Reproducible Benchmarking Platform for Networked Control Systems

Samuele Zoppi[1], Onur Ayan[1], Fabio Molinari[2], Zenit Music[2], Sebastian Gallenmüller[3], Georg Carle[3], Wolfgang Kellerer[1]

[1]Chair of Communication Networks, Department of Electrical and Computer Engineering, TUM, Germany
[2]Control Systems Group, Department of Electrical Engineering and Computer Science, TU Berlin, Germany
[3]Chair of Network Architectures and Services, Department of Informatics, TUM, Germany
Email: {samuele.zoppi, onur.ayan, sebastian.gallenmueller, carle, wolfgang.kellerer}@tum.de, molinari@tu-berlin.de, zenitmusic@mailbox.tu-berlin.de

ABSTRACT

The evolution of the Internet of Things accelerated the development of Cyber-Physical Systems. Among them, Networked Control Systems (NCSes) gained notable attention thanks to their application to industrial operations. Experimental NCSes require expertise from control, computation and communication disciplines. This requirement, together with the fragmentation of hardware and software used for the implementation of NCSes, represents a challenge for the reproducibility of research results. In this paper, we present the first reproducible experimental benchmarking platform for NCSes. The proposed platform is open-source and designed to be easily reproducible and extensible by anyone. Additionally, we present an NCS benchmarking methodology that aids the reproducibility of NCS experiments. To this end, it defines the parameters of the experiment and the relevant Key Performance Indicators (KPIs) that need to be observed during its execution. Finally, we evaluate in details the proposed KPIs and validate the benchmarking methodology by reproducing the platform and comparing the KPIs in different scenarios. Results present the performances of the two platforms and prove the validity of the proposed NCS benchmarking methodology.

KEYWORDS

CPS, NCS, Implementation, Open-source, Benchmarking, Reproducibility, Experiment, Key Performance Indicators, Delay, Jitter, QoC.

1 INTRODUCTION

The advent and evolution of the Internet of Things, where billions of devices have gained networked connectivity and Internet access, contributed to the rapid development of Cyber-Physical Systems (CPSes)[20]. A CPS consists of the interconnection of sensors and actuators with a computation logic, together interacting over the same physical system. In fact, in CPSes, the information acquired by sensors is processed and used to instruct actuators to perform specific actions.

A specific class of CPSes, where networked sensors and actuators logically belong to the same automatic control system, is called Networked Control Systems (NCSes) [12]. Over the last decade, NCSes gained popularity thanks to the capability of distributing control functions over a communication network, thus enhancing the flexibility and functionalities of existing control systems. Their application is particularly relevant to industrial operations, where NCSes can be employed, for instance, in the closed loop regulatory control of production machines [11]. For these reasons, a considerable amount of work has been performed in the literature that models the behavior of NCSes to understand their performances in different operating conditions [21].

Despite the large number of results achieved by the control, computation, and networking research communities, the reproducibility and comparison of NCS experimental results still represent an obstacle to overcome. This challenge arises for different reasons. First, NCS theory requires expertise belonging to control, computation, and networking domains, where different methodologies and procedures have been developed to evaluate the performances of their systems. Second, diverse hardware platforms and software tools are available and have been used to implement NCSes. The fragmentation of hardware, software, and expertise increased the difficulty of reproducing and comparing NCS research results.

The issue of reproducing results generated from digital artifacts has already been addressed by the Association for Computing Machinery (ACM). Following the International Vocabulary of Metrology [1], experimental results are *reproducible* if the results can be recreated by other researchers using different equipment. In the context of CPSes, the issue of reproducibility was addressed in the 1st workshop on

benchmarking CPSes (CPSBench18), part of the CPSweek 2018, a premier annual event on CPSes. Research work presented in the workshop proposed performance metrics and benchmarking scenarios designed for reproducible experimental results. However, most of the contributions describe conceptual methods and do not tackle the insidious complications of reproducing CPS experiments.

This paper enables reproducible NCS research experiments. It enables this by (i) presenting the implementation details of the first open-source[1] NCS benchmarking platform designed for reproducible results, (ii) proposing a novel NCS benchmarking methodology based on the joint expertise of control, computation, and communication, and (iii) evaluating the reproducibility of the platform and the validity of the methodology with experiments in different scenarios.

The remainder of this paper is structured as follows. In Sec. 1.1, a review of the state-of-the-art is discussed. In Sec. 2, the implementation details of the proposed NCS benchmarking platform are presented. Sec. 3 defines a novel NCS benchmarking methodology for reproducible experimental results. Sec. 4 presents the results of the evaluation of the proposed platform by applying the proposed methodology in different benchmarking scenarios. Finally, Sec. 5 summarizes the achieved results and discusses improvements and future work.

1.1 Related Work

NCSes have been extensively researched in the literature. A vast majority of the existing research work follows a theoretical approach. However, as stated by Lu et al. [16], conveying full-scale practical research with a real implementation of a CPS is a difficult task due to the complexity and the replicability of experimental platforms. Therefore, research work in the field of NCSes conducting experimental studies is rather limited. Chamaken et al. [6] implement a hybrid setup of an NCS combining hardware in the loop, i.e. a simulation of the plant dynamics, with a real network. On the contrary, experimental results of Eker et al. [8] use the network in the loop approach, i.e. a simulated network, with real hardware as a control system. A different research approach provides prominent examples where the complete NCS consists of real hardware [3, 4, 7]. Drew et al. [7] propose an NCS design that takes into account network delays and packet dropouts, and evaluate it in an experimental scenario. They show that, by optimizing the network-aware control logic, their system performs better than the conventional network-unaware controllers. Bachhuber et al. [3] conduct an end-to-end latency analysis of a vision-based NCS. On the other hand, Baumann et al. [4] present measurement results from the case study of balancing an inverted pendulum over a multi-hop wireless network. However, in all these cases, authors do not address the issue of repeating their experimental research results.

Although practical NCS implementations pose a major challenge in reproducing NCS experiments, there has been an attempt in the literature to define conceptual CPS benchmarking scenarios tackling reproducibility. In particular, Boano et al. [5] and Gallenmüller et al. [9] elaborate on how to implement experimental benchmarks and define key performance indicators (KPIs) for the comparison of experimental results. Nonetheless, they do not conduct a practical study for the verification of the proposed methods in their own work. To the best of our knowledge, none of the existing literature tackles the problem of reproducibility and benchmarking in a full-scale practical scenario. Therefore, in this paper, we present the first experimental platform and practical benchmarking methodology for NCSes.

2 NCS PLATFORM IMPLEMENTATION

In this section, the implementation details of the proposed open-source[1] NCS benchmarking platform are presented. Our platform aims at controlling, using a common IP communication network, a so-called *two-wheeled inverted pendulum robot* (TWIPR), which has a long tradition in literature [13, 18] and industry [19].

The implementation was developed keeping in mind reproducibility design principles, providing a platform that can easily be reproduced and deployed for arbitrary research purposes. Hence, all the software and hardware components used in the proposed platform are low-cost and highly accessible. In fact, our TWIPR is built using the Lego Mindstorms™ platform, communicates using standard Ethernet and W-LAN network interfaces, is open-source[1], and is programmed using the Python programming language that is supported by the vast majority of the operating systems and computing machines.

The description of the implementation is organized as follows. First, Sec. 2.1 describes the architecture of our NCS together with the description of its components. Afterwards, Sec. 2.2 describes the time and delay model used to analyse the NCS and to design the control logic. The formal description of the control problem of balancing the TWIPR is discussed in Appx. A.

2.1 NCS Architecture

The proposed NCS has the architecture presented in Fig. 1 and is organized according to the three CPS domains [15]: control systems, computing systems, and communication networks.

The set of elements composing the *control system* is twofold. On one side, the plant, i.e. the TWIPR or robot, mounts sensors and actuators capable of sensing the physical system

[1] https://github.com/tum-lkn/iccps-release.

