POLITECNICO DI TORINO
Repository ISTITUZIONALE

MEbots: Integrating a RISC-V Virtual Platform with a Robotic Simulator for Energy-aware Design

Original
MEbots: Integrating a RISC-V Virtual Platform with a Robotic Simulator for Energy-aware Design / Pollo, G., Hamdi, M.A., Risso, M., Ruotolo, L., Furbatto, P., Isoldi, M., Chen, Y., Burrello, A., Macii, E., Poncino, M., Jahier Pagliari, D., Vinco, S.. - (2025). (ISLPED 2025 IEEE/ACM International Symposium on Low Power Electronics and Design Reykjavik (ISL) August 6 - 8, 2025) [10.1109/ISLPED65674.2025.11261755].

Availability:
This version is available at: 11583/3003697 since: 2025-10-06T13:24:36Z

Publisher:
IEEE

Published
DOI:10.1109/ISLPED65674.2025.11261755

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

26 November 2026

MEbots: Integrating a RISC-V Virtual Platform with a Robotic Simulator for Energy-aware Design

Giovanni Pollo*, Mohamed Amine Hamdi*, Matteo Risso*, Lorenzo Ruotolo*, Pietro Furbatto*, Matteo Isoldi*, Yukai Chen†, Alessio Burrello*, Enrico Macii*, Massimo Poncino*, Daniele Jahier Pagliari*, Sara Vinco*

*Dept. DAUIN, Politecnico di Torino, Italy, name.surname@polito.it

† IMEC, Leuven, Belgium, name.surname@imec.be

arXiv:2505.16682v1 [cs.RO] 22 May 2025

***Abstract*—Virtual Platforms (VPs) enable early software validation of autonomous systems' electronics, reducing costs and time-to-market. While many VPs support both functional and non-functional simulation (e.g., timing, power), they lack the capability of simulating the environment in which the system operates. In contrast, robotics simulators lack accurate timing and power features. This twofold shortcoming limits the effectiveness of the design flow, as the designer can not fully evaluate the features of the solution under development. This paper presents a novel, fully open-source framework bridging this gap by integrating a robotics simulator (Webots) with a VP for RISC-V-based systems (MESSY). The framework enables a holistic, mission-level, energy-aware co-simulation of electronics in their surrounding environment, streamlining the exploration of design configurations and advanced power management policies.**

***Index Terms*—simulation, drones, virtual platform, SystemC**

I. Introduction

Virtual Platforms (VPs) enable comprehensive system modeling and simulation before physical production [1] and are thus a crucial resource in the design of modern embedded systems, characterized by heterogeneity and tight integration with the physical environment.

State-of-the-art VPs cover system functionality together with non-functional aspects such as timing, power consumption, or thermal behavior [2]–[4]. However, their focus is mostly on software development, thus their scope is restricted to digital System-on-Chips (SoC), together with their auxiliary electronic components (sensors, actuators, batteries, etc.) [2], [3], [5]. In this perspective, the surrounding environment is normally considered a secondary aspect, either neglected or left as a "user-provided" input. Nonetheless, for robots and autonomous systems, evaluating the optimization of functional and non-functional aspects (e.g., power management policies) in relation to the environment in which they operate is fundamental. For example, the battery lifetime of an Unmanned Aerial Vehicle (UAV) can not be optimized without considering mission parameters such as cruise speed and elevation, its weight, and the accuracy of its autonomous navigation software [6] which strongly depend on the surrounding environment and the scope of the mission. On the other hand, robotics simulators [7]–[9] are insufficient to address this problem. They are capable of modeling complete 3D environments, as well as robots' sensors, actuators, and digital computations, but they are restricted to a high-level behavioral simulation of the electronics subsystem, thus neglecting precise modeling of the timing or power/energy consumption of the digital subsystem.

In this work, we propose to overcome this twofold shortcoming by proposing **MEbots**, a tool that connects a comprehensive digital VP capable of functional and non-functional modeling with a full-fledged robotic simulator [7]. The choice of VP fell on **ME**SSY [3], which extends the RISC-V Instruction Set Simulator (ISS) GVSoC [10] with the SystemC-AMS simulation of external components (such as virtual sensors/actuators) and power-related devices (batteries, harvesters, DC/DC converters, etc.) [11]. For the robotics simulator, we chose We**bots** [7], a popular software capable of modeling dynamic objects such as robots and drones in complex 3D environments. Their integration in a holistic co-simulation framework allows to combine detailed timing and power estimation with the simulation of flight dynamics as an effect of mechanical and environmental factors.

To prove the effectiveness of the proposed solution, we applied MEbots to the simulation of a Crazyflie nano-UAV [12] in the context of a gate-traversal application [13], validating and optimizing the power/energy behaviour of the system. This analysis proves that MEbots enables: 1) a *Design Space Exploration (DSE) process* aimed at selecting HW and SW configurations that optimize specific non-functional metrics (e.g., battery lifetime) before the creation of a physical prototype; 2) the *study and optimization of context-dependent power management policies*, that require both the availability of detailed information on the power state of the system and inputs from the external environment.

