NCSbench: Reproducible Benchmarking Platform for Networked Control Systems

Samuele Zoppi*, Onur Ayan*, Fabio Molinari†, Zenit Music†, Sebastian Gallenmüller‡, Georg Carle‡, and Wolfgang Kellerer*
*Chair of Communication Networks, Department of Electrical and Computer Engineering, TUM, Germany
†Control Systems Group, Department of Electrical Engineering and Computer Science, TU Berlin, Germany
‡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 (NCS) gained notable attention thanks to their application to industrial operations. Experimental NCS require expertise from control, computation, and communication disciplines. This requirement, together with the fragmentation of implementation platforms and experimental investigations, represents a challenge for the reproducibility and comparison of research results. In this paper, we tackle this problem by proposing a novel NCS benchmarking methodology that aids the reproducibility of NCS experiments. Relying on a novel approach to model the architectural elements and the delays of NCS, the methodology defines the experiment parameters and the relevant Key Performance Indicators (KPIs) that need to be observed during its execution. Furthermore, we detail the implementation of the first reproducible benchmarking platform for NCS. The proposed platform is open-source and designed to be easily reproducible and extensible by anyone. Finally, we replicate and evaluate the platform following the proposed NCS benchmarking methodology. The experimental results evaluate and compare the KPIs during the execution of the platform in different benchmarking scenarios, proving the validity of the proposed benchmarking methodology.**

***Index Terms*—CPS, NCS, Open-source, Benchmarking, Reproducibility, KPI, Delay, Jitter, Packet loss, QoC.**

I. 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 (CPS) [1]. A CPS consists of the interconnection of sensors and actuators with a computation logic, together interacting over the same physical system. In fact, in CPS, the information acquired by sensors is processed and used to instruct actuators to perform specific actions.

A specific class of CPS, where networked sensors and actuators logically belong to the same automatic control system, is called Networked Control System (NCS) [2]. Over the last decade, NCS 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 NCS can be employed, for instance, in the closed-loop regulatory control of production machines [3]. For these reasons, a considerable amount of work is present in the literature that models and characterizes the performance of NCS in different operating conditions [4].

Despite a large number of results achieved by the control, computation, and networking research communities, the reproducibility and comparison of NCS experimental results are still obstacles to overcome. This is challenging for different reasons. 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. Therefore, the literature shows an extensive variety of hardware and software platforms, experimental methods, and measured performance metrics. As existing studies mainly focus on a few aspects of the entire system, results do not provide a complete characterization of the NCS. Additionally, the fragmentation of hardware, software, and expertise increases the difficulty of reproducing and comparing research results.

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

The rest of the paper is structured as follows. Sec. I-A provides a review of the state-of-the-art. Sec. II presents the novel NCS benchmarking methodology and Sec. III details the implementation of the proposed NCS benchmarking platform. Sec. IV evaluates the proposed methodology by benchmarking the open-source NCS platform in different scenarios. Sec. V concludes the paper.

A. Related Work

NCS have been extensively researched in the literature [4], [5]. A vast majority of the existing research work follows a theoretical approach. However, as stated by Lu et al. [6],

[1]The source code is available at: https://github.com/tum-lkn/NCSbench.

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 NCS conducting experimental studies is rather limited. Zhang et al. [7] and Chamaken et al. [8] 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 Kawka et al. [9] and Eker et al. [10] 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 [11]–[14]. Drew et al. [11] 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. [12] conduct an end-to-end latency analysis of a vision-based NCS. On the other hand, Baumann et al. [13] present measurement results from the case study of balancing an inverted pendulum over a multi-hop wireless network. Mager et al. [14] propose a reliable multi-hop wireless protocol that enables the remote control of multiple experimental inverted pendulums. However, in all these cases, authors do not address the issue of reproducibility of their platforms to allow repeatable experimental 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 [15]–[19]. Nethi et al. [15] present a platform for the emulation of NCS to enable their comparison in different scenarios. Wu et al. [16] develop FARE, a framework for benchmarking the reliability of CPS, not tackling, however, the specific aspects of NCS. Ding et al. [17] propose a framework for the design of fault-tolerant industrial NCS parametrizing the network and the control systems. The framework allows experiments with real NCS, but does not tackle the aspects of reproducibility and comparison. Boano et al. [18] and Gallenmüller et al. [19] 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 NCS.