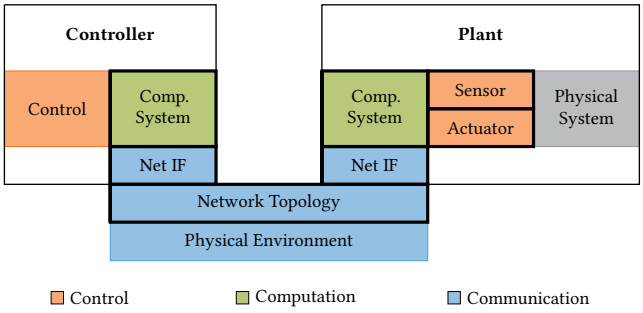


Figure 1: Architecture of the NCS platform. The control, computation, and communication domains of CPS are represented. Every box is a different component of the architecture, boxes surrounded by thick black contours represent hardware elements.

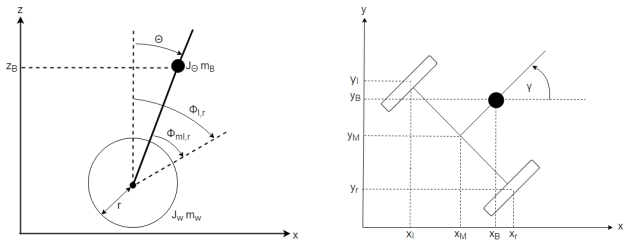
and executing the actuation commands respectively. On the other side, the controller, which is detached from the plant, receives the sensors readings, executes the control logic, and transmits instructions to the actuator. Due to its complexity, its logic is fully described in Appx. A.

Two different *computing systems* provide computing power and access to the network interfaces to both controller and plant. The interconnection of the control application with the network interface is achieved with the implementation of the upper-layer protocols of the OSI communication stack.

The *communication network* physically interconnects the computing system of the controller with the computing system of the plant and enables the flow of information between them. In our architecture, it defines the lower-layers of the OSI communication stack.

2.1.1 Control System. The plant is built by following the default instructions of the *Gyro Boy* robot of the Lego Mindstorms Education EV3 Core Set™ until step 61 [10].

As represented in Fig. 2, the robot's body is supported by two wheels, each one splined to an actuator, the *DC brushed EV3 Large Servo Motor™*, capable of rotating it. The control variables are the voltages applied to the left and right motors,



(a) Side view of the robot. **(b) Top view of the robot.**

Figure 2: Two-wheeled Inverted Pendulum Robot.

respectively, $u_l(t), u_r(t) \in [-\bar{V}, \bar{V}]$, where $\bar{V}$ is the full-scale voltage of the motor and equal to $8\,V$.

An *incremental encoder* is mounted on each motor's shaft and measures the rotation angle of the corresponding wheel with regards to the robot's body. As in Fig. 2a, $\Phi_l(t)$ and $\Phi_r(t)$ indicate the rotation of the left and rights wheels with regards to z-axis. As in [13], $\Phi(t)$ describes the average rotation angle of the two wheels with regards to the z-axis, i.e. $\Phi(t) = \frac{1}{2}[\Phi_l(t) + \Phi_r(t)]$. The robot can move onto a 2D plane, i.e. the plane formed by axes x and y in Fig. 2b. The position of the body in the 2D plane at time t is $(x_M(t), y_M(t))$. The system is inherently nonholonomic.

A *one-dimensional gyroscope*, the EV3 Gyro Sensor™, is mounted on the body and measures the pitch rate $\dot{\Theta}(t)$. With regards to Fig. 2a, $\Theta(t)$ is called pitch angle and denotes the angle at time t between the z-axis and the axis passing through the robot's body. Its derivative over time $\dot{\theta}(t)$ is referred to as the pitch rate. Due to the gravity force, intuitively, the position $\Theta(t) = 0$ exhibits an unstable equilibrium. The control goal is, thus, to balance the robot, i.e. to hold $\Theta(t) = 0$, while tracking a desired position and orientation in plane $x - y$, i.e. to hold $(x_M(t), y_M(t), \gamma(t)) = \left(x_M^{ref}(t), y_M^{ref}(t), \gamma^{ref}(t)\right)$. This task can be achieved by employing a closed-loop controller, that gets the sensors measurements and computes the adequate control action to be delivered to the two motors. The extensive derivation of the control law can be found in Appx. A.

2.1.2 Computing Systems. Two computing systems are deployed in our implementation; one for the controller and one for the robot differing in requirements.

The robot should be mobile and battery powered, requiring a computing system optimized for compact size and low energy consumption. Any PC available to a researcher should be able to run the controller, i.e. we assume a powerful multi-purpose 64-bit computer and one of the widely spread operating systems: Windows, macOS, or Linux. Such a controller offers a flexible platform for implementing powerful control algorithms that could not be processed on the resource constrained robot.

Both computing systems must implement compatible higher-layers communication protocols. For this reason, both computing systems deploy the widely spread TPC/IP network stack and the same application protocol. It consists of two messages; the sensor value message, created by the robot and sent to the controller, and the actuation command message, created by the controller replying to the sensor value message, containing the voltages to be applied at the motors. In addition, sequence numbers and timestamps are transmitted for packet loss, reordering detection, and delay measurements.

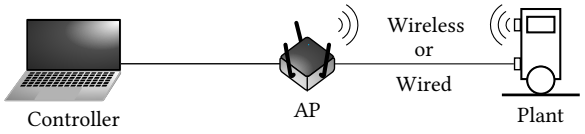


Figure 3: Two-hop network topology of the implemented NCS benchmarking platform. The robot supports two network interfaces: wired via an Ethernet adapter or wireless via a 2.4 GHz W-LAN dongle.

2.1.3 Communication Network. Our network is designed to be easily reproducible and flexible with respect to the possible communication technologies. It is structured according to the OSI communication model and logically separated from the computing system at the network layer, i.e. everything below is part of the communication network.

The *network topology* defines the connectivity of different network nodes at the network and link layers. In our case, a simple two-hop topology is implemented and shown in Fig. 3. The first hop connects the controller to a W-LAN Access Point (AP) via Ethernet. The second hop connects the AP to the robot in two different configurations: either wired Ethernet or wireless W-LAN network interfaces. In our architecture, the *network interfaces* define the link-layer medium access scheme. The robot has no native network interface, wired and wireless connections are realized via the USB 2.0 interface, which simplifies the choice of the network technology in use. For any given experiment in this paper, only one of the two connections is used exclusively.

Finally, the *physical environment* describes the physical characteristics of the communication and is particularly important in wireless. In our platform, the wireless communication between the robot and the AP takes place in a quiet indoor office environment, at an approximate distance of 1-2 m, and it is subject to low external interference.

2.2 Timing and Delays Model

When all the components of the NCS architecture are interconnected, information regularly flows between the plant and the controller over the communication network. In particular, sensor readings from the plant are transmitted to the controller, which uses this information to calculate and send up-to-date actuation commands to the actuator. The plant is responsible for the periodic operation of the control loop and regularly triggers sensor readings every T_s.

The time evolution of the k-th sampling period is shown in Fig. 4. At time $t^k_{\mathrm{S,R}}$, the sensor values of the robot's encoders and gyroscope are taken, handed over to the robot's operating system at $t^k_{\mathrm{S,STX}}$ and transmitted over the communication network at $t^k_{\mathrm{S,NTX}}$. The controller's network interface receives the sensor data at $t^k_{\mathrm{S,NRX}}$ and its operating system delivers the packet to the control application at time $t^k_{\mathrm{S,SRX}}$. Afterwards, the controller calculates the actuation values for the motors and hands over the actuation message to the operating system at $t^k_{\mathrm{A,STX}}$, which sends the packet over the network at $t^k_{\mathrm{A,NTX}}$. Finally, at time $t^k_{\mathrm{A,NRX}}$, the robot's network interface receives the actuation packet, and, at time $t^k_{\mathrm{A,SRX}}$, its operating system delivers it to the actuator application, which applies the commands to the two motors at $t^k_{\mathrm{A,W}}$.

