A Coordinated Approach to Control Mechanical and Computing Resources in Mobile Robots

Sajad Shahsavari, *Student Member, IEEE*, Hashem Haghbayan, *Member, IEEE*, Antonio Miele, *Senior Member, IEEE*, Eero Immonen, *Member, IEEE*, and Juha Plosila, *Member, IEEE*

***Abstract*—Energy management of mechanical and cyber parts in mobile robots consists of two processes operating concurrently at runtime. Both the two processes can significantly improve the robots' battery lifetime and further extend mission time. In each process, information on energy consumption of one of the two parts is captured and analyzed to manipulate various mechanical/computational actuators in a robot, such as motor speed and CPU voltage/frequency. In this article, we show that considering management of mechanical and computational segments separately does not necessarily result in an energy-optimal solution due to their co-dependence; as a consequence, a runtime co-management scheme is required. We propose a proactive energy optimization methodology in which dynamically trained internal models are utilized to predict the future energy consumption for the mechanical and computational parts of a mobile robot, and based on that, the optimal mechanical speed and CPU voltage/frequency are determined at runtime. The experimental results on a ground wheeled robot show up to 36.34% reduction in the overall energy consumption compared to the state-of-the-art methods.**

***Index Terms*—CPU Dynamic Voltage/Frequency Scaling (DVFS), event camera, locomotion cost optimization, runtime resource management.**

I. Introduction

AUTONOMOUS battery powered mobile robots are currently being introduced in many intralogistics operations, such as transportation, forestry, mining, and extraterrestrial operations [1]. One of the major challenges in the field of mobile robot design and control is to reduce the energy consumption of its operation to increase the battery operational lifetime. In fact, even if battery technology is rapidly evolving [2], an efficient exploitation of system's resources is fundamental to enable the breakthrough in prolonging battery operational lifetime. A tangible proof of this concept in nowadays life is the widespread integration of asymmetric multicore CPUs, as the big.LITTLE architecture [3], in mobile and edge computing devices; they can opportunistically provide computing resources to the running applications having performance needs, thus avoiding any waste of energy consumption due to resource over-provisioning.

In the robotics field, the research has been focused mainly on optimizing the robot's mechanical operation and kinetic energy consumption (e.g., [4], [5]). However, kinetic energy consumption is not the only source of the energy drain and a mobile robot, as a cyber-physical device, contains a *cyber* part beside the physical, such as on-board electrical devices, microcontrollers, and sensors, each of which requires electric power to operate and contributes to the overall energy consumption [6]. The growth of intelligent and power-hungry applications for analyzing various types of sensory data from one side and the decrease of mechanical power consumption of small and tiny robots from the other side makes the cyber-part a considerable power/energy drain that should be considered in the power management process.

There exist various power/energy management techniques for embedded and mobile computing systems (e.g., [7], [8], [9], [10], [11]), which consider only the cyber part of a robot. Also, there exist efficient motion planning algorithms for mobile robots to optimize the power/energy consumption of their physical actions [12], [13], [14], [15], [16], [17], [18]. However, cyber and physical parts of a mobile robot are mutually influential on each other and, even though they might not rely on each other in a direct/immediate manner, their actions or outcomes still have an impact on one another. As an example scenario, consider an autonomous rover with the mission to smoothly navigate an unknown rugged terrain with an advanced perception system for obstacle detection and path planning. Besides a controller to manipulate the mechanical actuators of the rover, such as steering and motor speed knobs, the rover is equipped with an advanced onboard computing module capable of runtime monitoring of the computing power and manipulating the Dynamic Voltage/Frequency Scaling (DVFS) characteristics of its processing units to save power at runtime. As the rover moves forward, the camera continuously captures varying visual information, which is processed by the computing module, resulting in varying computing power consumption. According to this power consumption variation, the computing control unit of the rover dynamically adjusts the voltage/frequency of the processing units to save the power while guaranteeing the required Quality of Service (QoS). The QoS is dependent on the required accuracy of the obstacle detection and path planning algorithm and is required to be higher at higher rover speeds. This shows

Received 17 June 2024; accepted 19 September 2024. Date of publication 6 November 2024; date of current version 12 December 2024. This work was supported in part by the research Council of Finland -funded project 357220 - DOMINIC Developmental Multi-Robot Systems in Cognitive Manufacturing. This article was recommended for publication by Associate Editor Kiju Lee and Editor Paolo Robuffo Giordano upon evaluation of the reviewers' comments. *(Corresponding author: Sajad Shahsavari.)*

Sajad Shahsavari, Hashem Haghbayan, and Juha Plosila are with the Autonomous Systems Laboratory, Department of Computing, University of Turku, 20014 Turku, Finland (e-mail: sajsha@utu.fi; mohhag@utu.fi; juplos@utu.fi).

Antonio Miele is with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, 20133 Milano, Italy (e-mail: antonio.miele@polimi.it).

Eero Immonen is with the Turku University of Applied Sciences, 20520 Turku, Finland (e-mail: eero.immonen@turkuamk.fi).

Digital Object Identifier 10.1109/TRO.2024.3492345

that the computing workload is dependent on the mechanical speed of the rover and the complexity of the perceived visual data, which is related to the location of the robot. From the other side, the QoS depends on the availability of the computing resources and the capacity of the individual processing units. Therefore, in the case of limited computing resources, the rover needs to tune the speed to keep the QoS requirement low enough. This dependence of mechanical and computational parts directs us toward the fact that management of these two parts must be done together instead of optimizing these two control units oblivious to each other.

In this article, inspired by the intuitive dependencies between the mechanical and computational activities of the example rover described above, we experimentally demonstrate that separate independent control of mechanical and computational operations does not guarantee an optimal approach for energy management, but an intelligent energy management algorithm must monitor and adjust the mechanical and computational features in tandem to provide an optimal solution for the entire system. To address this, we propose a novel runtime co-management approach for a mobile robot in an uncertain environment, aiming at minimizing the overall energy consumption while satisfying the involved QoS requirements. The considered actuators are the knobs for controlling the speed of the motor and the DVFS settings of the CPU. To manipulate the actuators, the measured instantaneous power of the mechanical and computational parts is fed back to the controller to train the internal models of the parts. The controller then runs a predictive search algorithm over the possible configurations to fine-tune the system configuration for minimizing the overall energy consumption at runtime.

A preliminary approach to address this problem is presented in our earlier work [19], where we consider the optimization as a convex problem and use a reactive Hill Climbing (HC) algorithm to adjust the same two knobs based on naive local search. In this article, however, we show that the apply-and-measure strategy of the HC algorithm results in a significant performance deficiency and energy consumption, which can be avoided by using a proactive approach, which estimates the power consumption in advance based on a predictive machine learning model.

Overall, the main contributions of this article can be summarized as follows.

- A comprehensive investigation to motivate co-management of computational and mechanical parts in mobile robots by delving into details of their relationship under different environmental conditions and computational capabilities.
- Proposing an internal predictive model to estimate the computational and mechanical energy consumption of a mobile robot. This model is capable of being trained at runtime and is used for proactive optimization of energy control.
- Design of a controller for co-managing mechanical and computational activities of the mobile robot by selecting the most energy-efficient motor speed and CPU voltage/frequency configuration at runtime. The controller tunes the actuators based on the internal predictive model under varying environmental complexities and workloads.

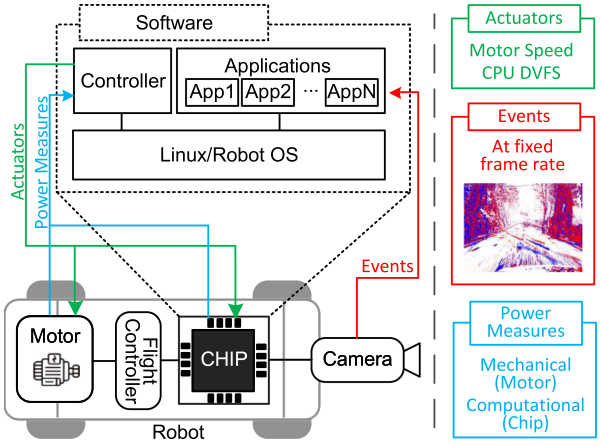

Fig. 1. Schematic of the reference robot system.

- A comprehensive experimental analysis of the proposed controller under different environmental conditions and processing capabilities in comparison with related state-of-the-art approaches for a ground mobile robot, showing up to 36.34% improvement in the energy consumption of the robot. In a mid-length track, averaged over various environment complexities, the results show 16.43% energy saving against the case of optimizing energy of the mechanical and computational parts separately, and 13.57% against a preliminary naive idea for the runtime co-management.

The rest of this article is organized as follows. Section II introduces the working scenario and motivates the proposed work by experimentally demonstrating the demand for the proposed co-management method. Section III discusses related works, where separate actions are generally taken for energy optimization of the mechanical and computational parts. Section IV discusses the proposed approach in detail, presenting the estimation models and the control algorithm. The results for the experimental evaluation of the proposed approach are presented in Section V. Finally, Section VI concludes this article.

II. Working Scenario and Motivation

We present the robot's operational scenario and a motivational example to demonstrate the co-dependence of mechanical and computational knobs on both energy and application performance. The example demonstrates an overview of the dynamics of the system under different environmental complexities and motivates the proposed approach for co-optimization of computational and mechanical energy at runtime.

A. Working Scenario

The reference robotic system consists of the overall robot architecture, and the running vision applications for autonomous operation. The schematic of the reference system is depicted in Fig. 1 while details on the two layers are provided as follows.

1) Overall Robot Architecture: The robot we consider in this article is an autonomous driving rover. The robot architecture can be divided into the mechanical and the computing parts; the former consists of an electric DC motor connected to the wheels and the latter has an onboard embedded computing board

connected to an event camera. An event camera, detecting only the changes occurring in the field of view (pixels) [20], [21], is employed for the perception of the surrounding environment because of its higher speed and energy-efficiency compared to RGB cameras. An additional rationale for incorporating an event camera into the framework of this study stems from its direct correlation with the rate of event generation over time. On the one hand, this rate is intricately tied to the velocity of the rover, and on the other hand, it is intertwined with the computational load. This intrinsic connection makes the count of generated events over time a pertinent metric, effectively illustrating the dynamic interplay between the mechanical and computational facets of the system.

The computing board comprises a multicore CPU architecture exposing various hardware and execution knobs; here we consider DVFS, that, as commonly known, is a fundamental feature to trade between performance and power consumption in a modern computing system. Further execution knobs, such as Graphics Processing Unit (GPU) acceleration and task migration in asymmetric CPUs will be considered in our future work. The computing board controls the motor speed through a flight controller and acquires events sensed by the camera. The entire system is powered by a battery pack and features sensors to measure the power consumption of both mechanical and computing parts. Power sensors are accessible from the computing board.

Linux and Robot Operating System (ROS) [22] run on top of the described hardware platform. Linux provides all operating system facilities for accessing the sensors and knobs of the computing board (in particular, the chip power sensors and DVFS management knobs) and for enabling multiprocess execution. In particular, DVFS is tuned by specifying the desired frequency level, and Linux automatically applies the corresponding voltage level; for this reason, from now on we will refer to the only frequency tuning. Then, ROS gives access to the sensors and actuators of the mechanical part (in particular, the power sensor and flight controller) and the event camera.

2) Applications: Applications here considered are based on vision algorithms for perception functionalities on input taken from the event camera; the produced results are then used as input for the robot path planning and control. Examples of such vision applications are for image reconstruction and corner detection. The workload running on the system is composed of multiple applications executed concurrently. Applications enter and leave the system with a trend unknown at design time.