The rest of the paper is organized as follows: Section II provides essential background and overviews of related works; Section III details our methodology; Section IV presents the experimental results, and Section V concludes the paper. The complete code of MEbots can be found at https://github.com/eml-eda/messy [14].

This work has received funding from the Key Digital Technologies Joint Undertaking (KDT-JU) under grant agreement No 101095947 and grant agreement No 101112274. The JU receives support from the European Union's Horizon Europe research and innovation program.

This publication is part of the project PNRR-NGEU which has received funding from the MUR – DM 117/2023.

©2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Accepted for publications at ISLPED 2025 IEEE/ACM International Symposium on Low Power Electronics and Design.

TABLE I
RELATED WORKS: VPS (TOP) AND ROBOTIC SIMULATORS (BOTTOM)

	RISC-V/ARM support	Off-chip components	Power modeling	Environment & flight	License
QEMU	✓	✗	✗	✗	GPLv2
Renode	✓	✓	✗	✗	MIT
gem5	✓	✗	≈	✗	BSD
Spike	✓	✗	✗	✗	BSD-3
RISC-V-TLM	✓	✗	✗	✗	GPLv3
GVSoC	✓	✗	✓	✗	Apache 2.0
MESSY	✓	✓	✓	✗	Apache 2.0
Webots	✗	✓	≈	✓	Apache 2.0
Gazebo	✗	✓	≈	✓	Apache 2.0
IsaacSim	≈ (HIL)	✓	✗	✓	NVIDIA EULA
AirSim	≈ (HIL)	✓	✗	✓	MIT
MEbots	✓	✓	✓	✓	Apache 2.0

II. RELATED WORKS

A. SoC Virtual Platforms

VPs of digital SoCs have been developed for decades [15] and differ by level of accuracy and focus.

QEMU [16], [17] is an open-source, versatile emulator and virtualizer of a target architecture that supports a wide range of instruction sets (e.g., ARM, RISC-V) with a focus on functional simulation. Renode [18], [19] is designed to simulate complete electronic systems with RISC-V support; besides functionality, it supports basic timing modeling, yet with low accuracy (minimum resolution of $1\mu s$ and CPU performance modelled only with the average Million of Instructions per Second - MIPS), but lacks power/energy (and external environment) modeling entirely. Gem5 [20], [21] is highly regarded for its flexibility and comprehensive support for multiple ISAs, including RISC-V. It offers a variety of CPU models and supports detailed hardware modeling. While supporting accurate timing simulation and state- or equation-based power models, those are limited only to digital SoC modules, without consideration of external components such as batteries and actuators (e.g. motors).

In addition to these multi-platform solutions, multiple VPs have been recently developed specifically for the RISC-V ecosystem. Spike [22], [23] is primarily designed for functional simulation and validation without providing performance assessments, nor any kind of power/energy estimation. RISC-V-TLM [24], [25] provides a simple and extensible simulation that abstracts communication details and includes only essential peripherals, enabling early virtual prototyping. However, its simplicity also comes with limitations, as it may not fully capture the complexity of real-world hardware interactions. GVSoC [10] is an open-source event-driven VP specifically designed for PULP systems [26], with accurate timing modeling and support for multi-core clusters, multi-level memory hierarchies, and various I/O peripherals and accelerators. LUT-based power models can be defined in GVSoC, but they are limited to SoC components only, meaning the power behavior of external devices cannot be modeled.

MESSY [3] is a recently proposed open-source framework that wraps a timing-accurate VP (currently GVSoC) into a top-level system description (in SystemC-AMS) to support the modeling of power behavior of each component (including power consumption, generation, conversion, and storage). However, MESSY does not support the simulation of environmental or mechanical aspects of the system.

B. Robotic Simulators

3D world simulators are essential for the development of complex autonomous systems, as they enable the validation and optimization of software components that determine how the system engages with its surroundings, e.g., training and testing of machine learning models for perception, the validation of control policies for navigation, obstacle avoidance, etc [13]. Webots is a popular simulator, offering accurate physics modeling capabilities for robots and environmental objects, as well as a vast library of pre-defined components and sensors/actuators (e.g., proximity sensors, cameras) [7], [27]. Gazebo [8], [27] is widely regarded as the most popular open-source alternative to Webots, and it allows to choose from different physics engines for simulation. IsaacSim [9], [28] and AirSim [29], [30] allow for detailed particle physics, to closely mirror real-world dynamics.

While all these tools allow the definition of custom controllers for the robots, in most of them the simulation is entirely behavioral, i.e., it completely neglects the detailed architecture of the HW on which the controller SW will be eventually deployed. This implies that: i) in most cases, it is not possible to validate the *actual code* that will be deployed on the robot; ii) the controller reacts instantaneously, or its timing is modeled at a very high level (i.e., with a fixed, user-defined execution delay); iii) other non-functional properties, in particular power/energy behavior, either are completely ignored (e.g., IsaacSim and AirSim) or can be described with some limitations (e.g., Gazebo imposes limitations by supporting only a single battery per model [31], while Webots restricts power consumption modeling to motor components [32]). Notably, IsaacSim and AirSim provide some degree of hardware integration through hardware-in-the-loop (HIL) simulations, allowing seamless interaction with physical systems for more comprehensive testing. However, their hardware support remains limited (e.g, IsaacSim primarily targets the Jetson series [33]) and does not account for alternative architectures such as the RISC-V based ones.