Thanks to the timing diagram shown in Fig. 4, it is possible to identify the delay components of the NCS and distinguish the delays arising from the control system, computing system, and communication network. Control system delays arise from the *processing time* of the control algorithms. At the robot during sensing $d^k_{\mathrm{P,S}}$ and actuation $d^k_{\mathrm{P,A}}$, and at the controller computing the control logic $d^k_{\mathrm{P,C}}$. Computing systems delays arise while *processing* the packets from and to the network interface at the robot $d^k_{\mathrm{P,STX}}$, $d^k_{\mathrm{P,ARX}}$ and at the controller $d^k_{\mathrm{P,SRX}}$, $d^k_{\mathrm{P,ATX}}$. Finally, network delays can be classified in uplink delay $d^k_{\mathrm{N,S}}$, when sensor values are transmitted, and downlink delays $d^k_{\mathrm{N,A}}$, when actuation commands are transmitted.

In an ideal operation, all the *delays* are bounded and within the sampling period of the control loop. However, in a real implementation, the delays vary according to the chosen software and hardware of the control system, computing system and communication network. While shorter delays can be compensated via *busy waiting*, the event of higher delays must be carefully taken into account using a proper control strategy. Both cases are taken into account by the control logic and that is described in Appx. A.

Moreover, *packet loss* additionally affects the operation of our NCS. In general, packet loss can occur for several reasons, such as buffer overflows in the operating systems and in the network elements, or due to transmission errors arising from the physical transmission of the packet. In our NCS, we assume that packet loss is exclusively introduced by wireless communication and that the event of packet loss additionally arises whenever a packet experiences a delay larger than a specific *delay upper-bound*. This model is a valid design choice for NCSes. In fact, delay is an accepted performance metric in control, computation, and communication domains, and allows the separate optimization of the three systems.

3 BENCHMARKING METHODOLOGY

In this section, a novel NCS benchmarking methodology is presented. The methodology relies on the experience gained during the implementation of the proposed platform, and on the combined knowledge of control, computation, and communication domains. The purpose of the proposed methodology is to define the necessary amount of information in

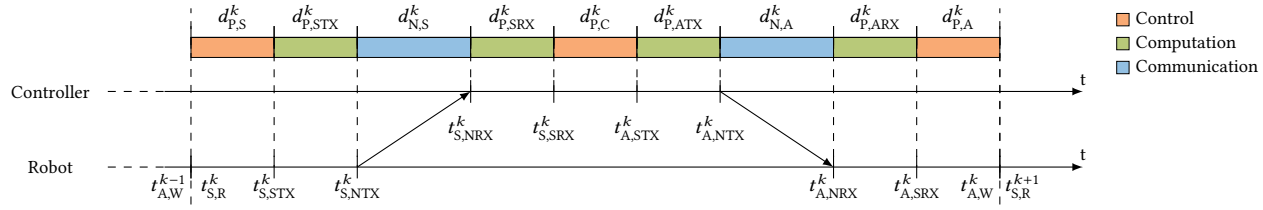


Figure 4: Model of the different *processing* (P) and *networking* (N) delay components of the control, computation, and communication CPS domains of the implemented NCS.

order to reproduce and evaluate experimental results using the NCS platform. In accordance to ACM's reproducibility terminology [1], we first want to recreate our own results thereby establishing repeatability. In a second step, we recreate the NCS benchmark across the different involved research groups making our results replicable. We provide the entire framework containing the source code, the plotting scripts, and the measurements used for this paper as open-source[1], thereby encouraging others to recreate our results and fostering the development towards a fully reproducible benchmark.

First, we define *benchmark* as a series of experiments. An *experiment* is a time-bounded execution of the NCS platform. In order to reproduce experiments, the conditions of the experimental evaluation must be detailed. In particular, the *duration* T_e of the experiment must be defined. In our benchmark, experiments use a real-world setup and different setups are possible, e.g. obtaining values through simulation or emulation. The result of each experiment is a set of values. From these results, we derive *key performance indicators (KPIs)* describing the most relevant features of the system during the experiment. The KPIs may depend on the settings an experiment is performed in. These settings, relevant to describe an experiment, we call *scenario*. A whole series of different experiments and scenarios might be necessary to create a comprehensive report for the behavior of the NCS.

3.1 Scenario Description

The *scenario description* describes all the relevant parameters of an experiment and it is based on the NCS architecture presented in Sec. 2.1 and depicted in Fig. 1. In fact, it is structured according to the three different CPS domains: control systems, communication networks, and computational units. For every component of the NCS architecture, software (algorithms), and hardware parameters must be specified to replicate the experiments. To make the experiments and ultimately the benchmarks reproducible, it is key to document all relevant information of a scenario in a *scenario description*. For this reason, in Tab. 1, we provide an exemplary scenario description by summarizing all the components of the proposed platform A of Sec. 2, and of a second reproduced platform B. The scenario above only presents a minimal description of the most basic setup for our TWIPR.

3.1.1 Control Parameters. The *control application software* running on the computing systems and implementing the control logic that drives the NCS. The *physical system*, describing the physical properties of the robot itself. The *hardware* used on the robot, such as the used sensors and actuators.

3.1.2 Network Parameters. The *network topology*, describing the connectivity between the nodes of the network.

In the scenario description, all the network parameters are part of the *lower layers*. This involves all functions being part of the network layer, link layer, and physical layer, which are implemented in the *network stack* and *network drivers* included in Linux, and in the eventual *firmware* executed by the network interface cards (NICs).

The *network hardware*, i.e. the hardware models of the network interfaces used by robot and controller.

The *physical environment*, defining the physical conditions that the network operates in, such as the interference with other wireless nodes. These properties strongly affect wireless networks, making them inherently difficult to reproduce without an echo chamber. For our benchmark, we try to minimize the impact of the physical environment on the measured results. Our benchmark should be widely reproducible across different research groups, therefore, we decided to not require access to special equipment such as echo chambers. For this reason, we suggest to execute the benchmark in a quiet wireless environment, thereby minimizing the impact of external interference and moving objects on the measurement results. For benchmarks using wired networks, such as a full-duplex switched Ethernet, the physical environment has no impact on the measurement results as long as the network is not overloaded. Therefore, wired network measurements are easier to reproduce and can even be used to emulate the behavior of wireless networks on the network layer.

Parameter	Description
Control Application SW	Python 3 open-source[1] code implementing the control logic of described in Appx. A.
Control Physical Sys.	*Gyro Boy* robot of the Lego Mindstorms Education EV3 Core Set™.
Control HW	DC brushed EV3 Large Servo Motors, EV3 Gyro Sensor.
Network Topology$_A$	Two-hop network shown in Fig. 3 connected via the AP TP-Link TL841ND.
Network Topology$_B$	Two-hop network shown in Fig. 3 connected via the AP Edimax BR6208AC.
Network Stack Controller	Ubuntu 18.04 LTS (Kernel version 4.15).
Network Stack Robot	Debian Jessie (Kernel version 4.4).
Network HW Controller$_A$	Intel 82579LM 1 GbE NIC.
Network HW Controller$_B$	ASIX AX88179 1 GgE.
Network HW Robot$_A$	Apple A1277 USB-to-Ethernet dongle, Edimax EW-7811Un W-LAN USB dongle.
Network HW Robot$_B$	Edimax EU-4306 USB-to-Ethernet dongle, Edimax EW-7811Un W-LAN USB dongle.
Network Physical Env.	Quiet office environment (low interference, no moving objects), indoor 1-2 m.
Computing Sys. Higher layers	UDP, application protocol described in Sec. 2.1.2.
Computing Sys. HW Controller$_A$	Intel Core i2520M (2 cores, 2.5 GHz, 8 GiB RAM).
Computing Sys. HW Controller$_B$	Intel Core i7-6700 (4 cores, 3.40 GHz, 16 GiB RAM).
Computing Sys. HW Robot	32-bit ARM9 SoC (1 core, 300 MHz, 64 MiB RAM).