II. Benchmarking Methodology

In this work, we present a novel benchmarking methodology for NCS experiments. The methodology relies on the combined knowledge of control, computation, and communication domains and the experience gained during the implementation of the proposed NCS platform of Sec. III. In fact, we not only extend the existing methodologies [18], [19] including experimental knowledge, but provide a novel approach to model the architectural elements (Sec. II-A) and the delays of NCS (Sec. II-B).

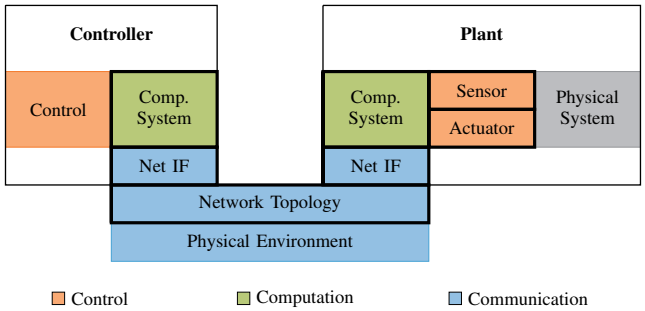


Fig. 1: Architecture of the NCS platform. The control, computation, and communication domains of CPS are represented. Every box is a component of the architecture, boxes surrounded by thick black contours represent hardware elements.

The purpose of the benchmarking methodology is to define the necessary amount of information in order to reproduce and evaluate experimental results using the NCS platform. Following the ACM's reproducibility terminology [20], 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.

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, as described in Sec. II-C, 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* and define in Sec. II-A. A whole series of different experiments and scenarios might be necessary to create a comprehensive report for the behavior of the NCS.

A. NCS Architecture and Scenario Description

We propose an architecture for experimental NCS as in Fig. 1. The architecture is composed of several software and hardware elements organized according to the three CPS domains [21]: control, computation, and communication.

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

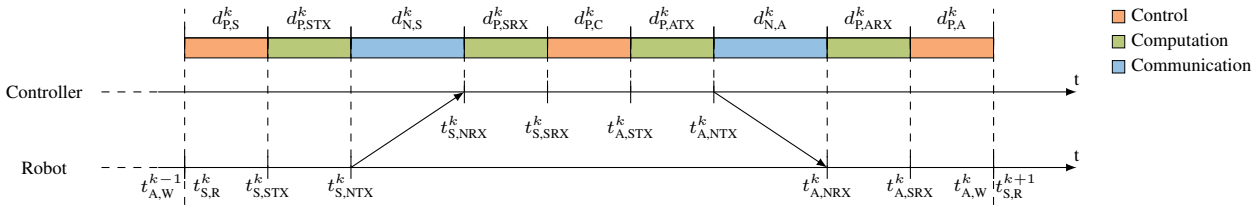


Fig. 2: Model of the timings of an NCS together with the *processing* (P) and *networking* (N) delays of the control, computation, and communication CPS domains.

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.

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.

To make the experiments and ultimately the benchmarks reproducible, it is key to document all relevant information of the NCS architecture in a *scenario description*. For every component of the NCS architecture of Fig. 1, software (algorithms) and hardware parameters must be specified to replicate the experiments.

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.

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* of the operating system, 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.

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.

B. Timings and Delay 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, every sampling period, the sensor measures the state of a plant and sends it to the controller, which computes and sends a command to the actuator that steers the plant.

The time evolution of the k-th sampling period is shown in Fig. 2. At time $t^k_{\text{S,R}}$, the sensor values (S) of the plant's sensors are read (R), handed over to the plant's network stack (STX) at $t^k_{\text{S,STX}}$ and transmitted over the communication network (NTX) at $t^k_{\text{S,NTX}}$. The controller's network interface receives the sensor data (NRX) at $t^k_{\text{S,NRX}}$ and its network stack delivers the packet to the control application (SRX) at time $t^k_{\text{S,SRX}}$. Afterwards, the controller calculates the actuation values for the actuators and hands over the actuation message (A) to the network stack at $t^k_{\text{A,STX}}$, which sends the packet over the network at $t^k_{\text{A,NTX}}$. Finally, at time $t^k_{\text{A,NRX}}$, the plant's network interface receives the actuation packet, and, at time $t^k_{\text{A,SRX}}$, its network stack delivers it to the actuator application, which applies (W) the commands to the actuators at $t^k_{\text{A,W}}$.