In this work, we chose Webots as a starting point to benefit from its wide library of pre-defined components, and its good balance between ease of use, simulation fidelity and resource efficiency. However, our methodology is, for the most part, orthogonal to the selected world simulator and can be extended to any simulator previously described.

III. METHODOLOGY

A. General Framework Architecture

Fig. 1 shows a high-level architectural diagram of our simulation framework, which connects a full-system VP (MESSY) with a robotic simulator (Webots). The figure highlights in blue the components involved in the connection.

The proposed methodology is orthogonal to the specific tools: any VP capable of modeling virtual sensors and actuators (i.e., components that interact with the external "environment") can be instrumented as described in the following, and

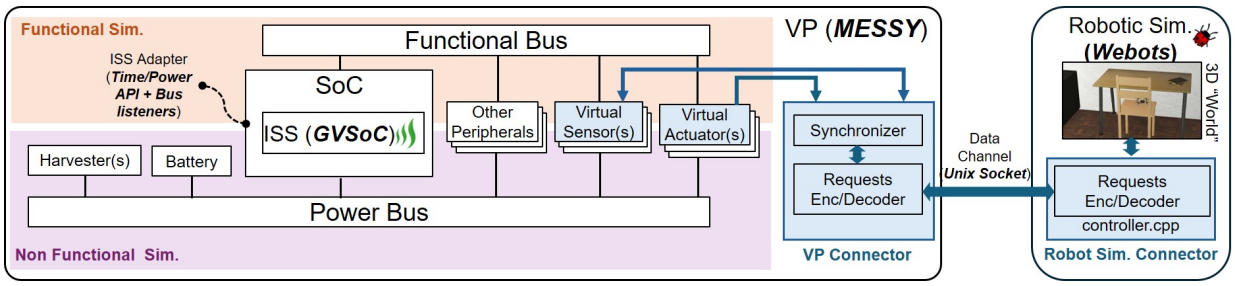

Fig. 1. MEbots architecture: the blue boxes highlight the components involved in the connection between a HW-SW VP (MESSY with the GVSoC ISS, left) with a robotic simulator (Webots, right).

Webots could be replaced by one of the alternatives in Table I. This section sketches the main idea behind the construction of MEbots, while details related to the choice of MESSY and Webots will be discussed in the next section.

a) Rationale: The link between any SoC VP and a robotic simulator is at the level of sensors and actuators: the sensors must indeed collect data from the environment (e.g., images) to provide input to the computing units, and the actuators must put into practice the control strategies elaborated by SW. This is where the co-simulation can be built by adding a *Connector*, in charge of implementing data exchange and time synchronization between the two frameworks.

b) Role of the connector: The role of the Connector is twofold, as it is in charge of implementing both data exchange and time synchronization. On the VP side, the Connector interacts with a subset of the virtual sensors and actuators. Typical examples are cameras, depth sensors (that receive data coming from the world modeled by the robot simulator), or motor controllers (that influence the robot's movements in said world). Whenever the virtual sensors/actuators receive commands from software and must interact with the environment, their request is forwarded to the Connector, that i) *synchronizes* time between the simulators before ii) *encoding and sending* the request over a data channel. On the robot simulator side, the Connector decodes such incoming request, and performs the appropriate action on the environment (e.g., capturing a picture, changing the motors settings). If the request involves a return value (e.g., the picture), the latter is sent back over an inverse path, after an additional synchronization step.

c) Time synchronization: Time synchronization is crucial to ensure that the two simulations proceed in lock-step. Robot simulators usually implement a fixed-timestep physics engine, with a coarse temporal granularity (in the order of ms). The implementation of the *Synchronizer* depends on the selected VP's timing-awareness, which is a design choice influenced by use-case needs in terms of simulation accuracy vs speed. For example, a purely functional VP can be synchronized by delaying each of its requests to the robot simulator by one time-step (i.e., the minimum non-instantaneous delay possible in this scenario), while for a more timing-accurate VP, the time associated with each virtual sensor request can be aligned to the next closest robot simulator timestep. Since GVSoC supports close to cycle-accurate execution, our implementation follows the second approach, as we advance the robot simulator by an amount of time that matches the software's progress, rounding to the nearest simulator timestep.

B. MEbots Implementation: MESSY and Webots

1) MESSY for HW-SW virtual prototyping: MESSY is primarily implemented in SystemC/SystemC-AMS, and focuses on modelling the power and energy behaviour of a complete embedded system, including a SoC, external peripherals, batteries, harvesters and power converters. A simplified scheme of the MESSY architecture is shown on the left of Fig. 1. MESSY simulation relies on a *functional bus*, implementing an AXI-like protocol handling functionality (i.e., exchange of data) and timing, and a *power bus*, modeling power flows between components through voltage and current ports, e.g., managing the distribution of power from a battery (or a harvester) to multiple loads (SoC, peripherals, etc) through converters.