The camera driver packages the events into batches having varying size and delivers them to the running applications at a fixed predefined rate R through an input first-in-first-out buffer. The batch size is a random variable; it depends on the external environment complexity and on the speed of the robot. The applications are structured as a loop continuously executing the specific vision algorithm on each received event batch in a race-to-idle manner; when the execution of the vision algorithm is completed on the current batch, if the subsequent event batch is not available yet, the application will wait for it, otherwise it will restart processing immediately. The applications are annotated with a QoS requirement, expressed in terms of a throughput (batches per second) to be guaranteed. Indeed, processing all incoming event batches may be computationally unfeasible; due to the intrinsic error tolerance of vision tasks, the specified QoS may be set to a lower value than the incoming event batch rate R. The fact that a new event batch pushed in the input buffer causes the deletion of the oldest batch allows that applications automatically skip unnecessary incoming event batches when QoS is lower than the input rate R.

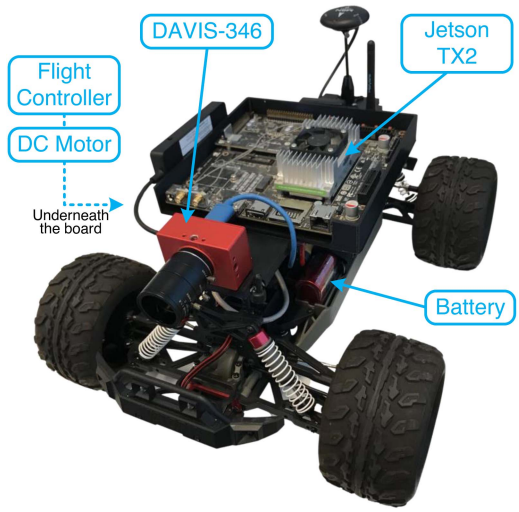

Fig. 2. Demonstrator setup.

The actual application's performance or throughput,[1] measured as the current number of event batches processed per second, is dependent on the external and internal working conditions. External condition include the complexity of the environment and the speed of the robot that will cause the camera to generate a corresponding number of events; the higher the environmental complexity, the larger the number of events per batch. In the same way, the speed of the robot will also have an influence on the number of generated events. Internally, the performance of the application in elaborating events is a function of the complexity of the executed vision algorithm and of the computing resources assigned to the application itself. Indeed, it is worth mentioning that different vision algorithms may have different performance in the same working conditions. Then, regarding to the assigned computing resources, the higher the voltage/frequency level set for the CPU, the higher the application's throughput.

Demonstrator: For experimental purposes, we have used the demonstrator reported in Fig. 2. A 3000 Kv brushless DC motor is used to provide the propulsive force for the robot.[2] The motor is controlled by an electronic speed controller connected to a Pixhawk (Pix32) flight controller. In the experimental sessions of this work, we vary the motor speed in the range between 0.25 and 5.0 m/s by means of a 0.25 m/s step, thus resulting in 20

[1]In this article, we will use the terms "performance" or "throughput" interchangeably to refer to the same concept.

[2]https://www.conrad.com/p/reely-new1-brushless-110-rc-model-car-electric-monster-truck-4wd-100-rtr-24-ghz-incl-batteries-and-charger-1559978

different speed configurations, while in the motivating example discussed in the following section, we use a more coarse-grained step, for the sake of readability of the reported plots. An NVIDIA Jetson TX2[3] is installed on the robot as a power-efficient high-performance embedded board, which provides computing resources to run the necessary applications for intelligent and autonomous operation. The Jetson TX2 features a heterogeneous architecture chip with an asymmetric multicore CPU and a GPU enabled for general purpose computing acceleration. The CPU is divided into two core clusters: a quad-core ARM Cortex-A57 and a dual-core NVIDIA Denver. DVFS can be controlled at the granularity of the single cluster, spanning from 499 MHz to 2.0 GHz through 11 steps. In this work, we consider only the quad-core ARM Cortex-A57 cluster.

The installed event camera is a DAVIS-346,[4] which can capture both intensity images and a stream of asynchronous events with a high temporal resolution (up to 10 million events in a second). The power supply is an 11.4 Volt 3500 mAh lithium polymer battery, which is used to power up both electric motor and Jetson TX2. The power consumption of the electric motor is measured by a mounted power module, while the power consumption of the Jetson TX2 is measured by the on-chip sensors. Finally, the considered platform runs Linux as operating system and ROS on top of that.

The workload has been generated with parallel execution of instances of three state-of-the-art vision applications processing events for the following: i) image reconstruction [23], ii) corner detection [24], and iii) corner detection with filtering [25], (one instance of each application).

B. Motivating Discussion

Recent methods for energy-optimal path/speed planning and control of the autonomous robots have been mainly focused on minimizing locomotion/mechanical energy costs, i.e., the energy spent in the robot's propulsion system, such as DC motors in ground [12], [13], [14], [15], [16], [17] or aerial vehicles [18], or hydraulic pumps in legged robots [26]. In this section, by presenting a motivational example, we demonstrate that the cost of computation should be also taken into account in robot's speed planning, since it considerably affects the overall energy consumption of the robot.

Considering the working scenario described in Section II-A, Fig. 3 shows the mechanical energy consumption per distance (i.e., integration of measured power over travelling time) by the motor for different speed configurations. It can be noticed that if the robot moves at minimum speed, the energy consumption will be high. This is because integration of power is performed over a longer period of time, and despite low electric power requirement, the integral interval governs the amount of energy consumption. On the contrary, if the robot moves fast (with maximum speed), the motor requires high electric power, and thus (despite shorter travelling time and integration interval), the energy consumption will be high as well, due to the large integrand. According to the trend shown in Fig. 3, if we consider only the mechanical energy, the speed value that achieves minimum mechanical energy consumption is 2 m/s.

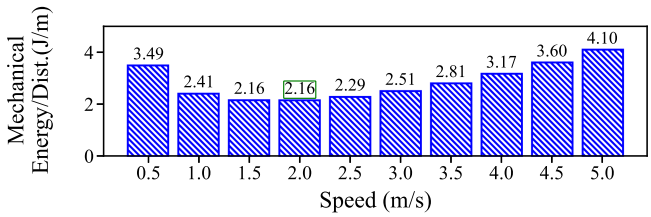

Fig. 3. Mechanical energy per distance for different speed configurations.

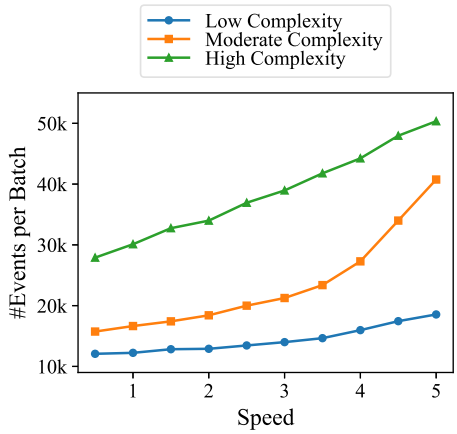

Fig. 4. Number of events per batch is dependent on the inherent complexity of the environment and speed of the robot.

Now, let us consider also the computing energy. In particular, we run the corner detection application with the input rate of 30 event batches per second and a QoS requirement of 28 event batches per second by considering three levels of environment complexity, namely low, moderate, and high, respectively.[5] To run this new experiment, we should notice that the CPU frequency level is tuned by means of DVFS control to the minimum step that provides enough computing resources to achieve the application's required QoS (as demonstrated in [7]). As introduced in the description of applications, this CPU frequency configuration is dependent on the number of events per input batch that, in turn, increases with i) the higher motor speed and ii) the level of complexity of the external environment where the robot is operating. Indeed, each external environment has a distinct intrinsic complexity level, which affects directly the batch sizes in the incoming stream of events (larger batches in more complex environments). Furthermore, in each specific environment complexity, varying speed will consequently vary the batch sizes, with larger batches in higher speeds. Fig. 4 gives a graphical view of the relationship between the number of events per batch and the environmental complexity and robot's speed.

Results of this second experiment considering both mechanical and computational energies are reported in Fig. 5. We

[3]https://developer.nvidia.com/embedded/jetson-tx2

[4]https://inivation.com/wp-content/uploads/2019/08/DAVIS346.pdf

[5]The levels of environmental complexity are described in Section V-A.

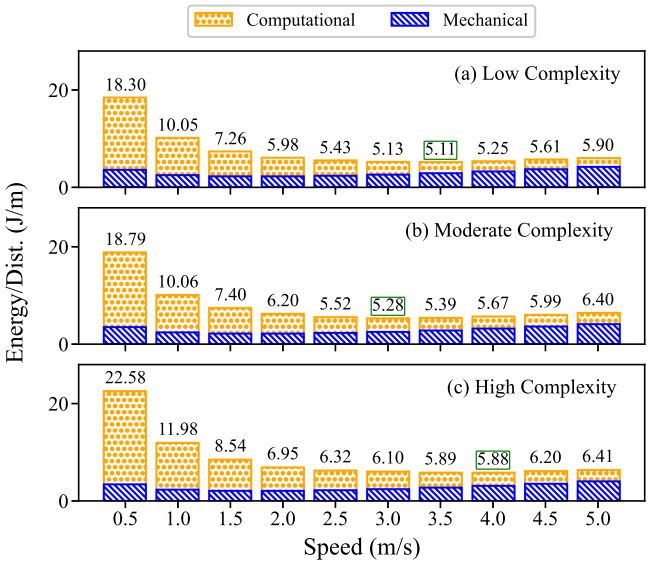

Fig. 5. Total energy per distance (mechanical + computational) for different speed configurations in 3 different environment complexities.

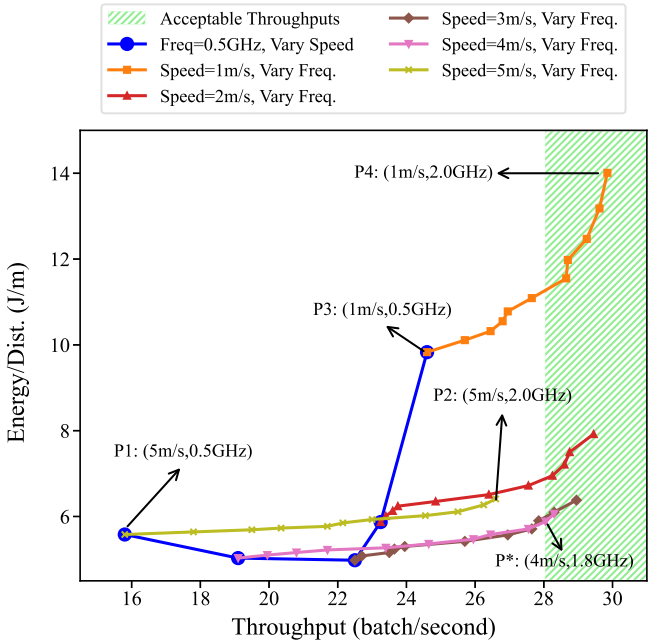

Fig. 6. Energy per distance versus application throughput for high-complexity environment. This shows the effect of changing the speed and frequency on both energy and throughput criteria. The frequency is increased from P1 to P2 while keeping the speed constant at 5 m/s. From P1 to P3, the frequency is kept constant and speed is decreased. Also, from P3 to P4, the speed is kept constant at 1 m/s and frequency is increased to its maximum level. P* shows the energy-optimal configuration.

can notice that the computational energy cannot be neglected, and in various scenarios, especially at low engine speed, it is predominant. This last case is due to the large idle power consumption, to keep the computing board on and running. Furthermore, comparing to the first experiment, we can notice that the both the trend in the plot and the minimum-energy speed change significantly, and, minimum-energy speed configuration is not the same for all complexities.