Table 1: Summary of scenario description parameters for two NCS platforms. Platform A, developed during the first implementation, and platform B, reproduced for benchmarking purposes.

KPI	Description
$d_{P,S}$	Sensor readings on the robot.
$d_{P,C}$	Calc. of actuator commands on controller.
$d_{P,A}$	Execution of actuator commands on the robot.
d_N	Average one-way network delay including stack processing on robot and controller.
$\hat{\Delta}_T - d_{P,A}$	Robot round-trip delay.
$\hat{\Delta}_T$	Measured variable sampling period.

Table 2: Summary of time KPIs.

3.1.3 Computing System Parameters. The *higher layers*, i.e. the transport layer and higher layer protocols, are part of the computing system, connecting the control and the networking domains of the NCS. The *transport protocol* is implemented in the OS, therefore the OS version is required for describing the computing system parameters. The *application protocol*, used for the logical exchange of sensor values and actuation commands between the controller and the plant.

The *hardware*, which provides computing power and access to the communication facilities.

3.2 KPIs

The KPIs capture the most important metrics necessary to analyze and understand the operation of the NCS platform during the benchmarking experiment.

Sec. 2.2 and Appx. A describe the operation of the platform during its execution. A fundamental aspect that has emerged during the implementation and analysis of our platform is the role of time and delays in the system. In fact, their precise analysis enables the controller to operate even in presence of a variable delays and packet loss. For this reason an important part of the proposed KPIs is relative to time and delays.

On the other hand, the Quality of Control (QoC) is the second dimension that needs to be evaluated in NCSes. In general, QoC is a metric that describes the performance of a control system. In our example, it can be used to determine the ability of the control logic to compensate variable delays and packet loss, and to compare it with other control strategies.

3.2.1 Time KPIs. Delays, i.e. the time needed for information exchange and processing on the robot and the controller, influence the overall performance of the NCS. Delays can arise from control, computation, or communication, with lower delays offering a better service for the NCS. We assess the performance by measuring the individual delays presented in Fig. 4. To evaluate the influence of the network stack of the controller, we recorded a packet trace on the ingress/egress network interface via tcpdump.

Recording network delays and performing clock synchronization required a constant packet exchange and increased processing, thus overloading the CPU of the robot and impacting the control performance. Due to this limitation, we did not record the specific delays $d_{P,STX}$, $d_{N,S}$, $d_{P,SRX}$, and $d_{N,A}$ attributed to the network communication on the robot. For this reason, we calculate the average one-way network delay d_N assuming symmetrical network delays, and including the stack delays of controller and plant,

$$d_{\mathrm{N}} = 0.5 \cdot \left[t^{k}_{\mathrm{A,SRX}} - t^{k}_{\mathrm{S,STX}} \right] . \quad (1)$$

As KPI, we report each delay listed in Tab. 2 as median value. We measure the jitter as a property of the delay fluctuation. Low jitter allows a constant stream of information, supporting smooth control performance. To determine jitter, we provide quartiles and 99.9th percentiles in addition to the median delay.

3.2.2 Control KPIs. They indicate the *quality-of control* (QoC) of the NCS and provide a baseline for the comparison of experiments. First, we select the *Integrated Absolute Errors* (IAE) of the states Θ and Φ, i.e. Σ_θ and Σ_Φ. Additionally, we calculate the total control effort over time, i.e., Σ_ν.

$$\Sigma_\Theta = \|\Theta(kT_s) - \Theta_{ref}\| \quad (2)$$

$$\Sigma_\Phi = \|\Phi(kT_s) - \Phi_{ref}\| \quad (3)$$

$$\Sigma_\nu = 0.5 \cdot (\|\nu_l(kT_s)\| + \|\nu_r(kT_s)\|) \quad (4)$$

In our experiments the states' reference is always set to zero, indicating the initial wheels' position and the z-axis of the robot, i.e. $\Theta_{ref} = \Phi_{ref} = \varnothing_{1xTe}$. Σ_Θ and Σ_Φ represent the cumulative absolute deviation of Θ and Φ from their corresponding reference values during the experiment. Smaller values of Σ_Θ and Σ_Φ correspond to a higher QoC. Σ_ν represents the total control effort spent to balance the robot. Also in this case, a smaller Σ_ν indicates better stability and hence a higher control performance.

With respect to the implemented control logic, an additional metric shows the performance of the control system. The *number of predictions* used by the robot to compensate for packet loss and delays must be taken into account. As described in Sec. 2.2 and in Appx. A, a prediction is applied whenever a packet is not received within the delay upper-bound.

4 PLATFORM EVALUATION

In this section, we provide a comprehensive evaluation of the NCS platform and of the benchmarking methodology. We achieve this by presenting the NCS benchmarking KPIs in details and in different scenarios. The evaluation captures the essence of the proposed benchmarking methodology. In fact, experiments were performed reproducing the platform and testing it with different computers and networks.

Every experiment of our evaluation is conducted as follows. Before the experiment starts, the robot lies on the ground continuously sending sensor values to the controller. The controller, however, does not send actuation commands until the robot is manually lifted to the vertical position. For this reason, the beginning of the experiment is the time at which the robot manually reaches the vertical position for the first time, and corresponds to 0 s in our evaluation.

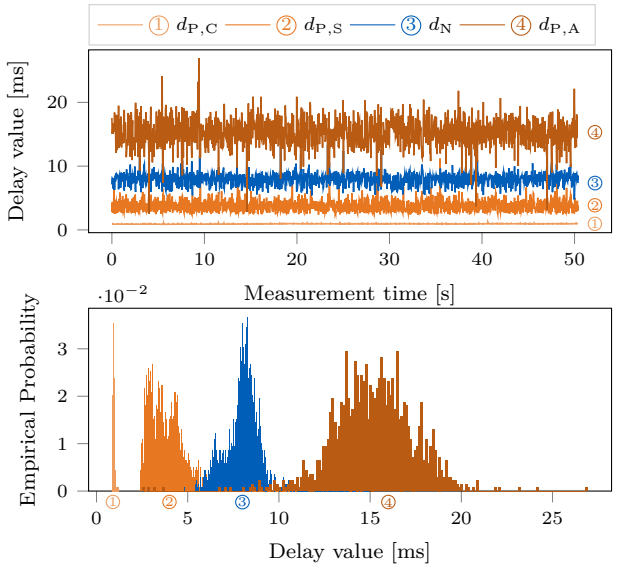


Figure 5: Time evolution and empirical distribution of the delays of the controller, sensor, actuator, and network.

Afterwards, continuous exchange of information between the robot and the controller takes place and enables the control loop to balance the TWIPR. The duration of the experiment is determined by the control logic and is equal to $T_e = 1400$ sampling periods, i.e. 49 s with $T_s = 35\,ms$. Consequently, we set a delay upper-bound equal to 29 ms. This value is smaller than the sampling period and takes into account the additional time needed to instruct the actuators.

Whenever the experiment ends, the controller stops sending actuation messages to the robot, opening the control loop. In this way, for every experiment, an even number of samples is collected, and the KPIs can be correctly calculated and compared.

4.1 KPIs Evaluation

KPIs belonging to the control, computation, and communication domains need to be evaluated to understand the dynamics of an NCS. The scenario selected for the detailed evaluation of the KPIs is described by the parameters of platform A in Tab. 1 communicating over W-LAN.