MESSY is used to instantiate functional and power models of the system components, that can be either sourced from the tool's library or defined ad hoc. It also allows to integrate an external ISS through a customizable *Adapter*, to achieve accurate HW-SW prototyping. The current version supports GVSoC [10], an ISS for SoCs of the PULP family, popular in ultra-low-power robotics applications [6]. Through the Adapter, GVSoC uses its APIs to advance the SoC simulation in time as needed and to forward the corresponding SoC consumption to the power bus. Moreover, I/O requests to external devices initiated by the PULP SoC are intercepted by a set of listeners, and forwarded to MESSY's virtual peripheral models. We refer readers to [3] for details.

2) Webots for drone simulation: Together with the environment and the robotic platform, Webots allows the definition of custom robot *controllers*, managing exchanges of sensors and actuators data and robot operation. On the Webots side, the primary required changes are focused on enabling the interception of commands from MESSY and executing corresponding actions. Thus, these changes involve modifying the `controller.cpp` file. Specifically, we implement a controller that waits until it receives a new command via the communication channel. When that happens, the command is parsed to extract the necessary arguments and parameters. The associated logic is then executed to perform the specified action (e.g. take a picture and forward it to MESSY). This approach leverages an opcode-like structure, where each command is linked to a specific action, enabling precise control

over the Virtual World and its objects through MESSY, while also allowing simple extensibility by just adding a new opcode.

3) Connector implementation: To enable the communication between the two tools, we implemented the above mentioned connector using UNIX sockets. To ensure standardized communication, we chose to transmit JSON-like packets using a C++ library [34]. While this approach slightly increases communication overhead, it provides a structured and easily extensible format. Each packet includes the command opcode and, when necessary, an accompanying payload.

In this setup, MESSY operates as the socket client, while Webots serves as the socket server. This architecture is designed to support future scalability by enabling multiple MESSY instances to connect to Webots. Such a setup reflects real-world scenarios where multiple embedded systems, each with its own structure and power model, collaborate to control a single (or multiple) robotic systems.

Figure 2 illustrates an example of the communication process, retrieving data from a virtual camera in the 3D environment and adjusting motor speeds based on the output of a neural network running in GVSoC. At the start of the simulation, Webots initializes a socket and MESSY connects to it as client. Then, the simulation proceeds in sync.

The flow begins with GVSoC requesting an image from the Camera Module (1). This request, transmitted over MESSY's functional bus, prompts the Camera Module to generate a `GET_DATA` opcode and send it to the Encoder/Decoder ((2)). This automatically triggers a synchronization action, which advances the Webots simulation (if needed), until the first timestep greater than the current MESSY time. The Encoder then processes the command, formats it into a JSON-compliant packet, and transmits it through the UNIX socket ((3)). On the Webots side, the incoming packet is intercepted, and the corresponding action is executed ((4)). In this case, the 3D World captures an image, encapsulates it into a new JSON packet, and transmits it back to MESSY. The returned data payload is directed to the module that initiated the request ((5)). In this example, the ISS is notified of the image's readiness by polling a status register ((6)), although interrupt-based communication is also supported. The ISS then simulates the neural network execution on the received image ((7)), completing the inference task. For subsequent operations (i.e. setting the speed of the motors), the workflow repeats with a different opcode, following a similar sequence ((8), (9)). In (9), similarly to (2) (and in any interaction between the two simulators, all of which go through the connector), timesteps are re-synchronized.

This modular and consistent communication pattern ensures smooth integration and synchronization between MESSY and Webots, regardless of the operation being performed.

A key feature of MEbots is its scalable opcode structure, designed for efficient expansion and maintainability. The whole simulation is controlled by a configuration file (also in JSON format) that not only defines the system architecture, including details about each module's sensors, actuators, and power consumption, but also specifies the unique opcodes assigned to each module. Using this configuration file, a code generation mechanism automatically produces a structured code skeleton already implementing the MESSY/Webots connection. Developers only need to extend this skeleton by implementing the necessary logic for each module.

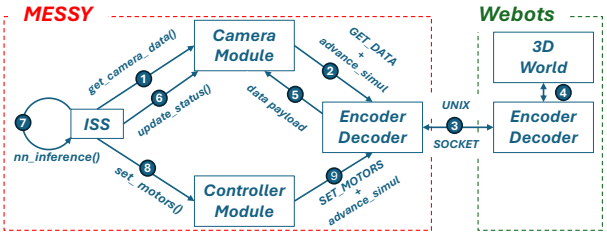

Fig. 2. Example of MESSY and Webots exchanging messages.

IV. Case Study: Drone Gate Traversal

To validate the proposed methodology, we configured MEbots to reproduce a Crazyflie nano-UAV in a gate-traversing scenario, described in [13].