Thanks to the timing diagram shown in Fig. 2, 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* (P) 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 (N) 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 with simple techniques as busy waiting, the event of higher delays must be carefully taken into account using a proper control strategy.

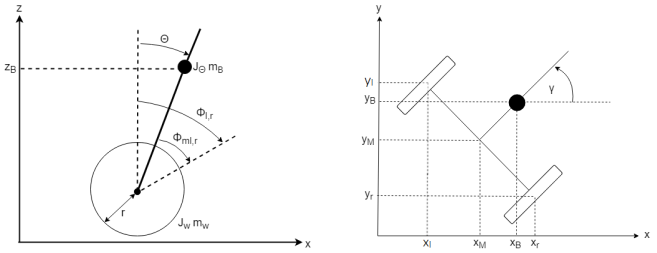
C. KPIs

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

A fundamental aspect that has emerged during the implementation and analysis of the proposed NCS platform is the role of time and *delays* in the system. Delays, i.e. the time needed for information exchange and processing on the robot and the controller, strongly influence the overall performance of the NCS. For this reason, an important part of the proposed KPIs is relative to time and delays. Delays can arise from control, computation, or communication, with lower delays offering a better service for the NCS. We assess the time KPIs by proposing a timings model of the NCS (Sec. II-B) and by measuring the individual delays presented in Fig. 2. For each measured delay, additional metrics can be obtained to parametrize a large class of NCS. By calculating the maximum, minimum, mean, and the probability density function of the measurement values over a time window, it is possible to characterize the stochastic fluctuations of the delays.

Moreover, *packet loss* additionally affects the operation of 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 the network and that the event of packet loss additionally arises whenever a packet experiences a delay larger than a specific *delay upper-bound*. As for the delays, also in this case, additional metrics can be calculated to better characterize the stochastic fluctuations of the packet loss.

On the other hand, *Quality of Control* (QoC) has an important role in the system. QoC is a metric that describes the performance of a control system and depends on the physical system and the control logic. QoC KPIs are functions that quantify the evolution of the physical system's state and the controller commands over a time window, i.e. the input and output information of the controller. One example is the LQR cost function used for the design of LQR control laws.



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

Fig. 3: Model of the two-wheeled Inverted Pendulum Robot.

III. NCS Platform Implementation

In this section, the implementation details of the proposed open-source[1] NCS benchmarking platform are presented following the architecture of Sec. II-A. Our platform aims at controlling, using a common IP network, a so-called *two-wheeled inverted pendulum robot* (TWIPR), which has a long tradition in literature [22], [23] and industry [24].

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.

This flexibility allows the proposed platform to be used for benchmarking of arbitrary NCS. In fact, the elements of the NCS architecture of Fig. 1 can be easily changed. Different physical plants can be built using Lego™, new control logics can be programmed in Python, arbitrary TCP/IP network interfaces can be connected, and the most popular computing systems and operating systems can be used.

The description of the implementation is organized as follows. In Sec. III-A, III-B, and III-C we detail the components of the NCS architecture for every CPS domain. Sec. III-D describes the measurement of the benchmarking KPIs.

Furthermore, we summarize in Tab. I the scenario description of a platform A and of a second reproduced platform B used in our evaluation. The scenario above only presents a minimal description of the most basic setup for our TWIPR performing the task of self-balancing. Additional parameters can be added to the scenario description for more complex scenarios. For instance, a TWIPR able to move would require the definition of the path and the surrounding environment. Tab. II summarizes the time and control KPIs measurement in our implementation.

A. 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 [26]. As represented in Fig. 3, the

Parameter	Description
Control Application SW	Python 3 open-source[1] code implementing the control logic of described in [25].
Control Physical System	*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. 4 connected via the AP TP-Link TL841ND.
Network Topology$_B$	Two-hop network shown in Fig. 4 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 GbE.
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. III-B.
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).