Fig. 5 shows the time evolution and the histogram of the delays of the controller, sensor, actuator, and network defined in Fig. 4. The sensor reading delay $d_{\mathrm{P,S}}$ ② demonstrates a stable behavior with occasional outliers reaching up to 5.5 ms. Similarly, the controller delay $d_{\mathrm{P,C}}$ ① is very stable, showing almost no outliers. Overhead caused by the controller network stack is constant and marginal (approx. 37 μs) over the entire experiment. The actuator delay $d_{\mathrm{P,A}}$ ④ shows an

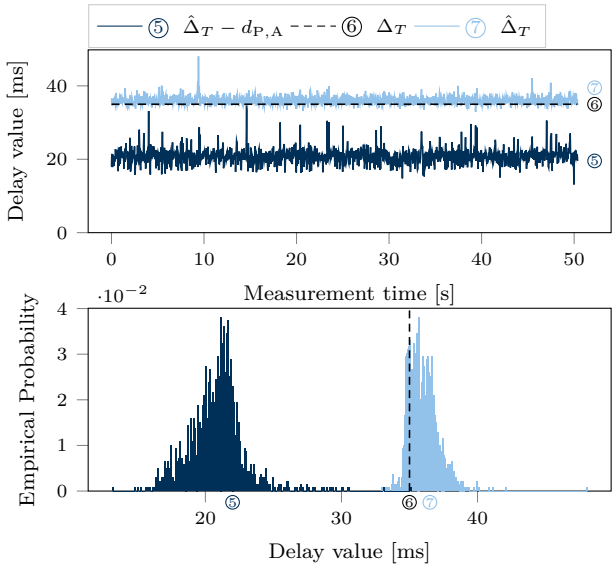


Figure 6: Time evolution and empirical distribution of the round-trip delays and of the measured sampling period.

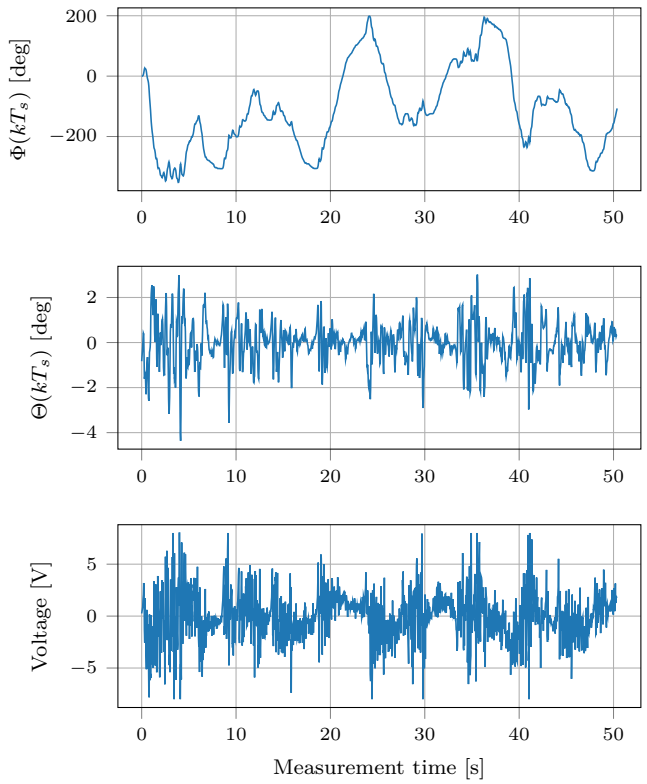


Figure 7: Time evolution of the filtered pitch angle Θ, the filtered avg. rotation angle Φ, and the avg. applied voltage at the motors.

unstable behavior over the entire measurement period. Its jitter, also expressed by the width of its distribution in the histogram, is attributed to the control algorithm, which implements busy waiting. Therefore $d_{P,A}$ ④ includes a waiting period that directly depends on the previous steps and their individual delays. In fact, in order to instruct the motors every 35 *ms*, actuation commands are only applied after a delay upper-bound of 29 *ms* from the beginning of the sampling period. In our platform, approx. 6 *ms* are required to actuate the motors.

When analyzing the jitter given in the histograms, $d_{P,C}$ ① shows the most stable behavior (0.9 ms - 1.2 ms), indicating that the controller always has enough computing power to handle the control process in a timely manner. The sensor reading delay $d_{P,S}$ ② shows a minimal time of 2.4 ms for sensor readings with a tail of up to 11.1 ms. The network delay $d_{N,S}$ ③ roughly resembles a normal distribution ranging from 4.7 ms up to 15 ms, and it is originated by the CSMA/CA mechanism of W-LAN in our physical environment.

Fig. 6 shows the time evolution and histogram of the cumulative delays. The timestamp t_{ASRX} is collected by the robot application after receiving the actuation message, resulting in the delay $\hat{\Delta}_T - d_{P,A}$ ⑤. Where $\hat{\Delta}_T$ is the measured sampling period of the NCS during the experiment. The histogram of ⑤ shows a wide distribution, ranging from 13.2 ms up to 35.1 ms, employing the jitter of all the previous steps. However, if $d_{P,A}$ is included in the plot ($\hat{\Delta}_T$ ⑦), the jitter decreases, as the actuator algorithm applies the actuation commands only 29 ms after the beginning of the sampling period. This effect results in a rather constant measured sampling period $\hat{\Delta}_T$ ⑦, and allows the compensation of the previous delays, leading to a rather low jitter. Thus, the distribution of $\hat{\Delta}_T$ ⑦ is more compact and allows a constant delivery of the actuation commands close to the ideal sampling period Δ_T ⑥. Its jitter is caused by the precision of the busy waiting technique and by the time required to actuate the motors.

The impact of the control logic is reflected in Fig. 7, showing the evolution of the control KPIs. The pitch angle $\Theta(kT_s)$ of the robot is highly varying, with occasional larger spikes every few seconds. Despite this, we can observe that its dynamic remains bounded during the entire execution and that its average value is equal to −0.0014 deg. The evolution of the motors' applied voltage strongly depends on the pitch angle. In fact, higher voltages are correlated with higher values of pitch angle. This effect is also shown in the position of the robot $\Phi(kT_s)$, which presents faster and slower oscillations. Faster oscillations, visible between 3-5 *s*, are caused by strong and opposite actuations commands needed to compensate high values of pitch angles and to balance the robot. Slower oscillations arise whenever the control logic tries to bring the robot to its initial position. This task has lower priority

Delay [ms]	Median ±95% C.I.	Q_1	Q_3	99.9%
A-wired				
$d_{P,C}$	0.94 ± 0.002	0.91	0.97	1.07
$d_{P,S}$	3.55 ± 0.038	3.04	4.24	5.41
d_N	4.38 ± 0.041	4.08	5.03	6.66
$d_{P,A}$	22.20 ± 0.087	20.86	23.16	24.98
$\hat{\Delta}_T$	35.77 ± 0.042	35.21	36.41	37.73
A-wireless				
$d_{P,C}$	0.95 ± 0.002	0.92	0.96	1.05
$d_{P,S}$	3.64 ± 0.049	3.03	4.36	6.20
d_N	8.09 ± 0.053	7.54	8.54	10.88
$d_{P,A}$	15.19 ± 0.118	13.79	16.55	19.94
$\hat{\Delta}_T$	35.89 ± 0.057	35.22	36.62	38.97
B-wired				
$d_{P,C}$	0.39 ± 0.001	0.38	0.39	0.45
$d_{P,S}$	3.89 ± 0.034	3.55	4.49	5.73
d_N	4.61 ± 0.026	4.40	4.82	6.57
$d_{P,A}$	22.38 ± 0.065	21.58	23.10	24.69
$\hat{\Delta}_T$	36.02 ± 0.036	35.55	36.54	37.61
B-wireless				
$d_{P,C}$	0.37 ± 0.001	0.37	0.38	0.43
$d_{P,S}$	3.84 ± 0.040	3.49	4.45	6.39
d_N	5.25 ± 0.055	4.85	6.29	8.74
$d_{P,A}$	21.27 ± 0.126	19.49	22.38	24.84
$\hat{\Delta}_T$	36.32 ± 0.049	35.70	36.95	38.76

Table 3: Time KPIs of the four evaluation scenarios.

compared to balancing the robot, and it is performed on a larger time scale.

4.2 Benchmarking

We prove the validity of the proposed benchmarking methodology and test the reproducibility of our platform by conducting experiments in different benchmarking scenarios.