A. Drone modeling

The drone has been modeled in both its functional and power aspects within MESSY, accurately modelling the camera, the motors, the battery and the SoC. As camera, we selected a Himax HM01B0 integrated into the drone's *AI Deck* [35], that captures monochrome images at a resolution of 320x320 pixels. Within MESSY, the camera is implemented as a virtual peripheral with a dedicated contiguous memory area capable of storing the full 320x320-pixel image. Datasheet specifications were used to derive current consumption in the active state (1.75 mA operating at 60 Frames Per Second, FPS) and idle state (0.14 mA, typical standby current under worst-case conditions). In this case study, GVSoC is used to simulate the GAP8 processor [36] on board of the AI Deck.

Motors plays a critical role in power consumption analysis, as they account for nearly 95% of the total power usage [13]. To achieve an accurate representation, we developed a custom power model that considers two key components. P_{hover} is the power required to maintain the drone in a hovering state: it factors in the mass m of the complete system (including the drone body, AI deck, and battery) and it incorporates several datasheet parameters such as the propeller pitch, revolutions per second (rps), and air density. P_{propel} instead focuses on the power required for horizontal movement at a steady cruising speed, and it takes into account the total system mass m, the propulsion system's efficiency η, and the cruising speed v. By combining these components, the model provides a simple but realistic estimate of the drone's power consumption.

We selected four different batteries to conduct a comprehensive exploration of battery capacity and weight impact on drone performance: the standard Bitcraze battery (250mAh, 7.1g), a UFX battery (250mAh, 5.8g) [37], a Cyclone battery (300mAh, 8.1g) [38], and a Lipol battery (350mAh, 9.2g) [39]. These batteries range from a slightly lighter alternative to the standard Bitcraze battery to a larger battery offering 40% more capacity but at a significant weight increase. Batteries have been modeled by extrapolating information from datasheets

to build their circuit equivalent models. Additionally, we incorporated the necessary DC/DC converters.

B. Gate traversal software

We built two distinct Webots simulation environments: an easy one, featuring a gate positioned almost directly in front of the drone, and a more challenging one, where the gate is near the edge of the drone's camera Field of View (FoV). This significantly increases the mission's difficulty, as the drone must rely on precise speed control and well-tailored logic to successfully navigate through the gate. The drone is 9.2cm×9.2cm×2.9cm, and each gate is a square with an internal opening of 40cm×40 cm. The gate frame has outer dimensions of 60cm×60cm, providing a 20cm wide and 3cm thick border. The gates are at a fixed height of 83cm above the ground, ensuring consistency across tests.

To control the drone, we employed a Convolutional Neural Network (CNN), that processes grayscale camera images downsampled to 168×168 pixels. The CNN architecture is taken from [13], modified to only use camera images and no Time of Flight (ToF). It consists of three sequential blocks, each containing Convolutional Layers, Batch Normalization, and ReLU6 activation. The number of channels increases progressively from 1 to 32. A final Fully Connected layer outputs the predicted yaw rate. We trained it using the open-source dataset [40] following the established data pre-processing used in [13]. We first trained for 100 epochs with a batch size of 256. The chosen optimizer is Adam [41], with an initial learning rate of 0.001 and a weight decay of 0.0001. Subsequently, we applied quantization-aware fine-tuning for 50 epochs using the open-source PLiNIO library, reducing both weights and activations to int8 precision [42]. Finally, the model has been deployed to GVSoC using the MATCH compiler [43].

In all experiments, the drone started from a predefined position, ascended to an altitude of 1 meter, and then cruised toward the direction indicated by the CNN at a specific speed, set to $v_{easy} \in \{0.5km/h, 1km/h, 1.5km/h\}$ in the easy scenario, and to $v_{hard} \in \{0.1km/h, 0.2km/h, 0.3km/h, adaptive\}$ in the hard scenario (as will be elaborated later on). After successfully passing through the gate, or when the battery SoC reaches a predefined threshold (10%), the drone lands.

V. Experimental Results

A. Battery choice exploration

The first batch of experiments focused on the simple gate-traversing scenario to evaluate the effectiveness of different batteries. Each speed scenario was tested with the four different batteries, and the results of these experiments are illustrated in Figure 3, where the x-axis represents the cruising speed of the drone, and the y-axis indicates the consumed state of charge (SoC) at the end of the simulation.