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

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, respectively, $\nu_l(t), \nu_r(t) \in [-\bar{V}, \bar{V}]$, 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. 3a, $\Phi_l(t)$ and $\Phi_r(t)$ indicate the rotation of the left and rights wheels with regards to z-axis. As in [23], $\Phi(t)$ describes the average rotation angle of the two wheels with regards to the z-axis, i.e. $\Phi(t) = 0.5 \cdot [\Phi_l(t) + \Phi_r(t)]$. The robot can move onto a 2D plane, i.e. the plane formed by axes x and y in Fig. 3b. The position of the body in the 2D plane at time t is $(x_M(t), y_M(t))$. Moreover, the orientation of the robot with regards to the x-axis at time t is denoted by $\gamma(t)$ and is called the yaw angle.

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. 3a, $\Theta(t)$ 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, 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 for the two motors. The plant is responsible for the periodic operation of the control loop and regularly triggers sensor readings every T_s. In a real implementation, delays vary according to the chosen software and hardware, which affect the sampling period, and packets can be lost due to large delays or network erasure. Both cases are taken into account by the control logic. Due to its complexity and space constraints, we omit the full derivation of the control law for our platform, which can be found in [25].

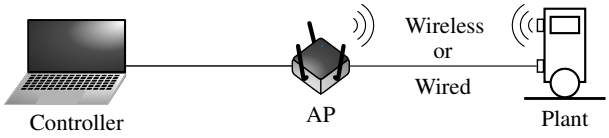


Fig. 4: Two-hop network topology of the implemented NCS benchmarking platform. The robot supports two network interfaces: an Ethernet adapter or a 2.4 GHz W-LAN dongle.

B. Computing Systems

Two different computing systems are deployed in our implementation: one for the controller and one for the robot.

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-layer communication protocols. For this reason, they deploy the widely spread TPC/IP network stack and the same application protocol. The application protocol 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.

KPI	Description
$d_{\mathrm{P,S}}$	Sensor readings on the robot.
$d_{\mathrm{P,C}}$	Calculation of controller's actuation commands.
$d_{\mathrm{P,A}}$	Execution of actuation commands on the robot.
d_{N}	Average one-way network delay including stack processing on robot and controller (STX, SRX).
$\hat{\Delta}_T - d_{\mathrm{P,A}}$	Robot round-trip delay.
$\hat{\Delta}_T$	Measured variable sampling period.
Σ_Θ	Total abs. deviation of the pitch angle.
Σ_Φ	Total abs. deviation of the wheels' rotation angle.
Σ_ν	Total abs. deviation of the average motors' effort.

Tab. II: Summary of time and control KPIs.

C. 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. 4. 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: 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.

D. KPIs Measurement

1) Time KPIs: We assess the time KPIs by measuring the individual delays presented in Fig. 2. 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 $d_{\mathrm{P,STX}}$, $d_{\mathrm{N,S}}$, $d_{\mathrm{P,SRX}}$, and $d_{\mathrm{N,A}}$ 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,

$$d_{\mathrm{N}} = 0.5 \cdot \left[t^k_{\mathrm{A,SRX}} - t^k_{\mathrm{S,STX}} \right]. \tag{1}$$

As KPI, we report each delay listed in Tab. II 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.

2) Control KPIs: 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.

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

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

$$\Sigma_\nu = 0.5 \cdot (\|\nu_l(kT_s)\| + \|\nu_r(kT_s)\|) \tag{4}$$

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. $\Theta_{ref} = \Phi_{ref} = \varnothing_{1xT_e}$. 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. The control KPIs are summarized in Tab. II

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 detailed in [25], a prediction is applied whenever a packet is not received within the delay upper-bound (Sec. II-C).

IV. 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.

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

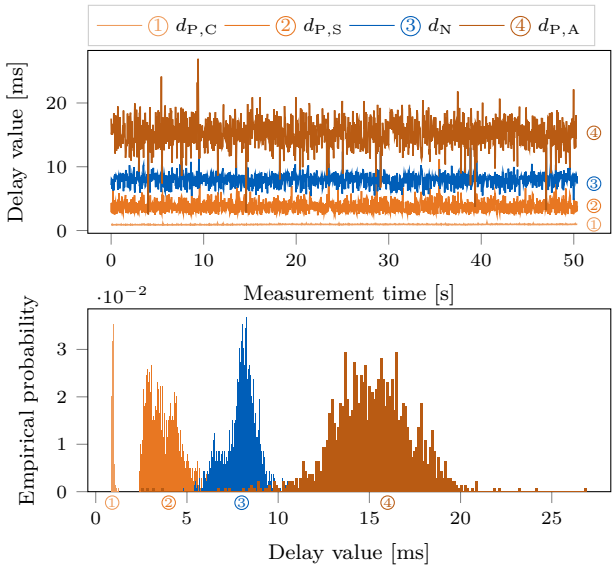


Fig. 5: Time evolution and empirical distribution of the delays of the controller, sensor, actuator, and network.