For this, we have built a second LEGO Mindstorms™ robot and tested it in different physical environments. Its components are fully described by the scenario description of platform B in Tab 1. It consists in a different computing system for the controller, and different network hardware interfaces for both controller and robot. This results in a total of four scenarios for our benchmarking evaluation. We call A-wired the scenario where platform A operates with Ethernet, and A-wireless its operation with W-LAN. Two additional scenarios arise from the reproduced platform B; the B-wired and B-wireless, representing, respectively, the reproduced platform communicating over Ethernet and W-LAN.

Tab. 3 and 4 summarize the benchmark KPIs resulting from the evaluation of the four scenarios. The time KPIs in Tab. 2 are presented as median with 95% confidence intervals, 1st and 3rd quartiles, and 99.9-th percentiles.

	Σ_Θ	Σ_Φ	Σ_ν	Predictions
A-wired	762.91	152090	2066.9	0
A-wireless	938.30	217080	2637.4	10
B-wired	601.51	179590	2804.3	0
B-wireless	785.72	129440	2726.1	1

Table 4: Control KPIs of the four evaluation scenarios.

Tab. 3 shows different performances of the deployed computing systems and communication networks. In fact, the median values of $d_{P,C}$ is lower for platform B than platform A, despite showing similar jitter and worst-case values. A minor difference is noticeable in the sensor processing delays $d_{P,S}$; platform A has smaller median delays but with a higher jitter. Also the average network delays present differences. The median of d_N is always smaller in Ethernet than W-LAN. In addition, W-LAN network delays have a higher variance and worst case delays up to 10 ms. The scenario A-wireless shows the worst network performance, with the highest median value and 99.9-th percentile. The actuator processing delays $d_{P,A}$ directly depend on the busy waiting procedure. Its quartiles reflect the network delays, being wider and with larger worst-case values for wireless communication in both platforms. Finally, the measured sampling period $\hat{\Delta}_T$ is comparable in all four scenario and mainly depends on the busy waiting performed by the actuator. However, it presents a higher median in platform B, and a larger jitter when operating with W-LAN.

Tab. 4 shows comparable values of QoC, for the two NCSes evaluated in the four scenarios. In general, Σ_Θ and Σ_Φ are lower in wired than wireless scenarios thanks to smaller median delays and jitter. However, platform A shows a high value of Σ_Φ caused by the high oscillations introduced by the delays of its W-LAN network interface. The total controller effort Σ_ν is similar across the scenarios, showing a lower value only in scenario A-wired. As expected, actuation predictions on the robot, triggered by packets arriving later than 29 *ms*, were not observed in wired scenarios. However, in the scenario A-wireless, 10 prediction events were observed, and, in the more stable scenario B-wireless, only 1 event was observed showing its superior QoC.

The results of this section prove the reproducibility of the proposed NCS platform and the validity of the benchmarking methodology. In fact, a new platform could be reproduced and used for benchmarking. Furthermore, the proposed KPIs are able to highlight the different performances of the two computing systems and network interfaces.

5 CONCLUSIONS

In this paper, we presented a novel, reproducible, experimental platform for NCSes. Our system is designed to be easily reproducible due to low-cost, accessible hardware and open-source software components. Additionally, thanks to the experience acquired during the implementation of the platform and the joint expertise of control, computation, and communication domains, we propose a new NCS benchmarking methodology that enables reproducible experiments. The methodology defines the scenario, i.e. the experiment parameters, and the most relevant KPIs for the performance evaluation of the NCS platform. We evaluate the proposed platform and the validity of our benchmarking methodology reproducing the platform and evaluating it in different scenarios. Evaluation results prove the effectiveness of the proposed KPIs and the validity of the benchmarking methodology.

ACKNOWLEDGMENTS

This work was supported by the DFG Priority Programme 1914 Cyber-Physical Networking grant numbers KE1863/5-1, RA516/12-1, and CA595/71.

A CONTROL DERIVATIONS

A.1 Dynamical Model of the Robot

The nonlinear continuous-time *dynamical model of the plant*, i.e. of the TWIPR, which has been formally computed in [13] and [17], is

$$\forall t \in \mathbb{R}_{\geq 0},\ \dot{x}(t) = f\left(x(t), u(t)\right), \tag{5}$$

where, $f : \mathbb{R}^6 \rightarrow \mathbb{R}^6$, $\forall t \in \mathbb{R}_{\geq 0}$, the state variable vector $x(t)$ is

$$x(t) = \left[\Theta(t), \dot{\Theta}(t), \Phi(t), \dot{\Phi}(t), \gamma(t), \dot{\gamma}(t)\right]^\top, \tag{6}$$

and the control variable vector $u(t)$ is $u(t) = [u_l(t), u_r(t)]^\top$.

For our control purposes, however, since the controller is executed on a microprocessor that makes use of sampled data, a linear discrete-time model is required.

The desired equilibrium is defined by the pair $(\bar{x}, \bar{u})$, assuming that

$$\bar{x} = \left[0, 0, \bar{\Phi}, 0, \bar{\gamma}, 0\right]^\top, \tag{7}$$

where $\bar{\Phi} \in \mathbb{R}$ and $\bar{\gamma} \in \mathbb{R}$ are arbitrary, and $\bar{u} = [0, 0]^\top$. By definition of equilibrium of a dynamical system, $f(\bar{x}, \bar{u}) = \bar{x}$. By [14], if the function f is continuously differentiable with respect to its arguments, if we let $x(t) = \bar{x} + \delta x(t)$ and $u(t) = \bar{u} + \delta u(t)$, it is possible to linearize Eq. (5) around the equilibrium, so that the linearized system becomes

$$\delta\dot{x}(t) = A\delta x(t) + B\delta u(t), \tag{8}$$

where

$$A = \left.\frac{\partial f}{\partial x}\right|_{x=\bar{x}, u=\bar{u}},\ B = \left.\frac{\partial f}{\partial u}\right|_{x=\bar{x}, u=\bar{u}}. \tag{9}$$

A discrete-time equivalent of Eq. (8) is then necessary, since measurements and control inputs are in digital domain. By [2], by applying the *Forward Euler method* with a discretization step $T_s \in \mathbb{R}_{>0}$, Eq. (8) can be discretized and the resulting discrete-time linear system is

$$\xi(k+1) = A_d\xi(k) + B_d v(k), \tag{10}$$

where $A_d \in \mathbb{R}^{6\times 6}$, $B_d \in \mathbb{R}^{6\times 2}$,

$$\begin{aligned}\xi(k) &:= x(kT_s) = \\ &= \left[\Theta(kT_s), \dot{\Theta}(kT_s), \Phi(kT_s), \dot{\Phi}(kT_s), \gamma(kT_s), \dot{\gamma}(kT_s)\right]^\top,\end{aligned} \tag{11}$$

$\forall k \in \mathbb{N}$, and

$$v(k) = [u_l(kT_s), u_r(kT_s)]^\top, \tag{12}$$

$\forall k \in \mathbb{N}$. In particular, $A_d = I - AT_s$ and $B_d = BT_s$. These matrices and all the used parameters can be found in the public repository containing the open-source code of our implementation and measurements[1]. For what concerning sensors, measurements, and employed filters, the interested reader can refer to [17].

A.2 Theoretical Control Framework

With regards to the states in Eq. (11), $\forall k \in \mathbb{N}$, $\xi_r(k) \in \mathbb{R}^6$ is the state reference. Formally, a control logic computes an input $v(k) \in \mathbb{R}^2$ at each sampling time $k \in \mathbb{N}$, which drives the quantity $|\xi_r(k) - \xi(k)|$ as close to 0 as possible. In control theory, a controller is said to be designed in *closed loop* if, at every sampling time k, the control input is computed based on the current state of the system, i.e.

$$v(k) = f_v\left(\xi(k)\right), \tag{13}$$

where $f_v : \mathbb{R}^6 \mapsto \mathbb{R}^2$ is the so-called *control law*. However, as highlighted in Section 2.2 and Fig. 4, first, the controller does not have a real-time knowledge of the state of the system, and, second, the computed control input is actuated with a delay. Moreover, whenever packet loss occurs, the control system gets technically in *open-loop*, since no updated information about the state is available. Another challenge concerns the presence of a *time-varying sampling period*. In fact, as discussed in Sec. 2.2, the total delay of the benchmarking platform is not constant, i.e. in general, $\forall k_1, k_2 \in \mathbb{N}$, $t^{k_1}_{\text{A,W}} - t^{k_1}_{\text{S,R}} \neq t^{k_2}_{\text{A,W}} - t^{k_2}_{\text{S,R}}$. Given these considerations, a suitable control strategy is designed as follows.