We better investigated the effect of motor speed and DVFS on the energy consumption and application's performance. Fig. 6 shows achieved results in the high environmental complexity scenario for different working configurations in the throughput versus energy per distance plot. In configuration P1, the robot is operating at speed equal to 5.0 m/s (maximum speed) and frequency of 0.5 GHz (minimum frequency); in this condition, the application is violating the throughput requirement (the green area represents the solution space where the QoS is satisfied). To increase the application's throughput, a first possibility may be to increase the CPU frequency; this action is represented by the yellow line, where in the last point, P2, the maximum CPU frequency level is set (2.0 GHz). In conclusion, despite the increase in energy consumption, still the required throughput is not met.

A second alternative from P1 is to decrease the motor speed, with a 1 m/s step, while keeping the CPU frequency fixed at the minimum level. This action, represented by the blue line, ends in P3 (speed equal to 1 m/s), where again the required throughput is not met and the energy consumption is even worse. However, from P3, if we increase the CPU frequency to the maximum level, the application is provided with the necessary computing resources to guarantee the QoS (point P4). Although P4 satisfies the required throughput, it is far from energy-optimality. In fact, on the orange line we can notice that there are some other configurations guaranteeing QoS with lower CPU frequency level. Moreover, if we explore DVFS also for all the remaining motor speed configurations, we can identify the optimal working configuration P*, having a speed of 4 m/s and a CPU frequency of 1.8 GHz.

In conclusions, we demonstrated that mechanical and computational contributions have to be jointly considered to reduce the overall energy consumption of a moving robot system running vision applications. In the considered system such an optimization is performed by acting on both the motor speed and DVFS of the multicore CPU. Moreover, this optimization task is not trivial due to the large number of working configurations and dependence from the environmental conditions and applications' QoS requirements; the situation is even more exacerbated by the fact that this optimization has to be performed at runtime due to the fact that the actual working scenario is unknown in advance. Given these motivations, this article presents a novel runtime energy-efficient control approach to jointly manipulate the mechanical and the computational actuators of the mobile robot together in uncertain environments to obtain the best possible overall energy consumption while satisfying application QoS requirements.

III. Related Works

This section reviews the literature on the considered working scenario. We divide the related works into three categories:

i) studies that assumed that the mechanical (locomotion) cost is the main (or the only) source of energy consumption and optimized the robot actions accordingly;

ii) studies that considered only computational resource allocation techniques for minimizing computational energy;

iii) studies that proposed a multiobjective optimization scheme that considers mechanical cost with some other criteria.

A. Locomotion Cost Optimization in Robots

There have been several research works in energy-efficient motion/path planning that assume that the mechanical energy (locomotion cost) is the major source of energy consumption. Those studies attempt to minimize the locomotion cost usually by employing an internal power model to predict the mechanical power consumption given robot's operational configuration such as speed, acceleration, angular velocity, and environmental properties, such as surface friction, wind speed, payload, or a combination thereof. The power models are either data-driven [12], [16], [18], or physics-based [13], [14], [15], [17], on top of which some search/optimization algorithm (such as A*, Dijkstra, dynamic programming, etc.) is used to identify the optimal configuration on the built energy map of the environment.

Mei et al. [12] presented an offline comparison of energy consumption between different predefined path types and speeds utilizing a six-degree polynomial power model. Although this study does not provide an optimization scheme, it shows that robot's energy efficiency peaks at some specific speed value for all the considered path types. In [13] and [14], a physics-based power model is proposed based on Newton's second law, which relates the mechanical power to robot's speed and acceleration. Tokekar et al. [15] utilized a mathematical modeling of a brushed DC motor based on its angular velocity for mechanical power prediction and solved the velocity profile optimization in a closed form for the case when velocity is not bounded, and by dynamic programming for the case when velocity has a maximum bound constraint. Wei and Isler [16] proposed an adaptive data-driven statistical model (based on Gaussian probability density function), which its covariance matrix is obtained with the aid of aerial segmented images. Then, a Dijkstra graph search algorithm is applied on the built energy map of the environment. In [17], a power model for Ackermann steering ground vehicles is presented based on the angular velocity of the wheels and the steering angle of the robot. On top of this energy model, an A* search algorithm is proposed to perform the energy optimal path planning. In [18], a data-driven DNN-based power model is proposed for rotary autonomous aerial vehicle, which predicts the electric power supplied by the battery with inputs such as velocity, acceleration, altitude, wind velocity, weight, and surface area of the payload.

While the data-driven models [12], [16], [18] tend to have high accuracy on predicting the power consumption, inference time of such compute-intensive models limits performing online real-time path planning algorithms that require inference of power estimation frequently. Moreover, training such models require huge amount of data and adapting them to new unknown environments with few collected data points is challenging. On the contrary, even if less accurate than the previous ones, physics-based models [13], [14], [15], [17] enforce the known dynamics of motion (Newton's Law) and require only a handful of parameters to be identified. In this sense, physics-based models are more suitable for adaptive online path planning that requires runtime update of the model parameters based on the operating conditions. Given this comparison, here we adopt a physics-based model based on Newton's second law to estimate the mechanical power consumption of the robot and update the model parameters in runtime. Moreover, although proposed offline optimizations provide insights on the robot's dynamics and optimal solution, they have limited practicality in real scenarios. Finally, none of the aforementioned studies consider the cost and performance of computation in the optimization problem.

B. Energy Optimization in Embedded Computing Systems

In embedded and mobile systems saving energy is fundamental since such devices are commonly battery powered. The research literature of the last 15 years on runtime resource management for energy and power optimization for this kind of systems is extensive. One of the first works, targeting an asymmetric multicore, is proposed in [7]. The idea was to use proportional–integral–derivative controllers to manage DVFS, CPU quota, and task migration between CPU clusters in order to minimize power consumption while guaranteeing applications' required throughput. The throughput is measured at application level by means of the HeartBeat mechanism [27]. Subsequent works have better refined control techniques, adopting estimation models [8], and greedy/heuristic algorithms for defining the runtime control policy [9], [11], and broadened the working scenario by considering asymmetric CPUs, GPUs, and accelerators [9], [10], [11] or multithreaded applications [10] with performance requirements changing over the time.

Other investigated strategies to save energy in embedded platform aim at reducing the computing load by acting directly on the executed applications and related parameters. Such strategies have also been employed in the robotic scenario and consist of offloading vision tasks to the Cloud [28], [29] or switching to lower accuracy perception models [30]; however, these investigations do not perform any optimization of the energy consumption of the computing boards, relying on the assumption that the robot computing resources do not suffice in executing complex tasks. Nonetheless, offloading the compute tasks requires the robot to maintain a persistent connection (e.g., cellular or WiFi) to the cloud, which introduces a new dimension for energy consumption, that is the communication energy. Moreover, providing such connection might not even be possible for robots operating in remote sites. The model switching techniques, on the other hand, trade the perception accuracy with onboard computational load/energy; in other words, based on the *approximate computing* concept, it is possible to reduce the computing load (and in turn, the energy consumption) at the cost of introducing errors in the elaborations potentially impacting on the robot operations.

None of the mentioned works has been employed in the robotic scenario aimed at performing a joint optimization with the mechanical energy consumption. The only exception is our previous work in [19] that is here extended. Current solutions for runtime resource management of the computing systems in the robotic scenario are based on standard mechanisms integrated in

the operating systems that are quite basic. For instance, in Linux DVFS governor and the HMP scheduler (the last one tailored for asymmetric CPU architectures) take decisions on the basis of the measured Instructions Per Cycle (IPC) of the CPU cores. However, the IPC has no direct relationship with the application's throughput; as a consequence, Linux control mechanisms are not capable at guaranteeing the required QoS without resource over-provisioning and consequent energy wasting.

C. Coordinated Optimization for Robot's Control

Recent studies investigated the energy drained by other parts of a robot along with its mechanical parts. The cost of wireless communication, as one part of the robot, has gained considerable attention in recent years in the field of networked mobile robots [31], [32], [33], [34], [35], [36]. In essence, these studies are motivated by the fact that the energy consumption of radio devices increases with the transmission distance while their connectivity decreases. It is shown that the cost of data communication in a robot is considerable with regard to its total energy consumption and lifetime in terms of battery duration [31], and therefore, a co-optimization scheme is needed to jointly minimize costs of communication and motion together while satisfying connectivity throughput. Specifically, Yan and Mostofi [32] proposed a co-optimization framework to schedule robot's speed, data transmission rate, and stop time in a predefined trajectory, while minimizing its overall energy consumption and satisfying a target network connectivity criterion (bit error rate). Licea et al. [33] proposed an offline optimization scheme to the problem of minimizing mechanical energy of a drone while maximizing the number of bits transmitted to a ground station during its flight. Zeng et al. [34] considered both communication and motion energy consumption and also the communication throughput requirement in the co-optimization. Liu et al. [35] considered a team of networked mobile robots that communicate with each other and with a ground station. They proposed a theoretical motion planning framework that optimizes communication QoS based on a two-layer hierarchical model predictive control. We refer the reader to the review in [36] for detailed discussion on communication-aware robotics. Considering communication energy is an interesting research direction especially for a group of cooperating robots or when there is a high interaction with the base station. However, in our working scenario, the robot is autonomous; it does not communicate large amount of data with ground station and, thus, does not require to satisfy communication QoS. Therefore, here the communication energy is not considered at present since it does not represent the main contribution to the overall energy consumption.

The only work in the robotics literature that considered the cost of computation is our previous work in [19], which proposed a preliminary search approach based on HC for co-optimization of mechanical and computational energy. The naive HC algorithm actually applies all the immediate neighbor configurations of speed and frequency on the robot, to identify the least energy-consuming one to be used next. This strategy is highly inefficient since it continues changing operational configuration of the robot in an almost uninformed way to identify the more promising local move in the search space, thus causing a long decision process due to the actual apply-and-measure process. The main improvement of our proposed approach in the present work is to use trained internal models to predict the power consumption of all configurations to avoid unnecessary trials. We show that our proposed approach here significantly improves the energy consumption by finding the optimal configuration almost instantly and, thus, reducing the search time and consequently total energy consumption.

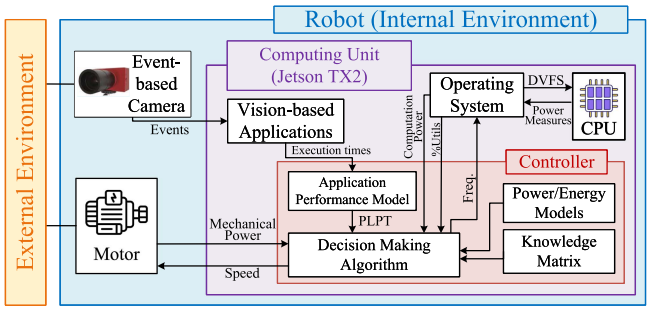

Fig. 7. Overview of the controller integrated in the overall working scenario.

IV. Proposed Controller

We propose a controller to manage, in a coordinated way, the mechanical and computational parts to guarantee the QoS required by the running applications, while minimizing the energy consumption of the entire robot. Fig. 7 shows the architecture of the controller (depicted in a red box) integrated in the overall robotic system. The controller interacts with the following:

- the vision applications by receiving the current performance measure in terms of execution times of the computation loop;
- the operating system to read the utilization level of each core in the CPU and the power consumption of the overall computing board, and to set CPU frequency by means of DVFS control;
- the mechanical part to read the related power consumption and to set the motor speed.