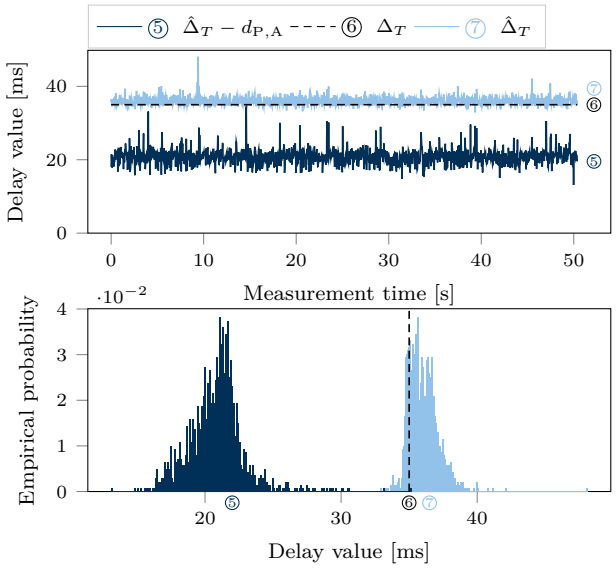


Fig. 6: Time evolution and empirical distribution of the round-trip delays and of the measured sampling period.

this way, for every experiment, an even number of samples is collected, and the KPIs can be correctly calculated and compared.

A. 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. I communicating over W-LAN.

Fig. 5 shows the time evolution and the histogram of the delays of the controller, sensor, actuator, and network defined

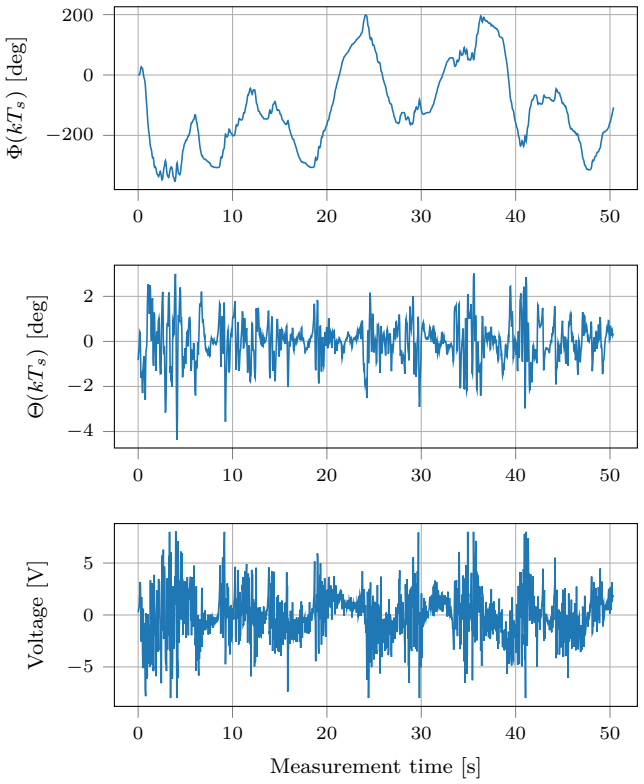


Fig. 7: Time evolution of the filtered pitch angle Θ, the filtered avg. rotation angle Φ, and the avg. applied voltage at the motors ν.

in Fig. 2. The sensor reading delay $d_{P,S}$ ② demonstrates a stable behavior with occasional outliers reaching up to 5.5 ms. Similarly, the controller delay $d_{P,C}$ 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 $d_{P,A}$ ④ shows an 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 $t_{A,SRX}$ is collected by the robot

application after receiving the actuation message, resulting in the delay $\hat{\Delta}_T - d_{\mathrm{P,A}}$ 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 $d_{\mathrm{P,A}}$ 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 $\Theta\left(kT_s\right)$ 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 $\Phi\left(kT_s\right)$, 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 compared to balancing the robot, and it is performed on a larger time scale.