First, the eventuality of packet loss is initially neglected. Since a time-varying sampling period adds unproductive complexity to the control problem, a constant sampling period framework is designed. After performing an extensive

evaluation of the platform, it was possible to define $\Delta_T \in \mathbb{R}_{>0}$ as the *overall delay upper-bound*, i.e.

$$\forall k \in \mathbb{N},\ t^k_{\mathrm{A,W}} - t^k_{\mathrm{S,R}} \leq \Delta_T. \tag{14}$$

The employed idea is to force the sampling period to be exactly Δ_T, by dilating the actuation time $d^k_{\mathrm{P,A}}$, so that, $\forall k \in \mathbb{N},\ t^k_{\mathrm{A,W}} - t^k_{\mathrm{S,R}} \approx \Delta_T$. The actuation time $d^k_{\mathrm{P,A}}$ is lower-bounded to $\underline{d}_{\mathrm{P,A}}$, i.e. $\forall k \in \mathbb{N},\ d^k_{\mathrm{P,A}} \geq \underline{d}_{\mathrm{P,A}}$, and can be forced to be

$$\forall k \in \mathbb{N},\ d^k_{\mathrm{P,A}} = \Delta_T - \left(t^k_{\mathrm{A,SRX}} - t^k_{\mathrm{S,R}}\right), \tag{15}$$

thus guaranteeing a constant sampling period $T_s = \Delta_T$. While applying Eq. (15), it is necessary to hold $\forall k \in \mathbb{N},\ d^k_{\mathrm{P,A}} \geq \underline{d}_{\mathrm{P,A}}$. This ensures that employing Eq. (15) dilates the processing time, thus resulting in a feasible approach.

According to the constant sampling period strategy,

$$\forall k \in \mathbb{N},\ t^k_{\mathrm{A,W}} - t^k_{\mathrm{S,R}} = t^{k+1}_{\mathrm{S,R}} - t^k_{\mathrm{S,R}} = \Delta_T.$$

This means that the actuated control input at $t \in \mathbb{R}_{>0}$ can be computed based only on the measurements at $t - \Delta_T$. With regards to Eqs. (11), (12), and (13), let $\check{\nu}(k)$, $\tilde{\nu}(k)$, and $\hat{\nu}(k)$ be defined, respectively, as, $\forall k \in \mathbb{N}$,

$$\check{\nu}(k) := f_\nu\left(x\left(k\Delta_T\right)\right), \tag{16}$$

$$\tilde{\nu}(k) := f_\nu\left(x\left((k-1)\Delta_T\right)\right) = \check{\nu}(k-1), \tag{17}$$

$$\hat{\nu}(k) := f_\nu\left(\hat{x}\left(k\Delta_T|(k-1)\Delta_T\right)\right). \tag{18}$$

Here, $\check{\nu}(k)$ is the ideal control input in the case the current state is available at the controller. On the other hand, $\tilde{\nu}(k)$ is the control input computed based on the last available state measurement. Finally, $\hat{\nu}(k)$ is the control input entailing a state prediction based on the last available measurement, i.e. $\hat{x}\left(k\Delta_T|(k-1)\Delta_T\right)$. Let's first assume that, $\forall k \in \mathbb{N}$, $\hat{x}\left(k\Delta_T|(k-1)\Delta_T\right)$ is computed based on the numerical integration of the continuous-time dynamics Eq. (5), i.e.

$$\hat{x}\left(k\Delta_T|(k-1)\Delta_T\right) = x\left((k-1)\Delta_T\right) + \int_{(k-1)\Delta_T}^{k\Delta_T} f\left(x(t), u\left((k-1)\Delta_T\right)\right) \mathrm{d}t, \tag{19}$$

performed at the remote controller during $d^k_{\mathrm{P,C}}$, under the following two assumptions:

Assumption 1. *$d^k_{P,C}$ needed to compute (19) guarantees that*

$$\Delta_T - (t^k_{A,SRX} - t^k_{S,R}) \geq \underline{d}_{P,A}.$$

Assumption 2. *(5) is an accurate model of the plant and any measurement noise is negligible.*

By Assumptions 1 and 2, the following trivially holds:

$$\hat{x}(k\Delta_T|(k-1)\Delta_T) \approx x(k\Delta_T). \tag{20}$$

This clearly implies that, $\forall k \in \mathbb{N}$,

$$\delta^p_\nu(k) := ||\check{\nu}(k) - \hat{\nu}(k)|| \ll ||\check{\nu}(k) - \tilde{\nu}(k)|| := \delta^+_\nu(k). \tag{21}$$

Under Assumption 1 and 2, $\hat{\nu}(k)$, computed as in (18) and (19), will be employed.

Thanks to this technique, the eventuality of *packet loss* can also be considered, in fact, it is possible to employ a similar strategy to guarantee robustness against this nonideality. Let $\bar{P}$ be the maximum number of consecutive packet dropouts. The controller does not know a priori whether a packet will be dropped or not. A feasible solution considers the controller to send, preventively and additionally to the current control input, a list of $\bar{P}$ inputs computed on the state predictions based solely on the measurements at time $(k-1)\Delta_T$. Formally, $\forall k \in \mathbb{N}$, the controller sends to the robot the control input vector

$$\hat{\boldsymbol{\nu}}(k) = \left[\hat{\nu}(k), \hat{\nu}(k+1), \ldots, \hat{\nu}(k+\bar{P})\right] \in \mathbb{R}^{2\times\bar{P}+1}, \tag{22}$$

where

$$\forall k \in \mathbb{N},\ \forall l \in \{1, \ldots, \bar{P}\},\ \hat{\nu}(k+l) = f_\nu\left(\hat{\hat{x}}(k+l)\right), \tag{23}$$

with

$$\forall k \in \mathbb{N},\ \forall l \in \{1, \ldots, \bar{P}\},$$

$$\hat{\hat{x}}(k+l) = \hat{\hat{x}}(k+l-1) + \int_{(k+l-1)\Delta_T}^{(k+l)\Delta_T} f\left(x(t), \hat{\nu}(k+l-1)\right) \mathrm{d}t, \tag{24}$$

and

$$\forall k \in \mathbb{N},\ \hat{\hat{x}}(k) = \hat{x}(k). \tag{25}$$

Formally, a packet is dropped at iteration k if

$$\exists t \in \left(t^k_{\mathrm{S,R}}, t^k_{\mathrm{S,R}} + \Delta_T\right) :\ t^k_{\mathrm{S,R}} + t_{\mathrm{TO}} \leq t < t^k_{\mathrm{A,NRX}}, \tag{26}$$

where t_{TO} is a given timeout measure. In order to ease the control formalism, a boolean variable is created, which stores the result of a packet dropout check. Formally, let, $\forall k \in \mathbb{N}$, $\vartheta(k) \in \{0, 1\}$ be evaluated according to the following

$$\exists t \in (t^k_{\mathrm{S,R}}, t^k_{\mathrm{S,R}} + \Delta_T) : \quad t^k_{\mathrm{S,R}} + t_{\mathrm{TO}} \leq t < t^k_{\mathrm{A,NRX}} \implies \vartheta(k) = 1, \tag{27}$$

else $\vartheta(k) = 0$. Let $\boldsymbol{\nu}^*(k) = \hat{\boldsymbol{\nu}}(k-l)$, $l \in \{1, \ldots, \bar{P}\}$, be the last available received control input, resulting from the occurrence of l consecutive packet dropouts. Formally,

$$\forall k \in \mathbb{N},\ \boldsymbol{\nu}^*(k) = \begin{cases} \hat{\boldsymbol{\nu}}(k) & \text{if } \vartheta(k) = 0 \\ \boldsymbol{\nu}^*(k-1) & \text{else.} \end{cases} \tag{28}$$

By these considerations, at every $k \in \mathbb{R}$, the robot applies the last available control input for that iteration, i.e.

$$\nu(k) = \left[\boldsymbol{\nu}^*(k)\right]_{\varpi(k)}, \tag{29}$$

where $\forall k \in \mathbb{N}$,

$$\varpi(k) = 1 + \min_{\substack{l \in \{0, \ldots, \bar{P}\}: \\ \vartheta(k-l)=0}} l. \tag{30}$$

Under Assumption 2, it is evident that the employed strategy guarantees that

$$\forall l \in \{1,\dots,\bar{P}\},\ ||\check{v}(k+l) - [\boldsymbol{\nu}^*(k)]_{l+1}|| \ll ||\check{v}(k+l) - \check{v}(k)||, \tag{31}$$

thus proving the state prediction of Eq. (22) is better than using the last available measurement as is.