The controller is a software process running on top of the operating system and implementing a feedback-control loop executed with a given period (here tuned to 1 s). The feedback-control loop is organized based on the Observe-Orient-Decide-act (OODA) scheme [37]:

- *Observe:* The controller collects all measures from sensors and monitors;
- *Orient:* It updates the internal knowledge repository based on the current observations;
- *Decide:* It performs a decision making in terms of how to actuate on the available knobs; this process is carried out based on the knowledge repository to fulfill the specified objective functions (i.e., minimizing energy consumption while guaranteeing QoS);
- *Act:* It enforces the decisions on the knobs.

Due to the complexity of the performed task, the controller is structured in various parts.

- *Power and energy models* that are used to estimate the power and energy consumptions of the mechanical and computational parts in the case a specific knob configuration would be enforced in the current running conditions.
- *Application performance model*, which takes in input the execution times of each running application taken for processing a single event batch and computes the applications' Per Loop Processing time (PLPT), an application-level metric here used to monitor applications' current performance level.
- *Knowledge matrix*, a data structure containing information on the current and previous status of the system that is relevant for the decision making process.
- *Decision making algorithm* that is the actual policy identifying the proper working configuration that minimizes the overall energy consumption while guaranteeing the applications' QoS requirements.

A detailed description of the various parts of the controller is given in the following sections.

A. Power and Energy Models

A limitation of the preliminary version of the approach presented in [19] was that to evaluate the quality of each explored configuration, it was necessary to enforce it to measure the actual power consumption. This approach is ineffective since in the considered problem, the configuration space is large and depends also on the running workload and environment complexity; moreover, the approach scalability would be very bad in the case the approach is extended to consider additional knobs. For this reason, we adopt two mathematical models to predict the power consumption of mechanical part and computational part in the case a specific configuration would be applied, without actually enforcing it. Consequently, energy consumption of the robot is also estimated. These estimation models are presented in the following.

1) Mechanical Power Estimation: Similar to [13], [14], we construct a model to predict mechanical power consumption of the robot based on its speed and applied forces, according to Newton's second law. The power demand for moving the robot on a flat surface is calculated by the following:

$$P_m = Fv \tag{1}$$

where v is the velocity of the robot and F is the propulsive force that the motor provides. In turn, F is computed as

$$F = F_a + F_f + F_d \tag{2a}$$

$$F_a = m\frac{dv}{dt} \tag{2b}$$

$$F_f = (C_r + C_v v) \cdot mg \tag{2c}$$

$$F_d = \frac{1}{2}\rho C_d A_f v^2 \tag{2d}$$

where F_a is the acceleration force, F_f is the rolling friction force, and F_d is the aerodynamic drag. Then, m is mass and g the gravitational constant, C_r and C_v are constant and viscous rolling coefficients respectively, ρ is the air density, C_d is the aerodynamic drag coefficient, and A_f is the robot's frontal area.

The parameters of this power model (C_r, C_v, and C_d) can be empirically identified by fitting the model in (1) and (2) to the measurements of the robot; parameter fitting is performed by means of regularized least squares method.

2) Computing Power Estimation: For the computing power, we adopt an empirically fitted model commonly used in the literature for multicore processing boards [8], [11] and identify the parameters for the considered compute board. In particular, the total power consumption of a computing board can be computed in an additive way according to the following formula:

$$P_c = P_{board}^{idle} + \sum_i \left(P_i^{idle}(f_i) + P_i^{work}(f_i) \cdot \%U_i \right) \tag{3}$$

where P_{board}^{idle} is the power of the board in idle conditions; it includes also the camera power consumption that we assume to be almost constant. The variable part of the power consumption is due to the CPU activity, that is a function of the frequency level f_i and utilization $\%U_i$ of each core i. In that part of the equation P_i^{idle} and P_i^{work} are the empirically measured power consumption of a single core i at a specific frequency level f_i when no process is running ($\%U_i = 0\%$) and when the core is fully occupied for computations ($\%U_i = 100\%$), respectively. Then, according to the current utilization of each core that is returned by the operating system, the power consumption of a core can be estimated by means of a linear scaling function.

3) Energy Estimation: The energy consumption of the robot in a specific period of time is computed as the integral of its power consumption over that period. Since the proposed approach acts at runtime, we adopt the energy per distance as the cost function for optimization. We estimate the energy per distance $E^d(v, f)$ of a given configuration with motor speed s and CPU frequency f by dividing the sum of computing and mechanical power by the motor speed

$$E^d(v, f) = \frac{P_m(v) + P_c(f)}{v}. \tag{4}$$

It is worth noting that both the power and the energy of the system are additive functions. Therefore, the presented models can be easily extended to any new mechanical/computing unit added to the robotic system; in particular, it is necessary to define a parametric power model specific for each unit and integrate it in the sum in the numerator of the ratio in (4). For instance, in the previous work in [38], it has been shown how to extend the power model of a computing system to include the contribution of the cooling fan. For the mechanical part, the model can be extended for armed robots by considering the power consumption of the arm actions as shown in [39].

B. Application Performance Model

As discussed in Section II-A, the event camera generates an input stream of batches at a predefined rate R. Fig. 8 shows how each application takes a different amount of time to process the same batch based on the computational complexity of the

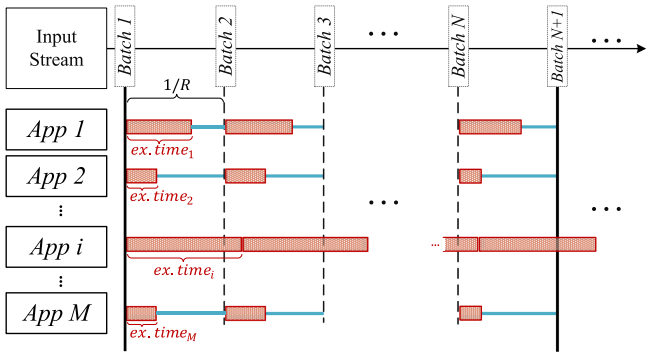

Fig. 8. Timeline showing a set of applications executing a series of batches.

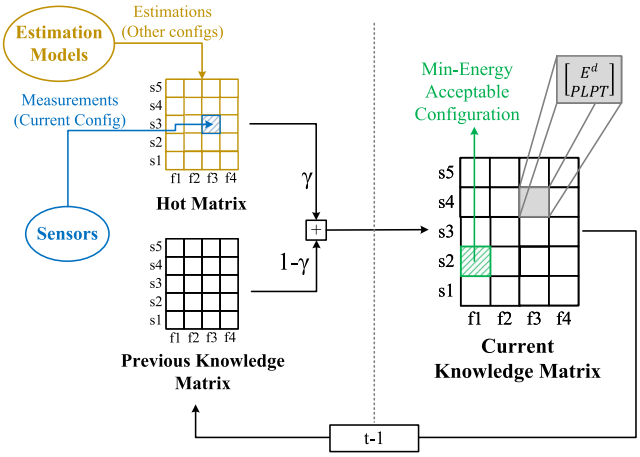

Fig. 9. Knowledge structure encapsulating all necessary information for decision making and its schematic weighted update mechanism.

specific vision algorithm; some application (e.g., *App* 1 and 2) ends computing before the given period (i.e., $1/R$), thus going to idle until the subsequent batch arrives, while some other one (e.g., *App* i) takes longer. In this context, *App* i achieves performance lower than R, thus, periodically skips some event batch; this is not an issue until the achieved performance is at least equal to the QoS requirement QOS_i. On the other hand, even if the execution time of *App* 1 and 2 is lower than $1/R$, their throughput saturates to R. In this case, a too-high CPU frequency leads to a resource over-provisioning and, in turn, a wasting of computing power consumption without any advanced in terms of performance [8].

This example shows how the throughput, commonly used in the literature for measuring the application performance (e.g., [7], [8]), cannot be used in the current context due to its saturation to R when over-provisioning CPU resources. For a finer-grained control, the proposed controller monitors the execution time of the applications for processing each event batch. Moreover, since the number of events per batch is not fixed, the execution time for the single application is variable in time. For this reason, the controller keeps track of the series of execution times for the last control period and computes a performance metric, called PLPT, as the third quartile Q_3 (75th percentile) statistic of such a series. The choice of Q_3 mitigates the effect of outliers and lowers the risk of performance violation in the control policy discussed in the following.

Among all the measured PLPTs, the controller will take into account the one of the most performance-demanding application. In fact, in order to save computing power [8], the preferred DVFS configuration should reduce as much as possible the idle time, and since the CPU has a single DVFS knob, such a tuning should be performed on the most performance-demanding application. Such an application i is the one minimizing its idle time percentage defined as

$$\%idle_i = \frac{1/QOS_i - PLPT_i}{1/QOS_i}. \quad (5)$$

Do note that when $\%idle_i$ is negative, the application is violating the QoS requirement. Finally, in case the QoS requirement is equal for all applications, the most challenging application is simply the one having the maximum PLPT value.

The application performance model integrates also an estimator of PLPT for a new desired configuration, based on the currently-measured metric value. Similarly to the power counterparts, this estimator supports the control decision process in exploring new working configurations without actually enforcing them. As discussed in [8] and [11], the application's performance has a linear relationship with the CPU frequency level. Moreover, the relationship with the motor speed is assumed to be sublinear; this is because the number of generated events is sublinear in the motor speed as can be noticed from Fig. 4 and also discussed in [40]. Thus, $PLPT_{new}$ of a new configuration (f_{new}, s_{new}) can be estimated from current values $PLPT_{curr}$, f_{curr}, and s_{curr}, as

$$PLPT_{new} = \left(\frac{f_{new}}{f_{curr}}\right) \cdot \left(\frac{s_{new}}{s_{curr}}\right)^{\beta} \cdot PLPT_{curr} \quad (6)$$

where β is an empirically derived constant.

C. Knowledge Matrix

To support the decision making process, the proposed controller exploits a data structure capturing relevant information on the performance and the energy consumption of the system for the various points of the working configuration space in the past history of the robot activity. This data structure is called *knowledge matrix*; as shown in the right part of Fig. 9, it is organized as a grid where each cell contains the computed metrics for a specific motor speed s and CPU frequency f configuration. Each cell contains a tuple with metrics for storing i) overall energy consumption per distance (E^d), and ii) PLPT for the most performance-demanding application.

The controller updates at each control cycle its knowledge by an adaptive exponential moving average of a *hot matrix*, containing measured and estimated values on the current working configuration, with the previous knowledge matrix to form an accumulated long-term understanding of the system behavior. As shown in the top left part of Fig. 9, the hot matrix is a grid with the same shape of the knowledge matrix; the cell representing the current working configuration is filled with measures obtained by sensors, while the power, energy and performance models

(1), (3), (4), and (6) are used for computing the tuples to populate all other cells.

Then, for each speed s and frequency level f, the element in knowledge matrix is updated as follows:

$$\mathbf{K}^t[s,f] = \gamma^t \cdot \mathbf{H}[s,f] + (1-\gamma^t) \cdot \mathbf{K}^{t-1}[s,f] \tag{7}$$

where $\mathbf{K}$ and $\mathbf{H}$ refer to knowledge matrix and hot matrix, respectively, t is the current control cycle, and $\gamma^t \in (\gamma_{min}, \gamma_{max}]$ is the decay factor. The factor γ supervises the amount of attention to the observation versus history and its value directly affects the reaction time of controller to unexpected situations. To control the robot's operation robustly and stably, we additionally control this reaction time by an adaptation scheme for the value of γ as

$$\gamma^t = \alpha \cdot \gamma^{t-1} + (1-\alpha) \cdot \gamma_{min} \tag{8}$$

where α is the constant decay factor and we set $\gamma^0 = \gamma_{max}$. The controller resets γ to its maximum value whenever its performance drops due to any significant change in the environment, application, or internal models' error. By using (8), the robot can effectively utilize the observed information at the moment in unknown situations, while exploiting the accumulated knowledge in the known situations.