B. 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 I. It consists of 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. III and IV summarize the benchmark KPIs resulting from the evaluation of the four scenarios. The time KPIs in Tab. II are presented as median with 95% confidence intervals, 1st and 3rd quartiles, and 99.9-th percentiles.

Tab. III shows different performances of the deployed computing systems and communication networks. In fact, the median values of $d_{\mathrm{P,C}}$ 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 $d_{\mathrm{P,S}}$; 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 $d_{\mathrm{P,A}}$ 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 scenarios 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.

Delay [ms]	Median $\pm 95\%$ C.I.	Q_1	Q_3	99.9%
A-wired				
$d_{\mathrm{P,C}}$	0.94 ± 0.002	0.91	0.97	1.07
$d_{\mathrm{P,S}}$	3.55 ± 0.038	3.04	4.24	5.41
d_{N}	4.38 ± 0.041	4.08	5.03	6.66
$d_{\mathrm{P,A}}$	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				
$d_{\mathrm{P,C}}$	0.95 ± 0.002	0.92	0.96	1.05
$d_{\mathrm{P,S}}$	3.64 ± 0.049	3.03	4.36	6.20
d_{N}	8.09 ± 0.053	7.54	8.54	10.88
$d_{\mathrm{P,A}}$	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				
$d_{\mathrm{P,C}}$	0.39 ± 0.001	0.38	0.39	0.45
$d_{\mathrm{P,S}}$	3.89 ± 0.034	3.55	4.49	5.73
d_{N}	4.61 ± 0.026	4.40	4.82	6.57
$d_{\mathrm{P,A}}$	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				
$d_{\mathrm{P,C}}$	0.37 ± 0.001	0.37	0.38	0.43
$d_{\mathrm{P,S}}$	3.84 ± 0.040	3.49	4.45	6.39
d_{N}	5.25 ± 0.055	4.85	6.29	8.74
$d_{\mathrm{P,A}}$	21.27 ± 0.126	19.49	22.38	24.84
$\hat{\Delta}_T$	36.32 ± 0.049	35.70	36.95	38.76

Tab. III: Time KPIs of the four evaluation scenarios.

	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

Tab. IV: Control KPIs of the four evaluation scenarios.

Tab. IV shows comparable values of QoC, for the two NCS 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.

V. Conclusions

In this paper, we presented a novel benchmarking methodology for NCS and a reproducible NCS experimental platform. The proposed benchmarking methodology enables reproducible experiments thanks to the experience acquired during the implementation of the platform and the joint expertise of control, computation, and communication domains. Relying on a novel approach to model the architectural elements and the delays of NCS, the methodology defines the scenario, i.e. the experiment parameters, and the most relevant KPIs for the performance evaluation of the platform. Following the NCS architecture and delays model, we implemented an open-source NCS experimental platform. The proposed platform is designed to be easily reproducible thanks to low-cost, accessible hardware and open-source software components. We evaluated the proposed platform and the validity of our benchmarking methodology reproducing the platform and evaluating it in different scenarios. The evaluation results prove the effectiveness of the proposed KPIs and the validity of the benchmarking methodology.

Acknowledgment

This work was supported by the DFG Priority Programme 1914 Cyber-Physical Networking grant numbers KE1863/5-1, RA516/12-1, and CA595/7-1. The authors would like to thank the anonymous reviewers for the valuable comments.

References

[1] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," in *Design Automation Conference*, 2010.

[2] K. Kim and P. R. Kumar, "Cyber–physical systems: A perspective at the centennial," *Proceedings of the IEEE Special Centennial Issue*, 2012.

[3] R. A. Gupta and M. Chow, "Networked control system: Overview and research trends," *IEEE Transactions on Industrial Electronics*, 2010.

[4] L. Zhang, H. Gao, and O. Kaynak, "Network-Induced Constraints in Networked Control Systems—A Survey," *IEEE Transactions on Industrial Informatics*, 2013.

