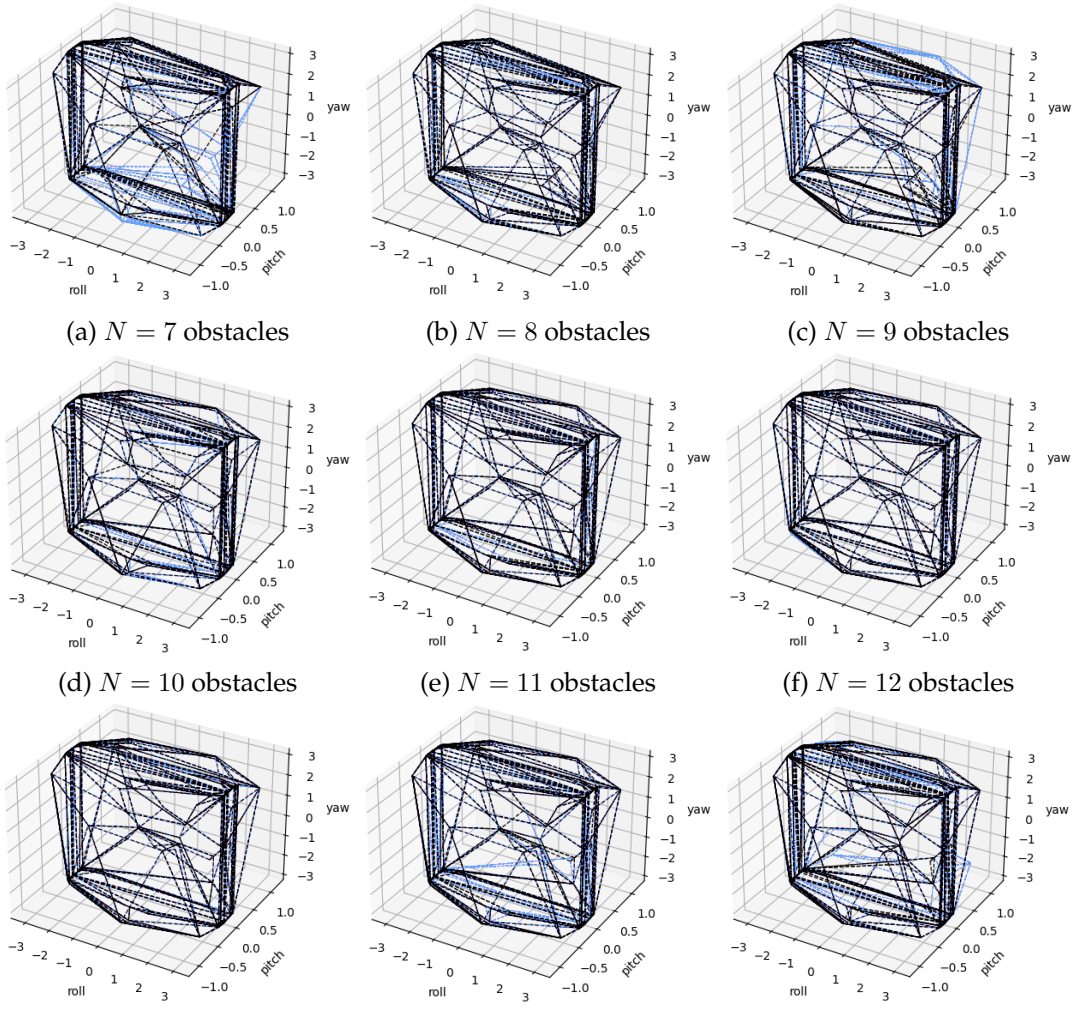
This chapter presents additional visualization of the convex hulls we build in our experiments for finding an upper bound for the number of obstacles in Section 4.3. We display the convex hulls when gathering orientation values with NSGAII for $N \in \{1, 2, ..., 15\}$ obstacles in Fig. C.1 and Fig. C.2.



(a) $N = 1$ obstacles (b) $N = 2$ obstacles (c) $N = 3$ obstacles

(d) $N = 4$ obstacles (e) $N = 5$ obstacles (f) $N = 6$ obstacles

Figure C.1.: Visualization of the convex hulls that present the collected orientation values for NSGAII for $N \in \{1, 2, ..., 6\}$ obstacles, which implicitly affect the range of the orientation values.



(a) $N = 7$ obstacles (b) $N = 8$ obstacles (c) $N = 9$ obstacles

(d) $N = 10$ obstacles (e) $N = 11$ obstacles (f) $N = 12$ obstacles

(g) $N = 13$ obstacles (h) $N = 14$ obstacles (i) $N = 15$ obstacles

Figure C.2.: Visualization of the convex hulls that present the collected orientation values for NSGAII for $N \in \{7, 8, ..., 15\}$ obstacles, which implicitly affect the range of the orientation values.