We first isolated the impact of sole battery capacity, by modeling all batteries assuming they had the same weight as the heaviest one. As expected, the smallest-capacity battery consumed the highest amount of charge, and the consumed charge showed a clear proportionality to the battery capacity,

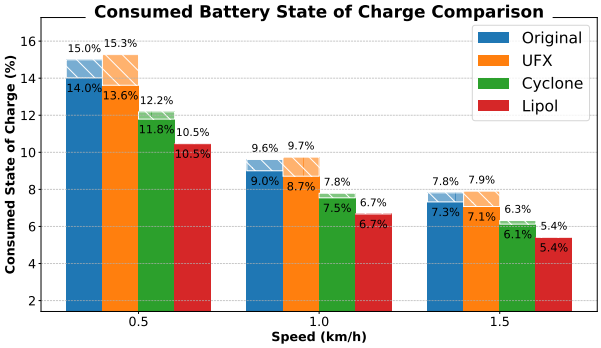

Fig. 3. Consumed State of Charge at different cruising speed and with different batteries when considering actual battery weight (full color) or the weight of the heaviest battery (shadows), to isolate the impact of capacity.

as reflected by the total bar heights (fully colored part plus shadowed part). For instance, the Cyclone battery has a capacity approximately $\sim 15\%$ smaller than the Lipol battery, and this is reflected in a higher SoC usage, falling within the range of $[16-17]\%$. The slight mismatch w.r.t. the expected proportional difference is due to non-linear battery dynamics and secondary effects (parasitic discharges, aging, etc).

When accounting for the actual weight of each battery (full-color bars), the results show a reduction in the consumed SoC, even if the trend is not modified (a lower total weight implies a lower P_{hover} and P_{propel} and thus a lower ΔSoC). A noteworthy comparison arises between the original battery and the UFX, both of which have identical capacity (250 mAh) but different weight (the original battery is 1.3g heavier compared to the UFX). Across all tested speeds, the latter consistently consumed less SoC, with savings of 3%, 3.5%, and 2.8% at $0.5km/h$, $1km/h$, and $1.5km/h$, respectively. These savings translate into a potential increase in flight time. To assess this increase, we first validated Bitcraze's claimed 7-minutes flight duration using our framework with the original battery, obtaining a flight time of around 6 minutes and 50 seconds. With this baseline, the UFX battery could extend the flight time by up to 15 seconds (4%), thanks to the sole weight reduction. Alternatively, by allowing for a slight increase in the battery pack's weight, we can achieve a significant 33% increase in battery life, at the price of a small reduction in the maximum recommended payload weight. This translates into an additional flight time of around 2 minutes and 30 seconds, which at a forward cruising speed of $1.5km/h$ translates into a covered distance of 62m. Overall, this experiment shows the importance of modelling flight dynamics (e.g. dependence on weight) to precisely estimate power/energy evolution in these systems, as enabled by MEbots.

B. Drone Gate-Traversing speed exploration

In drone racing, a key objective is to pass through gates in the shortest possible time, as shorter traversal times correlate with reduced battery consumption, as less energy is required to complete the objective. Using the challenging environment described above, we demonstrate how MEbots can be used to evaluate different flying strategies in terms of total battery consumed and task accomplishment. Notably, relying solely on Webots would not allow for an accurate estimation of total

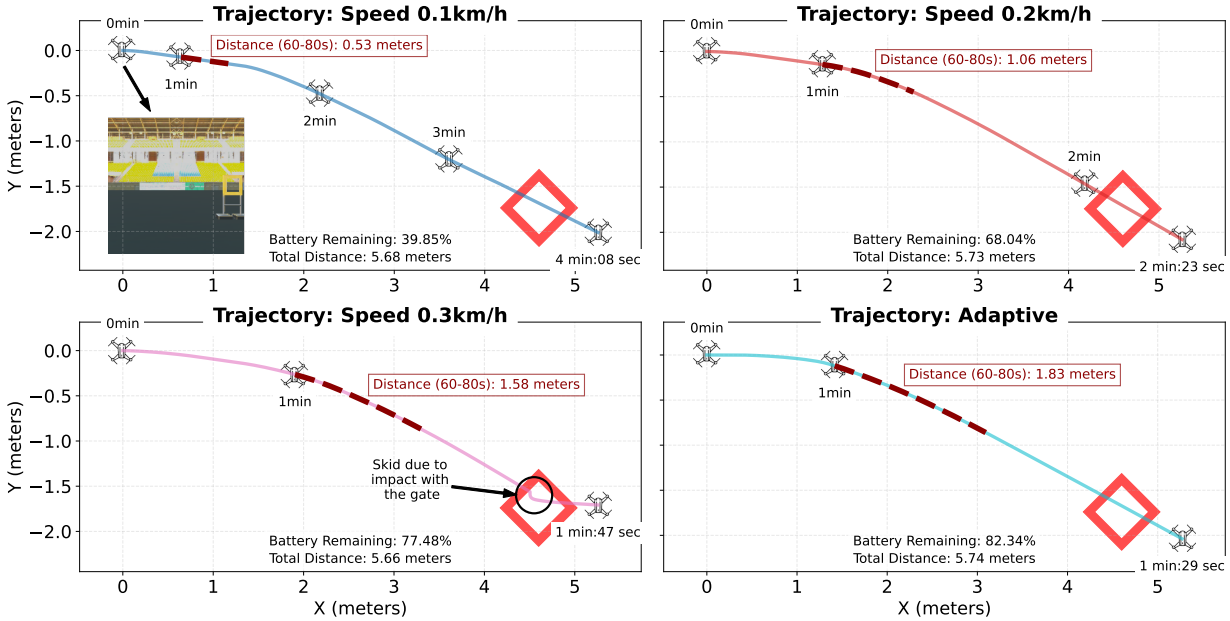

Fig. 4. Trajectory comparison with standard Bitcraze's battery at different fixed and adaptive speeds.

energy consumption, as the traversal time also depends on the accuracy of the CNN's prediction and on its reaction latency (both of which are precisely simulated in the VP).