[5] X. Zhang, Q. Han, and X. Yu, "Survey on Recent Advances in Networked Control Systems," *IEEE Transactions on Industrial Informatics*, 2016.

[6] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, and Y. Chen, "Real-time wireless sensor-actuator networks for industrial cyber-physical systems," *Proceedings of the IEEE*, 2016.

[7] Wei Zhang, M. S. Branicky, and S. M. Phillips, "Stability of Networked Control Systems," *IEEE Control Systems Magazine*, 2001.

[8] A. Chamaken and L. Litz, "Joint design of control and communication in wireless networked control systems: A case study," in *Proceedings of the 2010 American Control Conference*, 2010.

[9] P. A. Kawka and A. G. Alleyne, "Stability and Feedback Control of Wireless Networked Systems," in *Proceedings of the American Control Conference*, 2005.

[10] J. Eker, A. Cervin, and A. Hörjel, "Distributed Wireless Control Using Bluetooth," *IFAC Proceedings Volumes*, IFAC Conference on New Technologies for Computer Control, 2001.

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

[12] C. Bachhuber, S. Conrady, M. Schütz, and E. Steinbach, "A testbed for vision-based networked control systems," in *Computer Vision Systems*, 2017.

[13] D. Baumann, F. Mager, H. Singh, M. Zimmerling, and S. Trimpe, "Evaluating Low-Power Wireless Cyber-Physical Systems," in *IEEE Workshop on Benchmarking Cyber-Physical Networks and Systems*, 2018.

[14] F. Mager, D. Baumann, R. Jacob, L. Thiele, S. Trimpe, and M. Zimmerling, "Feedback Control Goes Wireless: Guaranteed Stability over Low-power Multi-hop Networks," in *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, 2019.

[15] S. Nethi, M. Pohjola, L. Eriksson, and R. Jantti, "Platform for Emulating Networked Control Systems in Laboratory Environments," in *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2007.

[16] L. Wu and G. Kaiser, "FARE: A Framework for Benchmarking Reliability of Cyber-Physical Systems," in *IEEE Long Island Systems, Applications and Technology Conference*, 2013.

[17] S. X. Ding, P. Zhang, S. Yin, and E. L. Ding, "An Integrated Design Framework of Fault-Tolerant Wireless Networked Control Systems for Industrial Automatic Control Applications," *IEEE Transactions on Industrial Informatics*, 2013.

[18] 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, "IoTBench: Towards a Benchmark for Low-power Wireless Networking," in *1st Workshop on Benchmarking Cyber-Physical Networks and Systems*, 2018.

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

[20] ACM. Artifact Review and Badging. Visited Nov. 11, 2019. [Online]. Available: https://www.acm.org/publications/policies/artifact-review-badging

[21] C. Liu, F. Chen, J. Zhu, Z. Zhang, C. Zhang, C. Zhao, and T. Wang, "Characteristic, Architecture, Technology, and Design Methodology of Cyber-Physical Systems," in *International Conference on Industrial IoT Technologies and Applications*, 2017.

[22] S. W. Nawawi, M. N. Ahmad, and J. H. S. Osman, "Real-time control of a two-wheeled inverted pendulum mobile robot," *World Academy of Science, Engineering and Technology*, 2008.

[23] Y. Kim, S. H. Kim, and Y. K. Kwak, "Dynamic analysis of a nonholonomic two-wheeled inverted pendulum robot," *Journal of Intelligent and Robotic Systems*, 2005.

[24] H. G. Nguyen, J. Morrell, K. D. Mullens, A. B. Burmeister, S. Miles, N. Farrington, K. M. Thomas, and D. W. Gage, "Segway robotic mobility platform," in *Mobile Robots XVII*, 2004.

[25] Z. Music, F. Molinari, S. Gallenmüller, O. Ayan, S. Zoppi, W. Kellerer, G. Carle, T. Seel, and J. Raisch, "Design Of a Networked Controller For a Two-Wheeled Inverted Pendulum Robot," in *8th IFAC Workshop on Distributed Estimation and Control in Networked Systems*, 2019.

[26] L. Group. LEGO MINDSTORMS Education EV3 Core Set. Visited Nov. 11, 2019. [Online]. Available: https://education.lego.com/en-us/products/lego-mindstorms-education-ev3-core-set-/5003400