A.2.1 Practical Control Implementation. However, in a practical implementation, Assumption 1 cannot be guaranteed, as solving a numerical integration problem requires a time which is incompatible with the constraint Δ_T. Instead of a state prediction based on Eq. (5), a state prediction based on the linear discrete-time model Eq. (11) is employed, which requires, under a computational point of view, only a matrix product. Thus, the following control input is computed and sent to the plant at each iteration k:

$$\forall k \in \mathbb{R},\ \tilde{\hat{\boldsymbol{\nu}}}(k) = \left[\tilde{\hat{v}}(k), \tilde{\hat{v}}(k+1), \dots, \tilde{\hat{v}}(k+\bar{P})\right] \in \mathbb{R}^{2\times(\bar{P}+1)}, \tag{32}$$

where

$$\forall k \in \mathbb{N},\ \forall l \in \{0,\dots,\bar{P}\},\ \tilde{\hat{v}}(k+l) = f_v\left(\tilde{\hat{x}}(k+l)\right), \tag{33}$$

with

$$\begin{aligned} &\forall k \in \mathbb{N},\ \forall l \in \{0,\dots,\bar{P}\}, \\ &\tilde{\hat{x}}(k+l) = A_d\tilde{\hat{x}}(k+l-1) + B_d\tilde{\hat{v}}(k+l-1), \end{aligned} \tag{34}$$

and

$$\forall k \in \mathbb{N},\ \tilde{\hat{x}}(k-1) = x(\Delta_T(k-1)), \tag{35}$$

which is the last available measurement at the controller at iteration k.

For the purpose of the current development, the employed control law is a discrete-time LQR, i.e. for an arbitrary $\mathbf{s} \in \mathbb{R}^6$,

$$f_v(\mathbf{s}) = \Re'\mathbf{s}, \tag{36}$$

where $\Re \in \mathbb{R}^{6\times2}$ is computed with the *Riccati* equation. Any interested reader can find a valid tutorial in [14, p. 170].

REFERENCES

[1] ACM. 2018. Artifact Review and Badging. https://www.acm.org/publications/policies/artifact-review-badging Visited Oct. 5, 2018.

[2] K. J. Åström and B. Wittenmark. 2013. *Computer-controlled systems: theory and design.* Courier Corporation.

[3] C. Bachhuber, S. Conrady, M. Schütz, and E. Steinbach. 2017. A Testbed for Vision-Based Networked Control Systems. In *Computer Vision Systems*. Springer International Publishing, Cham, 26–36.

[4] D. Baumann, F. Mager, H. Singh, M. Zimmerling, and S. Trimpe. 2018. Evaluating Low-Power Wireless Cyber-Physical Systems. *CoRR* abs/1804.09582 (2018). arXiv:1804.09582 http://arxiv.org/abs/1804.09582

[5] C. A. Boano, S. Duquennoy, A. Förster, O. Gnawali, R. Jacob, H. Kim, O. Landsiedel, R. Marfievici, L. Mottola, G. P. Picco, X. Vilajosana, T. Watteyne, and M. Zimmerling. 2018. IoTBench: Towards a Benchmark for Low-power Wireless Networking. In *1st Workshop on Benchmarking Cyber-Physical Networks and Systems*. IEEE, Oporto, Portugal.

[6] A. Chamaken and L. Litz. 2010. Joint design of control and communication in wireless networked control systems: A case study. In *Proceedings of the 2010 American Control Conference*. 1835–1840.

[7] M. Drew, X. Liu, A. Goldsmith, and K. Hedrick. 2005. Networked Control System Design over a Wireless LAN. In *Proceedings of the 44th IEEE Conference on Decision and Control*. 6704–6709.

[8] J. Eker, A. Cervin, and A. Hörjel. 2001. Distributed Wireless Control Using Bluetooth. *IFAC Proceedings Volumes* 34, 22 (2001), 360 – 365. IFAC Conference on New Technologies for Computer Control (NTCC 2001), Hong Kong, 19-22 November 2001.

[9] S. Gallenmüller, S. Günther, M. Leclaire, S. Zoppi, F. Molinari, R. Schöffauer, W. Kellerer, and G. Carle. 2018. Benchmarking Networked Control Systems. In *1st Workshop on Benchmarking Cyber-Physical Networks and Systems*. IEEE, Oporto, Portugal.

[10] LEGO Group. 2013. LEGO MINDSTORMS Education EV3 Core Set. https://education.lego.com/en-us/products/lego-mindstorms-education-ev3-core-set-/5003400 Visited Sept. 25, 2018.

[11] R. A. Gupta and M. Chow. 2010. Networked Control System: Overview and Research Trends. *IEEE Transactions on Industrial Electronics* 57, 7 (July 2010), 2527–2535.

[12] K. Kim and P. R. Kumar. 2012. CyberPhysical Systems: A Perspective at the Centennial. *Proc. IEEE* 100, Special Centennial Issue (May 2012), 1287–1308.

[13] Y. Kim, S. H. Kim, and Y. K. Kwak. 2005. Dynamic analysis of a nonholonomic two-wheeled inverted pendulum robot. *Journal of Intelligent and Robotic Systems* 44, 1 (2005), 25–46.

[14] M. Lalo and R. Scattolini. 2014. *Advanced and multivariable control*. Pitagoga edirice Bologna.

[15] C. Liu, F. Chen, J. Zhu, Z. Zhang, C. Zhang, C. Zhao, and T. Wang. 2017. Characteristic, Architecture, Technology, and Design Methodology of Cyber-Physical Systems. In *International Conference on Industrial IoT Technologies and Applications*. Springer, 230–246.

[16] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, and Y. Chen. 2016. Real-Time Wireless Sensor-Actuator Networks for Industrial Cyber-Physical Systems. *Proc. IEEE* 104, 5 (May 2016), 1013–1024.

[17] Z. Music, F. Molinari, O. Ayan, S. Zoppi, S. Gallenmüller, T. Seel, G. Carle, W. Kellerer, and J. Raisch. 2019. Design Of a Networked Controller For a Two-Wheeled Inverted Pendulum Robot. *Submitted for European Control Conference 2019*.

[18] S. W. Nawawi, M. N. Ahmad, and J. H. S. Osman. 2008. Real-time control of a two-wheeled inverted pendulum mobile robot. *World Academy of Science, Engineering and Technology* 39 (2008), 214–220.

[19] H. G. Nguyen, J. Morrell, K. D. Mullens, A. B. Burmeister, S. Miles, N. Farrington, K. M. Thomas, and D. W. Gage. 2004. Segway robotic mobility platform. In *Mobile Robots XVII*, Vol. 5609. International Society for Optics and Photonics, 207–221.

[20] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. 2010. Cyber-physical systems: The next computing revolution. In *Design Automation Conference*. 731–736.

[21] L. Zhang, H. Gao, and O. Kaynak. 2013. Network-Induced Constraints in Networked Control Systems – A Survey. *IEEE Transactions on Industrial Informatics* 9, 1 (Feb 2013), 403–416.