D. Decision Making Algorithm

The last component in the controller is the decision making algorithm that implements the OODA loop and the other modules to decide actuation to minimize energy consumption while guaranteeing required QoS.

The decision process is enabled by performing a search on the knowledge matrix $\mathbf{K}$ to find the best configuration

$$s^*, f^* = \arg\min_{s,f} \{\mathbf{K}[s,f][E^d] \text{ s.t. } \mathbf{K}[s,f][PLPT] \leq PLPT_{ref}\} \tag{9}$$

where $\mathbf{K}[s,f][E^d]$ is the estimated energy per distance of the configuration where speed equals s and frequency equals f, $\mathbf{K}[s,f][PLPT]$ is the PLPT estimation of that configuration, and $PLPT_{ref}$ is the reference PLPT for the most critical application. This last one is in turn computed as

$$PLPT_{ref} = \kappa \cdot \frac{1}{QOS_{ref}} \tag{10}$$

where QOS_{ref} is the throughput reference for such a most critical application and $0 < \kappa \leq 1$ is a coefficient to provide a higher safety bound for throughput satisfaction. The smaller the value of κ the more conservative the behavior of the controller to avoid throughput violations that may occur during the runtime decision making process. Indeed, this search is performed at runtime by checking all the elements (configurations) of the knowledge matrix in speed/frequency space for the minimum-energy QoS-guaranteeing one.

The proposed control technique is presented in Algorithm 1. The initialization phase (Lines 1–6) includes setting the

Algorithm 1: Controller Loop.

Main variables:
- $\mathbf{K}$, $\mathbf{H}$ ▷ Knowledge matrix and hot matrix
- γ ▷ Decay factor for updating the knowledge matrix
- $data_{mech}$, $data_{comp}$ ▷ Vectors for logging sensed data
- $state$ ▷ Current state of the decide finite state machine

Constants:
- D_f, D_s ▷ Numbers of frequency and speed levels
- γ_{max}, γ_{min}, α ▷ Decay factor parameters, Eq. (8)
- κ ▷ PLPT safety factor
- $\%err\_thr_{mech}$, $\%err\_thr_{comp}$ ▷ Error thresholds for the power models
- $period$ ▷ Period for the control cycle

1: /* ———— Initialization ———— */
2: $\mathbf{K} \leftarrow [\{E^d, PLPT\} = 0]_{D_f \times D_s}$ ▷ Knowledge matrix zero set
3: $\gamma \leftarrow \gamma_{max}$
4: $data_{mech} \leftarrow \emptyset$
5: $data_{comp} \leftarrow \emptyset$
6: $state \leftarrow$ `apply`
7: **while** `true` **do**
8: /* ———— Observe ———— */
9: $s, f \leftarrow$ GetCurrentConfig()
10: $p_m, p_c, \%U \leftarrow$ ReadSensors()
11: $data_{mech}$.add($\{s, p_m\}$)
12: $data_{comp}$.add($\{f, \%U, p_c\}$)
13: $apps \leftarrow$ GetRunningApplications()
14: **if** $apps$.size() > 0 **then**
15: $a_{crit} \leftarrow$ FindMostCriticalAppplication($apps$)
16: $PLPT, QOS_{ref} \leftarrow$ ReadPerformanceInfo(a_{crit})
17: **else**
18: $PLPT \leftarrow 0$
19: $QOS_{ref} \leftarrow 1$
20: **end if**
21: $PLPT_{ref} \leftarrow \kappa \cdot 1/QOS_{ref}$
22: /* ———— Orient ———— */
23: $\hat{p}_m, \hat{p}_c \leftarrow$ PredictPower($s, f, \%U$) ▷ Eq. (1) and (3)
24: **if** $\frac{\|p_m - \hat{p}_m\|}{p_m} > \%err\_thr_{mech}$ **then**
25: TrainMechanicalModel($data_{mech}$)
26: $data_{mech} \leftarrow \emptyset$
27: **end if**
28: **if** $\frac{\|p_c - \hat{p}_c\|}{p_c} > \%err\_thr_{comp}$ **then**
29: TrainComputationalModel($data_{comp}$)
30: $data_{comp} \leftarrow \emptyset$
31: **end if**
32: $\gamma = \alpha \cdot \gamma + (1-\alpha) \cdot \gamma_{min}$ ▷ Eq. (8)
33: $\mathbf{H} \leftarrow$ CreateHotMatrix($s, f, \%U, p_m, p_c, PLPT$)
34: $\mathbf{K} \leftarrow$ UpdateKnowledgeMatrix($\mathbf{K}, \mathbf{H}, \gamma$) ▷ Eq. (7)
35: /* ———— Decide ———— */
36: **if** $state =$ `apply` **then**
37: $s^*, f^* \leftarrow$ GetMinEnergyConfig($\mathbf{K}, QOS_{ref}$) ▷ Eq. (9)
38: **if** s^*=s and f^*=f **then**
39: $state \leftarrow$ `best`
40: **end if**
41: **else** ▷ i.e., $state =$ `best`
42: $s^*, f^* \leftarrow s, f$
43: **if** $PLPT > PLPT_{ref}$ **then**
44: $\gamma \leftarrow \gamma_{max}$
45: $state \leftarrow$ `apply`
46: **end if**
47: **end if**
48: /* ———— Act ———— */
49: ApplyConfig(s^*, f^*)
50: Wait($period$)
51: **end while**

knowledge matrix to zero and creating various utility variables and data structures used during the decision making. Then, the main loop (Lines 7–51) performs the OODA scheme at each control cycle. A description of the actions in each OODA phase is the following.

Observe (Lines 8–21): First, the controller gets from the operating system the current working configuration (s, f) and the sensory measurements (mechanical and computing power consumption p_m and p_c, respectively, and the vector of CPU cores' utilization $\%U$). These data are logged in two vectors, namely $data_{mech}$ and $data_{comp}$; in the former we add the $\{s, p_m\}$ tuple, while in the latter the $\{f, \%U, p_c\}$ one. Then, it exploits the application performance monitor to get the list of running applications, to identify the most performance-demanding one and get its PLPT and QoS. If no application is running, there is no QoS requirement as well; in this case, PLPT and QoS are set to constant values, zero and one, respectively. Finally, $PLPT_{ref}$ is computed.

Orient (Lines 22–34): As discussed in Section II, working conditions are a priori unknown and their changes may require one or both the models to be refined in order to return accurate estimations. The computation context may change due to entering new types of applications or leaving of currently running ones. The mechanical environment, though, may change due to differences in operating surface types (asphalt, dirt, grass, carpet, etc.) which directly affect the coefficients of mechanical power model. For this reason, during this phase, each power model is continuously checked against currently-acquired measures; in case the relative error is greater than a given threshold, the model is retrained (Lines 23–31).

After that, the decay factor γ is updated based on (8), and the hot matrix containing currently sensed and predicted information is built. Finally, all elements in the knowledge matrix are updated based on the hot matrix, according to (7) (Lines 32–34).

Decide (Lines 35–47): With the up-to-date information in the knowledge matrix, the controller searches for optimal configuration for the current working scenario; this activity is performed with a two-state state machine to perform a stable control. The controller starts in the `apply` state, as initialized at Line 6. In the `apply` state, `GetMinEnergyConfig` function performs an exhaustive search on the knowledge matrix $\mathbf{K}$ to find the minimum-energy configuration (f^*, s^*) where PLPT is lower than $PLPT_{ref}$. This process is repeated at each control cycle until `GetMinEnergyConfig` function returns the same configuration of the previous iteration, being (f, s); this means that knowledge matrix is not modified anymore, and the controller converged in a stable state to the best configuration. Therefore, the controller goes to `best` state. It is worth noting that, as we will experimentally show in the next section, this search process converges quickly to the best configuration since the search is driven by the estimation models, differently from the previous proposal in [19] where the local search process was performed by mutating either s or f by a step.

In the `best` state, the search process is stopped while the most performance-demanding application is still checked for QoS satisfaction. If this check on QoS requirement fails (by any change in the external environment complexity or internal application workload), the controller goes back to `apply` state with maximum decay factor γ to be able to quickly recover from that throughput violation.

A particular situation is when all applications leave the system; in such a case, during the Observe PLPT and QoS were set to, zero and one, respectively. This causes the QoS constraint $(\mathbf{K}[s, f][PLPT] \leq PLPT_{ref})$ to be true for all configurations of $\mathbf{K}$, and `GetMinEnergyConfig` function finds the configuration that just minimizes the energy consumption; indeed, it consists in setting the speed to the one that minimizes mechanical energy and the frequency to its minimum value. After the speed is optimized, the controller goes to `best` state after (at most) two control cycles, where no further actions is performed afterwards.

Act (Lines 48–49): The identified optimal configuration is enforced on the robot's actuators.

V. Experimental Evaluation

In this section, we discuss the experimental evaluation of the proposed approach presenting first the experimental setup and then the obtained results. Finally, Section V-C discusses the generality of the proposed approach and presents possible future extensions.

A. Experimental Setup

We have implemented a prototype of the proposed controller in C++ as a package of ROS to enable motor control and monitoring, and data acquisition from the camera. The communication channel to transmit performance measures from the applications to the controller has been implemented by means of Linux shared memory, similarly to [8]; Linux interface has also been used to control DVFS. Finally, for experimental purposes, the controller also logs in the knowledge matrix additional metrics for i) estimated mechanical and computing powers, ii) the throughput (processed batch per second) of the most challenging application. The prototypes counts approximately 5100 lines of code.

We have run the proposed control algorithm for the following four different external environments, i.e., i) low, ii) moderate, iii) high, and iv) variable complexities. This categorization primarily depends on the rate of events generated by the camera in an environment, which is influenced by several factors, including the number of present obstacles, scene contrast, and temporal resolution. The number of generated events increases according to the increase of the complexity; Fig. 10 presents the histogram of events per batch in the three considered scenarios. #Samples represents the count of occurrence of batches with specific number of events observed during the experiment. We can characterize these three histograms for the different complexity environments by interpolating on a normal distribution; the results are three distributions having means and standard deviations (μ, σ) of (12.23k, 2.31k), (16.64k, 4.07k), and (30.10k, 7.50k), respectively. A higher mean value in the normal distribution indicates that the robot has encountered a

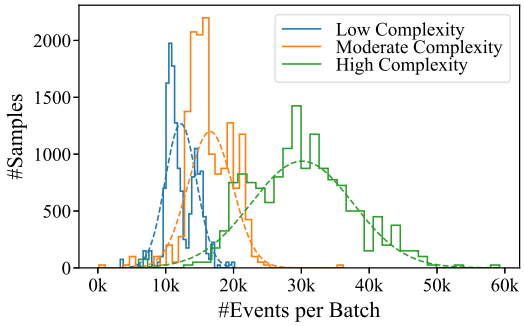

Fig. 10. Histogram of the number of events per batch in a 500-m track for the various environment complexities. Dashed lines represent the corresponding interpolation on normal distributions.

more diverse environment, where the event camera produces a greater number of events, signifying a more complex environment. The variable complexity case contains transition from low-complexity environment to high-complexity and vice versa, at $t \approx 85$ s and $t \approx 135$ s, respectively. The three applications enter and exit the system sequentially in a manner unknown to the controller. The corner detection application is created at the beginning of the experiment and continues running until the end of the mission. This is because it provides essential input features for the upstream higher level applications that are used for robot's operation. The other two applications, namely image reconstruction and corner detection with filtering, enter after a delay of 20 and 40 s, respectively, and leave the system with the same manner. This will synthetically mimic the dynamic compute workload in a real robotic scenario in which various applications may be required to run temporarily due to different internal or external environmental triggers. The experimental campaigns demonstrate the system behavior and performance of the proposed control algorithm for these different environmental scenarios along with discussion on its effectiveness, settling time and execution time, and comparison of PLPT and batch per second criteria for measuring application execution performance.