The initial view of the drone is shown in the top-left subplot of Figure 4. The figure presents four subplots, each comparing different flying strategies: three with constant cruising speeds of $0.1km/h$, $0.2km/h$, $0.3km/h$, and one employing an *adaptive* speed control, all with the default battery.

At constant speeds of $0.1km/h$ and $0.2km/h$, the drone successfully enters the gate without issues. However, the time required to reach the gate is substantial, leading to lower remaining battery life at the simulation's end. The total time for $0.2km/h$ is not exactly half that of $0.1km/h$ due to the additional time required for takeoff, landing, and acceleration to cruising speed. To confirm that cruising speeds were correctly implemented, we measured the distance covered by the drone during a portion of the path at quasi-cruising speed; the distance covered in 20 seconds at $v = 0.2km/h \equiv 0.056m/s$ is almost 1.1 meters, double that at $v = 0.1km/h$.

Among the constant speeds, $0.3km/h$ is the fastest while still enabling quasi-successful gate entry. As visible in the trajectory, the drone exhibits a minor skid due to an impact on the gate structure when traversing it, likely caused by the higher speed. Despite this, the drone re-stabilizes and traverses successfully. This establishes $0.3km/h$ as the upper limit for constant cruising speed. For completeness, we also tested higher speeds of $0.4km/h$ and $0.5km/h$, but in these cases, the drone failed to enter the gate, and we omit the corresponding plots for clarity.

The adaptive flying strategy, instead, implements a simple real-time tuning mechanism of the drone's speed based on its alignment with the gate. Specifically, the yaw rate predicted by the CNN is a value in $[-1, 1]$, and can be used to estimate how centered is the gate within the drone's field of view (lower absolute values correspond to a more centered gate). Thus, our adaptive policy averages the last 10 CNN predictions, and sets the drone speed to a constant 1 km/h when $\|\text{yaw}_{avg}\| \leq 0.3$, while decreasing it by 0.05 km/h, when $\|\text{yaw}_{avg}\| > 0.3$. Using this adaptive technique, the required time to reach the gate was reduced to 1 minute and 29 seconds, representing improvements of 64%, 38%, and 17% compared to constant speeds of $0.1km/h$, $0.2km/h$, and $0.3km/h$, respectively. Additionally, the adaptive approach resulted in significant energy savings, with the remaining SoC increasing from 39.85% in the worst-case scenario to 82.34%, with an impressive 42.5% improvement. Despite the dynamic speed adjustments, the total distance covered was nearly identical to that of the constant-speed cases, showing only a 1%, 0.17%, and 1.4% increase compared to $0.1km/h$, $0.2km/h$, and $0.3km/h$, respectively.

These results clearly highlight the importance of environment modelling when performing DSE to balance performance and energy efficiency. By modeling the entire system, including functional aspects like control logic and extra-functional properties such as energy consumption, MEbots permits the analysis of how different design choices influence the ability to achieve a Pareto-optimal trade-off between energy efficiency, drone speed and mission accomplishment.

VI. Conclusions and Future Works

This paper presented MEbots, a framework that combines a drone simulator with a VP equipped with an ISS and extra-functional support. The application to a nanodrone proved that MEbots simplifies early validation of software strategies and exploration of alternative design directions. In such a challenging and multi-disciplinary scenario, MEbots is thus a crucial resource that can help designers achieve better optimization. As future work, we would like to investigate MEbots support for other robotics scenarios, and further explore the software optimization that it enables.

REFERENCES

[1] A. Bartolini *et al.*, "A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores," in *Proc. of Symposium on Great Lakes Symposium on VLSI*, 2010, p. 311–316.

[2] M. Lora *et al.*, "Analog models manipulation for effective integration in smart system virtual platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 378–391, 2018.

[3] M. A. Hamdi *et al.*, "Integrating SystemC-AMS power modeling with a RISC-V ISS for virtual prototyping of battery-operated embedded devices," in *Proc. of ACM International Conference on Computing Frontiers: Workshops and Special Sessions*, 2024, p. 51–54.

[4] E. Fraccaroli *et al.*, "Modeling cyber-physical production systems with SystemC-AMS," *IEEE Transactions on Computers*, vol. 72, no. 7, pp. 2039–2051, 2023.

[5] S. Vinco *et al.*, "A layered methodology for the simulation of extra-functional properties in smart systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 10, pp. 1702–1715, 2017.

[6] L. Lamberti *et al.*, "Bio-inspired autonomous exploration policies with cnn-based object detection on nano-drones," in *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.

[7] Cyberbotics Ltd., "Webots," http://www.cyberbotics.com.

[8] Open Robotics, "Gazebo," https://gazebosim.org/home.