To show the performance of our proposed approach, we have compared it with the following three different control strategies:

i) *Average Speed and Maximum Frequency (AS-MF)*, which is a constant strategy that keeps the speed constant on its average value and frequency on its maximum value (2.5 m/s and 2 GHz, respectively, in the considered prototype);
ii) *Separate Optimization (SO)* of mechanical and computational parts of the robot, in which speed is kept constant to the one that minimizes mechanical energy (2 m/s) found based on design-time analysis shown in Fig. 3, and frequency is managed separately by an adaptive runtime controller proposed in [8];
iii) *Hill Climbing (HC)* strategy that manages mechanical and computational parts coordinately in an apply-and-measure way, proposed in [19].

The internal models utilized here in our proposed approach have been initialized to the one fitted to offline measurement data at design-time; then, they have been tuned at runtime based on online measurements by minimizing the regularized least squares error on online measurements. This is implemented as solving a linear system of equations using QR decomposition [41], by using Eigen C++ library.[6] The parameters of the initial mechanical model are presented in Table I. The mean absolute error of the initial offline mechanical and computational power models on out-of-sample test data is 1.92 and 0.32 W, respectively; correspondingly, the Meam Relative Error (MRE) is 16.30% and 4.39%, respectively; such values, and in particular, MRE of the mechanical model, are in line with the ones obtained in past works [18].

TABLE I
VALUES OF PARAMETERS USED IN MECHANICAL POWER MODELING

Parameter	m	C_r	C_v	C_d	A_f	ρ
Value	3.7 kg	0.031 N	0.023 N·s/m	0.68	0.1 m^2	1.2 kg/m^3

B. Experimental Results

Fig. 11 demonstrates the control actuators applied on the robot, i.e., speed and frequency, along with the system response i.e., energy and power consumption, and throughput of the most critical application over time for a 500-meter-long track in four scenarios with low, moderate, high, and variable environment complexities. The mission end time is also reported inside the plots with a vertical colored line. For the sake of presentation, the power, energy per meter, throughput and PLPT measures are smoothed by averaging over a moving window of size equal to 3 samples to reduce the oscillations. We examine the plots in accordance with each individual constraint as outlined in the following.

Energy consumption: It can be seen in Fig. 11 that when the controller starts the operation, in our approach, a rapid jump happens from the initial configuration of the actuators to the optimal configuration. This results in a fast energy per meter drop in the first few seconds in all four environmental scenarios that are shown in Fig. 11(a)–(d). This is due to the accurate embedded models that enable the search algorithm to determine the energy-optimal configuration among all available configurations. As it is shown in the magnified plot on top of Fig. 11(a), our proposed controller achieves lower energy consumption per meter than all other compared strategies when they are stabilized. At the settled state, the AS-MF strategy yields highest energy consumption per meter due to its nonoptimal speed for the mechanical part and maximum usage of the computing resources. Notably, the SO strategy does not reach to minimum energy consumption level, since it tunes the mechanical and computational parts separately. Furthermore, although the HC strategy can eventually reach close to the minimum energy consumption, it fluctuates around it and also suffers from slow convergence problem with a long settling time. This will also increase its total energy consumption. Moreover, HC follows a trial-and-error policy which causes a high number of fluctuation in the actuators, and therefore, in the energy and power consumption as well. Such occurrence of fluctuations clearly is the major disadvantage of HC approach

[6][Online]. Available: http://eigen.tuxfamily.org

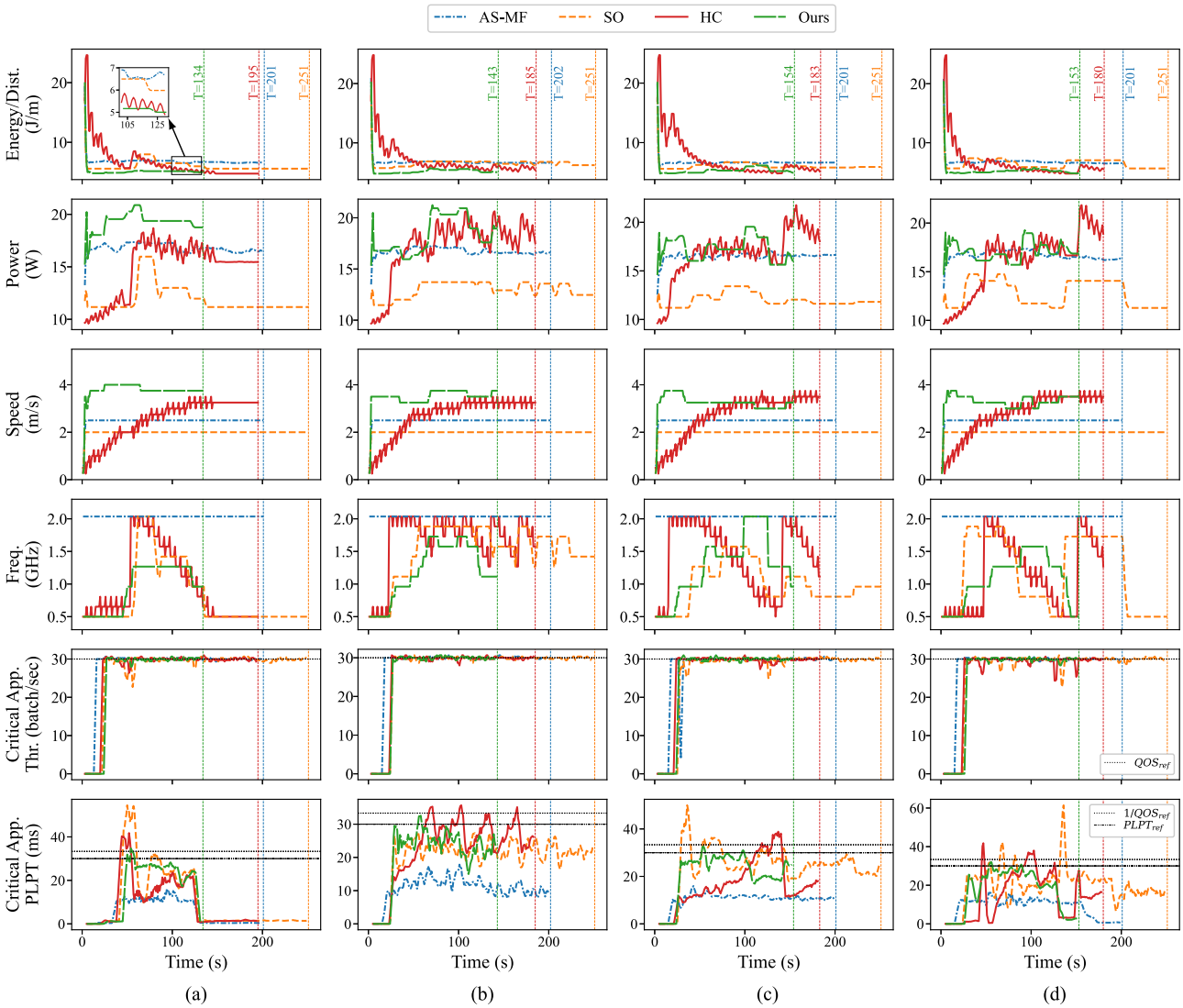

Fig. 11. Comprehensive demonstration of system behavior and control actions over time for a 500-meter-long track, a comparison of methods on four different environment complexities. Note fast settling of our controller to the optimal configuration (and minimum energy level) in a few first seconds, in contrast to slow trial and error of HC. The required QoS threshold and corresponding $PLPT_{ref}$ are shown as horizontal dashed black lines. (a) Low complexity. (b) Moderate complexity. (c) High complexity. (d) Variable complexity.

which, besides prolonging the settling time, may even damage the onboard electric circuits in the long-term. Finally, it can be seen in Fig. 11(d) that also in the variable scenario our approach, in addition to energy minimization, maintains satisfactory throughput, while HC and SO are perturbed by the external environment change.

Fig. 12 shows the overall energy consumption and total mission time of the robot moving in a straight path with different lengths and complexities, which is controlled by our proposed method versus the baselines. Evidently, the energy consumption in our method is less than all other baselines in all scenarios. Notably, HC performs poorly in the short distances because of its slow trial-and-error settling process, which causes the robot to operate in nonoptimal configurations for a long period of time after starting the mission. In the 100 m track [see Fig. 12(a)], for example, our proposed approach outperforms HC with respect to the overall energy consumption by 36.34% improvement that is because of the fast settling time of our approach. However, in longer tracks, our method performs slightly better than HC, with relative improvement of 8.22% in 2500 m track, for example [see Fig. 12(d)]. This is because, while the external complexity is approximately constant, the effect of the transition from initial configuration to optimal configuration mitigates (being smaller portion w.r.t. total mission time), thus, HC and our method provide comparable overall results. In addition, SO yields 16.43% and 17.50% higher energy consumption than our method in 100 m and 2500 m tracks, respectively. This is because of the setup of nonoptimal configuration in SO. Averaged over all track lengths and environment complexities, we improve the energy consumption of AS-MF, SO, and HC by 23.94%, 17.43%, and 18.45%, respectively. Finally, Fig. 12 also demonstrates the total spent time for the methods. Notably, with our method the robot finishes the track faster than other methods.

Mission time: It can be noticed in Fig. 11 that our proposed method selects different speed values for different environment complexities, thus leading to an overall mission time that is not

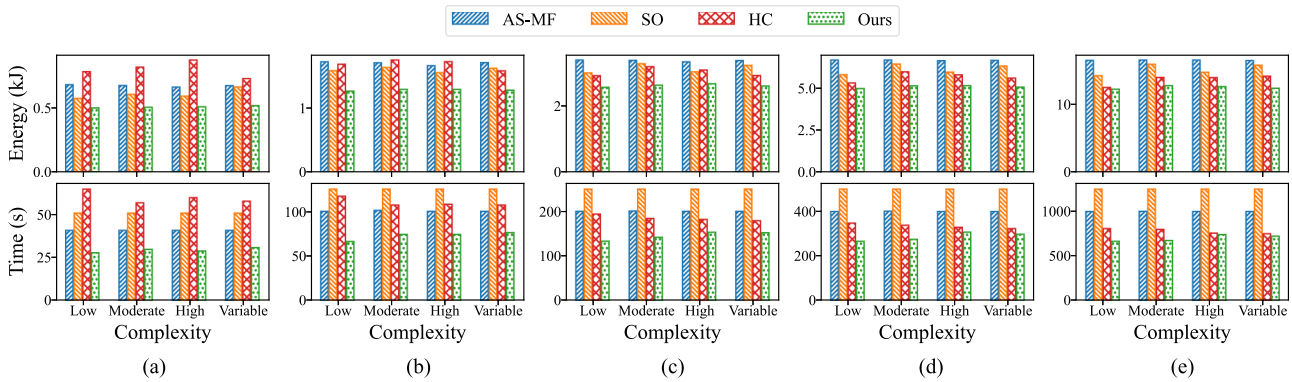

Fig. 12. Energy consumption and mission time for different track lengths and environment complexities. (a) 100 m. (b) 250 m. (c) 500 m. (d) 1000 m. (e) 2500 m.

constant, but depends on the environmental conditions. For example, in the low-complexity environment [see Fig. 11(a)], since the robot does not need to process much information to satisfy the required throughput, it can tune the speed higher than while facing a high complexity. This shows how the robot intelligently adapts the speed according to the environment complexity to avoid unnecessary high computational power in lower complexities. In high-complexity environment [see Fig. 11(c)], on the other hand, a lower speed is selected to be able to process all the incoming information. The AS-MF and SO methods maintain constant mission times for various complexities as they do not adjust the robot's speed at runtime; it remains constant (slight variation is due to initial acceleration). In contrast, our method and HC dynamically adapt the speed to different environmental complexities, resulting in varying mission times. Notably, our method consistently achieves faster mission completion times than HC across all scenarios. This performance advantage arises because HC requires a significant amount of time to iteratively determine the optimal speed through reactive trial and error, whereas our proposed method predicts and proactively adjusts the speed, avoiding any time wastage.