[9] NVIDIA, "NVIDIA Isaac Sim," https://developer.nvidia.com/isaac/sim.

[10] N. Bruschi *et al.*, "GVSoC: A highly configurable, fast and accurate full-platform simulator for RISC-V based IoT processors," in *Proc. of IEEE International Conference on Computer Design (ICCD)*, 2021.

[11] Accellera Systems Initiative, "SystemC analog/mixed-signal extensions," https://systemc.org/overview/systemc-ams/, 2024.

[12] Bitcraze, "Crazyflie 2.1+," https://www.bitcraze.io/products/crazyflie-2-1-plus/.

[13] K. Kalenberg *et al.*, "Stargate: Multimodal sensor fusion for autonomous navigation on miniaturized uavs," *IEEE Internet of Things Journal*, 2024.

[14] "MEbots: Integrating a RISC-V virtual platform with a robotic simulator for energy-aware design," https://github.com/eml-eda/messy.

[15] A. Akram *et al.*, "A survey of computer architecture simulation techniques and tools," *IEEE Access*, vol. 7, pp. 78 120–78 145, 2019.

[16] QEMU, "QEMU - a generic and open source machine emulator and virtualizer," https://www.qemu.org/.

[17] P. Adelt *et al.*, "QEMU support for RISC-V: Current state and future releases," *International Workshop on RISC-V Research Activities*, 2019.

[18] antmicro, "Develop your iot product with renode," https://renode.io/.

[19] F. Speiser *et al.*, "Embedded system simulation using Renode," *Engineering Proceedings*, vol. 79, no. 1, 2024.

[20] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, p. 1–7, 2011.

[21] A. Butko *et al.*, "Accuracy evaluation of GEM5 simulator system," in *International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, 2012.

[22] Spike, "Spike, a RISC-V ISA simulator," https://github.com/riscv-software-src/riscv-isa-sim.

[23] C. Duran *et al.*, "Simulation and formal: The best of both domains for instruction set verification of RISC-V based processors," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020.

[24] M. Montón, "A RISC-V SystemC-TLM simulator," https://arxiv.org/abs/2010.10119, 2020.

[25] V. Herdt *et al.*, "Advanced virtual prototyping for cyber-physical systems using RISC-V: implementation, verification and challenges," *Springer Science China Information Sciences*, vol. 65, 2021.

[26] D. Rossi *et al.*, "Energy efficient parallel computing on the PULP platform with support for openmp," in *Proc. of Convention of Electrical & Electronics Engineers in Israel (IEEEI)*, 2014.

[27] Z. Chen *et al.*, "A survey on open-source simulation platforms for multi-copter UAV swarms," *Robotics*, vol. 12, no. 2, 2023.

[28] M. Jiono *et al.*, "Software framework of autonomous mobile robots on Isaac Sim and ROS," in *International Conference on Control, Mechatronics and Automation (ICCMA)*, 2023, pp. 158–163.

[29] S. Shah *et al.*, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017.

[30] R. Madaan *et al.*, "AirSim drone racing lab," in *NeurIPS 2019 Competition and Demonstration Track*, vol. 123, 2020, pp. 177–191.

[31] Open Robotics, "Gazebo API reference - battery," https://gazebosim.org/api/gazebo/6/battery.html.

[32] Cyberbotics Ltd., "Webots reference manual - motor," https://cyberbotics.com/doc/reference/motor.

[33] NVIDIA, "NVIDIA Jetson," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/.

[34] N. Lohmann, "JSON for Modern C++," https://github.com/nlohmann, Nov. 2023.

[35] Bitcraze, "AI deck 1.1," https://www.bitcraze.io/products/ai-deck/.

[36] E. Flamand *et al.*, "Gap-8: A risc-v soc for ai at the edge of the iot," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2018, pp. 1–4.

[37] L. Guangdong UFine New Energy Co., "UFX 402525 250mAh 3.7V smallest rechargeable battery," https://www.gdufinebattery.com/products/ufx-402525-250mah-37v-smallest-rechargeable-battery, 2024, accessed: 2024-09-30.

[38] Atomic Workshop, "300mAh HD LiPo 60C battery (high power, ultra-compact)," https://www.atomicworkshop.co.uk/300LiPoHDPUP.htm, 2024, accessed: 2024-09-30.

[39] LiPol Battery, "15C LiPo battery," https://www.lipolbattery.com/15C-LiPo-Battery.html, 2024, accessed: 2024-09-30.

[40] K. Kalenberg *et al.*, "Stargate: Multimodal sensor fusion for autonomous navigation on miniaturized UAVs," https://zenodo.org/records/10546408, 2024.

[41] D. Kingma *et al.*, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2015.

[42] D. J. Pagliari *et al.*, "Plinio: A user-friendly library of gradient-based methods for complexity-aware dnn optimization," in *Proc. of Forum on Specification & Design Languages (FDL)*, 2023.

[43] M. A. Hamdi *et al.*, "MATCH: model-aware TVM-based compilation for heterogeneous edge devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2025.