Application throughput and PLPT: It can be seen in the fifth row of Fig. 11 that our proposed approach keeps the throughput up to the acceptable requirement (QoS equal to 30 batch/sec) in all complexity conditions. In course-grained analysis, we can notice that all approaches reach the required throughput. There are some oscillations especially at the beginning due to the various applications' enter perturbing the system's status; indeed, the goal of the proposed runtime approach and other competitors is to react and quickly converge to satisfy the throughput requirement. In this perspective, Fig. 13 compares these oscillations in more details by reporting the percentage of time in the experiment each approach violates the required throughput within a 5% tolerance bound (it is necessary to compensate the intrinsic oscillations of the dynamic system). SO and HC present relevant percentage of violations (up to 4%); SO demonstrates that separate optimizations are not effective while HC that actuating each explored configuration takes time. On the other hand, AS-MF has a 0% of violation, at the cost of consuming more energy than the proposed approach, as discussed before. In between, the proposed approach presents a percentage of violations lower than 1% that is affordable. In fact, this 1% violation is intrinsic in the runtime adaptive nature

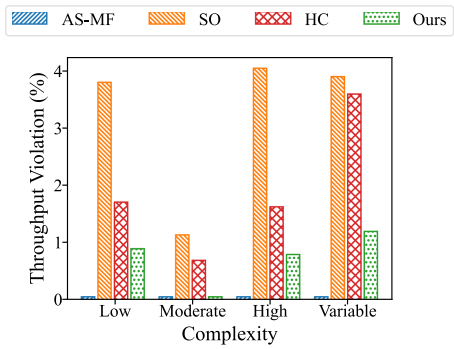

Fig. 13. Percentage of time the throughput is violated (considering a 5% tolerance bound).

of the proposed feedback-based control approach that adapts to internal workload changes such as applications' enter. This value is also confirmed in the variable complexity scenario where further external perturbations due to changes in environment complexity cause a slightly higher percentage.

The PLPT measurements for the most critical application are also reported in the last row of Fig. 11 for this experiment. We set parameter $\kappa = 0.9$ for computing $PLPT_{ref}$ to act before the actual throughput violation happens. This conservative behavior increases energy consumption, as it forces the controller to choose higher frequency levels (or lower speeds) to meet a stricter constraint. The horizontal dashed lines in the plots specify both the $PLPT_{ref}$ (bottom line) and also PLPT absolute limit equal to $1/QoS_{ref}$ (top line). It can be observed that our proposed method effectively maintains the highly variable PLPT below the upper bound $PLPT_{ref}$. Moreover, as can be seen, application throughput (processed batch per second) is closely dependent on the measured PLPT value, i.e., when PLPT is higher than the absolute limit, throughput drops. However, batch per second cannot encompass the dynamics when computing resources are over-provisioned because of its saturation. The saturation issue of batch per second measure is further demonstrated in Fig. 14 compared to PLPT for different frequency and speed levels. It can be seen that the PLPT, as is expected, decreases with frequency increase, while batch per second saturates to $R = 30$ when frequency is increased. This reasonable trend in PLPT enables computing resource management in real-time applications with online data streams. PLPT is used in our proposed control

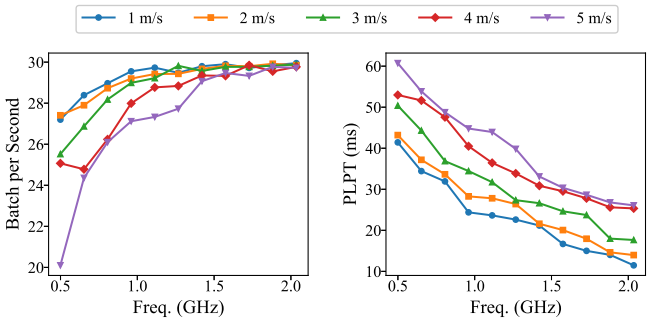

Fig. 14. Average value of batch per second and PLPT as indicator of application performance versus frequency level, for different speeds. Note the saturation of batch per second at high frequencies versus meaningful decreasing trend in PLPT. $R = 30$.

TABLE II
TIME TO SETTLE TO THE OPTIMAL CONFIGURATION

Complexity	Settling time (s) Method	
	HC	**Ours**
Low	141	11
Moderate	103	5
High	94	6

algorithm and SO to instantly find the required frequency that satisfies the reference PLPT, while with the saturated trend in batch per second, it could only be performed in a step-by-step manner.

Settling time: The transition time from the initial configuration to the most energy efficient one affects greatly the energy consumption and operation performance, especially in short distances and highly variable environmental complexities. It will be also important when the robot needs to change its path regularly because of e.g., obstacle avoidance. Therefore, we compared the settling time of our approach with HC. Here, the settling time is defined as the point at which the robot's energy consumption first enters a 5% bound around the optimal value. Do note that, since AS-MF is a static method, there is no settling time to analyze. Similarly, in SO, the speed is fixed to a design-time optimized value and only frequency is adjusted at runtime; therefore, it is not comparable with joint optimizations performed by HC and our one.

When considering the results of the considered experiment, we noticed that HC was not able to converge before the subsequent application enter/exit occurred; as a consequence, it is not fair to compare settling time in this context. Thus, we ran a new experiment with all the three applications starting at time 0 s. Table II shows the settling time of the two approaches for three environmental complexities. Our proposed method acts approximately 16 times faster than HC, on average, thanks to the embedded predictive models.

Execution time of the controller: The average execution time of the proposed controller loop is 42.12 ms; this is a negligible time when considering that the activation period is equal to 1 s. Particularly, runtime training of the internal models takes 15.64 ms, total inference time of the linear models for the whole search space consisting of a knowledge matrix with 220 different speed/frequency configurations is 1.45 ms, and the rest is due to sensor delays, application monitoring and also for applying the speed, frequency. As can be noticed, the execution time for the algorithm is acceptable for a macroscopic size robot with a normal change of speed rate.

C. Discussion and Future Work

This work proposing a novel approach for runtime resource management considering both computing and mechanical energy optimization has targeted an autonomous driving rover using an event camera for vision tasks. After the discussed experimental validation on the selected platform, we would like to remark that the proposed approach is not restricted to such a specific working scenario; at the opposite, we claim that it is generally applicable on a variety of other robotic systems with similar characteristics, and, in particular, if two conditions hold. The first condition pertains to the existence of a substantial application-level interdependency between the computing and mechanical aspects of the robot. For example, in the case here examined, changes in mechanical velocity affect the computational workload required to maintain the throughput of the running applications. This relationship holds true for various types of robotic systems due to their cyber-physical nature where running applications continuously interact with the surrounding environment; examples are vision-guided robotic arms and autonomous vehicles for terrestrial, naval or aerial operations. The second condition motivating the proposed co-optimization approach is the comparability of computing and mechanical contributions to the overall power/energy consumption; indeed, if one of the two contributions is predominant w.r.t. the other one, all runtime decisions can be taken only on the former one while considering the highest performance configuration of the latter, selected at design time. Today, many robotic systems exhibit similar levels of mechanical and computing power consumption due to the increasing prevalence of more intelligent tiny robots, where powerful computing boards are integrated in small robotic platforms.

Based on these considerations, this work suggests multiple avenues for future research aimed at broadening the considered working scenario to demonstrate the general validity of the proposed solution and to extend it to handle additional functionalities and components of the robotic system. At robot architecture level further computing knobs may be taken into account. When considering the computing part, we have already discussed how the past work targeting embedded and mobile devices has considered the mapping on heterogeneous processing units (such as GPUs and asymmetric CPU clusters, each provided with a separate frequency knob) and the tuning of the number of cores to be reserved for each single multithreaded application; all these aspects may be taken into account in the considered broaden scenario of the robotic system. In the same way, multiple motors, arms, and other components may be considered as additional knobs for the mechanical part of the robot.

At application level, we have here focused on a workload composed of a set of concurrent vision applications working with an even-based camera. However, we note that the robot generally runs multiple interconnected tasks, for perception, obstacle avoidance and path planning, arm control, etc., each one of them possibly exposing specific performance, speed, or other types of requirements. The controller proposed in this work is actually ready to work with a workload composing of any other type of applications or tasks, possibly exposing throughput requirements; it interacts with all these running applications to receive their constraints and requirements and to identify an operating configuration capable at satisfying all of them, while minimizing energy consumption. Moreover, if applications expose also speed constraints, they can be easily taken into account by discarding violating configurations during the optimization process. Future work will experimentally demonstrate this generality in interacting with the other robot tasks and addressing their exposed constraints.

While the proposed controller is easily adaptable to various applications, the addition of multiple tunable knobs may raise scalability issues for the proposed approach. In fact, the time complexity of the approach, based on an exhaustive update and scan of the knowledge matrix, grows linearly with the number of configurations composing the search space; therefore, such an approach may become unpractical when considering complex robotic systems. Another future direction is, therefore, devoted to defining more efficient optimization/decision-making approaches possibly utilizing graph-based and machine learning algorithms.

VI. Conclusion

The energy management of mobile robots involves the simultaneous operation of two processes: managing mechanical and computational components in real time. These processes play a crucial role in enhancing the robots' battery lifespan and extending their mission duration. In this article, we demonstrate that independently managing mechanical and computational energy, without considering each other, does not yield the most efficient energy consumption outcomes. Motivated by this insight, we introduce a proactive management approach that adjusts the robot's speed and CPU voltage/frequency concurrently, a control policy we refer to as co-management of mechanical and computational components. We employ an internal prediction model that calculates the optimal speed and CPU voltage/frequency based on the environmental complexities encountered by the robot during its mission. Our experimental results, obtained from a mobile vehicular robot, highlight the effectiveness and practicality of our approach when compared to traditional energy management techniques. For example, our proposed method improves the energy consumption of SO policy by 16.43% and of a preliminary co-management policy by 13.57% in a mid-length track. For short track lengths, our improvement in the energy consumption is up to 36.34%.

References

[1] Intel, "Autonomous mobile robots." Accessed: 25 Jul. 2023. [Online]. Available: https://www.intel.com/content/www/us/en/robotics/autonomous-mobile-robots/overview.html

[2] A. G. Olabi, Q. Abbas, P. A. Shinde, and M. A. Abdelkareem, "Rechargeable batteries: Technological advancement, challenges, current and emerging applications," *Energy*, vol. 266, 2023, Art. no. 126408.

[3] P. Greenhalgh, "ARM, Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7." 2011. Accessed: 4 Mar. 2024. [Online]. Available: https://www.eetimes.com/big-little-processing-with-arm-cortex-a15-cortex-a7/

[4] T. Kundu and I. Saha, "Energy-aware temporal logic motion planning for mobile robots," in *Proc. Int. Conf. Robot. Autom.*, 2019, pp. 8599–8605.

[5] C. Cunningham, J. Amato, H. L. Jones, and W. L. Whittaker, "Accelerating energy-aware spatiotemporal path planning for the lunar poles," in *Proc. Int. Conf. Robot. Autom.*, 2017, pp. 4399–4406.

[6] Y. Mei, Yung-Hsiang Lu, Y. Hu, and C. Lee, "A case study of mobile robot's energy consumption and conservation techniques," in *Proc. Int. Conf. Adv. Robot.*, 2005, pp. 492–497.

[7] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical power management for asymmetric multi-core in dark silicon era," in *Proc. Des. Autom. Conf.*, 2013, pp. 1–9.

[8] D. Angioletti, F. Bertani, C. Bolchini, F. Cerizzi, and A. Miele, "A runtime resource management policy for OpenCL workloads on heterogeneous multicores," in *Proc. Des., Autom. Test Europe Conf. Exhib.*, 2019, pp. 1385–1390.

[9] E. Del Sozzo, G. C. Durelli, E. Trainiti, A. Miele, M. D. Santambrogio, and C. Bolchini, "Workload-aware power optimization strategy for asymmetric multiprocessors," in *Proc. Conf. Des., Autom. Test Europe*, 2016, pp. 531–534.

[10] A. K. Singh, A. Prakash, K. R. Basireddy, G. V. Merrett, and B. M. Al-Hashimi, "Energy-efficient run-time mapping and thread partitioning of concurrent OpenCL applications on CPU-GPU MPSoCs," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, pp. 147:1–147:22, 2017.

[11] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra, "Integrated CPU-GPU power management for 3D mobile games," in *Proc. 51st Annu. Des. Autom. Conf.*, 2014, pp. 1–6.

[12] Y. Mei, Y.-H. Lu, Y. C. Hu, and C. G. Lee, "Energy-efficient motion planning for mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2004, vol. 5, pp. 4344–4349.

[13] S. Shahidinejad, E. Bibeau, and S. Filizadeh, "Statistical development of a duty cycle for plug-in vehicles in a North American urban setting using fleet information," *IEEE Trans. Veh. Technol.*, vol. 59, no. 8, pp. 3710–3719, Oct. 2010.

[14] Q. Yan, B. Zhang, and M. Kezunovic, "Optimization of electric vehicle movement for efficient energy consumption," in *Proc. North Amer. Power Symp.*, 2014, pp. 1–6.

[15] P. Tokekar, N. Karnad, and V. Isler, "Energy-optimal trajectory planning for car-like robots," *Auton. Robots*, vol. 37, pp. 279–300, 2014.

[16] M. Wei and V. Isler, "Energy-efficient path planning for ground robots by and combining air and ground measurements," in *Proc. Conf. Robot Learn.*, 2020, pp. 766–775.

[17] H. Zhang, Y. Zhang, C. Liu, and Z. Zhang, "Energy efficient path planning for autonomous ground vehicles with ackermann steering," *Robot. Auton. Syst.*, vol. 162, 2023, Art. no. 104366.

[18] D. Hong, S. Lee, Y. H. Cho, D. Baek, J. Kim, and N. Chang, "Least-energy path planning with building accurate power consumption model of rotary unmanned aerial vehicle," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 14803–14817, Dec. 2020.

[19] S. A. Mohamed, M.-H. Haghbayan, A. Miele, O. Mutlu, and J. Plosila, "Energy-efficient mobile robot control via run-time monitoring of environmental complexity and computing workload," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 7587–7593.

[20] G. Gallego et al., "Event-based vision: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 1, pp. 154–180, Jan. 2022.

[21] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240× 180 130 dB 3 μs latency global shutter spatiotemporal vision sensor," *IEEE J. Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, Oct. 2014.s

[22] M. Quigley et al., "ROS: An Open-Source Robot Operating System," in *Proc. Int. Conf. Robot. Automat. Open Source Softw.*, 2009, pp. 1–5.

[23] C. Scheerlinck, N. Barnes, and R. Mahony, "Continuous-time intensity estimation using event cameras," in *Proc. Asian Conf. Comput. Vis.*, 2018, pp. 308–324.

[24] S. A. S. Mohamed et al., "Asynchronous corner tracking algorithm based on lifetime of events for DAVIS cameras," in *Proc. Int. Symp. Adv. Vis. Comput.*, 2020, pp. 530–541.

[25] S. A. S. Mohamed et al., "Dynamic resource-aware corner detection for bio-inspired vision sensors," in *Proc. Int. Conf. Pattern Recognit.*, 2021, pp. 10465–10472.

[26] B. Cho, S.-W. Kim, S. Shin, J.-H. Oh, H.-S. Park, and H.-W. Park, "Energy-efficient hydraulic pump control for legged robots using model predictive control," *IEEE/ASME Trans. Mechatron.*, vol. 28, no. 1, pp. 3–14, Feb. 2023.
[27] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal, "Application heartbeats: A generic interface for specifying program performance and goals in autonomous computing environments," in *Proc. Int. Conf. Autonomic Comput.*, 2010, pp. 79–88.
[28] S. Chinchali et al., "Network offloading policies for cloud robotics: A learning-based approach," *Auton. Robots*, vol. 45, no. 7, pp. 997–1012, 2021.
[29] M. Penmetcha and B.-C. Min, "A deep reinforcement learning-based dynamic computational offloading method for cloud robotics," *IEEE Access*, vol. 9, pp. 60265–60279, 2021.
[30] R. Aldana-López, R. Aragüés, and C. Sagüés, "Latency vs precision: Stability preserving perception scheduling," *Automatica*, vol. 155, 2023, Art. no. 111123.
[31] C. C. Ooi and C. Schindelhauer, "Minimal energy path planning for wireless robots," *Mobile Netw. Appl.*, vol. 14, pp. 309–321, 2009.
[32] Y. Yan and Y. Mostofi, "Co-optimization of communication and motion planning of a robotic operation under resource constraints and in fading environments," *IEEE Trans. Wireless Commun.*, vol. 12, no. 4, pp. 1562–1572, Apr. 2013.
[33] D. B. Licea, M. Bonilla, M. Ghogho, S. Lasaulce, and V. S. Varma, "Communication-aware energy efficient trajectory planning with limited channel knowledge," *IEEE Trans. Robot.*, vol. 36, no. 2, pp. 431–442, Apr. 2020.
[34] Y. Zeng, J. Xu, and R. Zhang, "Energy minimization for wireless communication with rotary-wing UAV," *IEEE Trans. Wireless Commun.*, vol. 18, no. 4, pp. 2329–2345, Apr. 2019.
[35] Z. Liu, B. Wu, J. Dai, and H. Lin, "Distributed communication-aware motion planning for networked mobile robots under formal specifications," *IEEE Trans. Control Netw. Syst.*, vol. 7, no. 4, pp. 1801–1811, Dec. 2020.
[36] A. Muralidharan and Y. Mostofi, "Communication-aware robotics: Exploiting motion for communication," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 4, pp. 115–139, 2021.
[37] K. Byus, "Observe, orient, decide, act: A subjectivist model of entrepreneurial decision making," *J. Managerial Issues*, vol. 30, no. 3, pp. 349–362, 2018.
[38] A. Miele, L. C. H. Zárate, C. Bolchini, and J. E. Ortiz Trivino, "A runtime resource management and provisioning middleware for fog computing infrastructures," *ACM Trans. Internet Things*, vol. 3, no. 3, 2022, Art. no. 17.
[39] A. Mohammed, B. Schmidt, L. Wang, and L. Gao, "Minimizing energy consumption for robot arm movement," *Procedia CIRP*, vol. 25, pp. 400–405, 2014.
[40] S. A. Mohamed, M.-H. Haghbayan, M. Rabah, J. Heikkonen, H. Tenhunen, and J. Plosila, "Towards dynamic monocular visual odometry based on an event camera and IMU sensor," in *Prof. Int. Conf. Intell. Transport Syst.*, 2020, pp. 249–263.
[41] E. F. Van de Velde, *QR-Decomposition*. New York, NY, USA: Springer, 1994, pp. 125–140, doi: 10.1007/978-1-4612-0849-5_5.

Sajad Shahsavari (Student Member, IEEE) received the B.Sc. degree in computer engineering from the Amirkabir University of Technology, Tehran, Iran, in 2014 and the M.Sc. degree in artificial intelligence from the Sharif University of Technology, Tehran, Iran, in 2017. He is currently working toward the Ph.D. degree in machine learning-based digital twins for autonomous robots with the University of Turku, Turku, Finland.

He is currently a Researcher with Computational Engineering and Analysis (COMEA) Research Group, Turku University of Applied Sciences, Turku, Finland. His research interests include deep neural networks, time-series prediction, reinforcement learning, and data analysis.

Hashem Haghbayan (Member, IEEE) received the B.A. degree in computer engineering from the Ferdowsi University of Mashhad, in 2006, the M.Sc. degree in computer architecture from the University of Tehran, Tehran, Iran, in 2009, and the Ph.D. with honors in information and communication technology from the University of Turku, Turku, Finland, in 2018.

He is currently a Senior Research Fellow and an Adjunct Professor (docent) in embedded intelligent systems with the Department of Computing, Faculty of Technology, University of Turku (UTU), Turku. From 2018 to 2021, he has been a Postdoctoral Researcher with the Department of Computing, Faculty of Technology, University of Turku. His research interests include machine learning, autonomous systems, high-performance energy-efficient architectures, and on-chip/fog resource management.

Antonio Miele (Senior Member, IEEE) received the M.Sc. degree in computer engineering from Politecnico di Milano, Milan, Italy, and the M.Sc. degree in computer science from the University of Illinois at Chicago, Chicago, IL, USA, both in 2006, and the Ph.D. degree in information technology from Politecnico di Milano, Milan, Italy, in 2010.

He is currently an Associate Professor with Politecnico di Milano. He is coauthor of more than 80 peer-reviewed publications. His main research interests include related to design methodologies for embedded systems, in particular fault tolerance and reliability issues, runtime resource management in heterogeneous multi/many-core systems, and FPGA-based systems design.

Eero Immonen (Member, IEEE) received the two doctoral degrees one in robust control from Tampere University of Technology, Tampere, Finland, in 2006, and another in computational design from LUT University, Lappeenranta, Finland, in 2024.

He is currently a Principal Lecturer and leader of the Computational Engineering and Analysis (COMEA) Research Group, Turku University of Applied Sciences, Turku, Finland. With 10 years of industry experience, he led more than 100 digital twin simulation projects worldwide as R&D Manager with Process Flow Ltd., Reading, U.K. His work, including fluid-structure interaction and CFD-based shape optimization, has been authored or coauthored in academic literature, applying computational methods to real-world engineering problems.

Juha Plosila (Member, IEEE) received the Ph.D. degree in electronics and communication technology from the University of Turku (UTU), Turku, Finland, in 1999.

He is currently a Full Professor with the Department of Computing, UTU, leading there the Robotics and Autonomous Systems Unit (RAS) and research group called Autonomous Systems Laboratory (ASL). He has a strong research background in adaptive multiprocessing systems and platforms, development of self-aware multiagent monitoring and control architectures for massively parallel systems, machine learning and evolutionary computing-based methods, as well as application of heterogeneous energy efficient architectures to new computational challenges in the fields of cyber-physical systems and Internet-of-Things, with a recent focus on fog/edge computing (edge intelligence) and autonomous multirobot systems.