Energy-Efficient Mobile Robot Control via Run-time Monitoring of Environmental Complexity and Computing Workload

Sherif A.S. Mohamed[1], Mohammad-Hashem Haghbayan[1,3], Antonio Miele[2], Onur Mutlu[3], and Juha Plosila[1]

***Abstract*— We propose an energy-efficient controller to minimize the energy consumption of a mobile robot by dynamically manipulating the mechanical and computational actuators of the robot. The mobile robot performs real-time vision-based applications based on an event-based camera. The actuators of the controller are CPU voltage/frequency for the computation part and motor voltage for the mechanical part. We show that independently considering speed control of the robot and voltage/frequency control of the CPU does not necessarily result in an energy-efficient solution. In fact, to obtain the highest efficiency, the computation and mechanical parts should be controlled together in synergy. We propose a fast hill-climbing optimization algorithm to allow the controller to find the best CPU/motor configuration at run-time and whenever the mobile robot is facing a new environment during its travel. Experimental results on a robot with Brushless DC Motors, Jetson TX2 board as the computing unit, and a DAVIS-346 event-based camera show that the proposed control algorithm can save battery energy by an average of 50.5%, 41%, and 30%, in low-complexity, medium-complexity, and high-complexity environments, over baselines.**

I. Introduction

Constrained energy is a major challenge in the field of mobile robotics. The research focus has been mainly on optimizing the *motion planning* of a robot and *kinematic energy* [1], [2]. However, kinematic energy consumption is not the only source of energy drain. A mobile robot, as a cyber-physical device, also contains a *cyber-part* beside the *physical part*, such as on-board electrical devices, micro-controllers, and sensors, each of which consumes power and contributes to the overall energy consumption [3]. For example, the execution of heavy vision-based applications that use onboard computing units and sensors drains a significant portion of the available source of energy. *Computing energy consumption* of the cyber-part in a robot is one of the significant portion of the energy consumption in a robot that should be considered while optimizing the energy.

There have been several studies to reduce the power/energy consumption of the computing units for cyber-physical systems (e.g., [4], [5], [6], [7]). In recent mobile robots computing power management units significantly reduces the computing power/energy consumption via dynamic voltage and frequency scaling (DVFS), power gating (PG) and/or other hardware knobs, whenever full performance is not required (e.g., [8], [9], [10], [11], [12]).

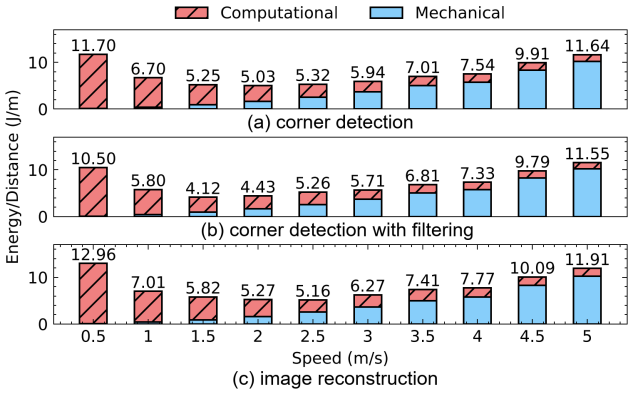


Fig. 1: Comparing average power consumption of computational and mechanical parts of a mobile robot for three vision-based applications at various robot movement speed.

Prior techniques try to improve the power/energy consumption of the cyber-part and/or physical-part of the robotic system separately and independent from each other, without considering the co-relations between these two parts. One of these co-relations is the effect of robot mechanical *control decision* on the *computing workload*. In a mobile robot with a normal frame-based camera as a sensor, the quality of an image of a scene captured at different speeds is not the same [13], [14]. This results in *workload variation* for processing different captured images of *the same scene* at *different speeds* of the robot[1]. Another example is a mobile robot with an *event-based camera* as the sensor. Here, both the *environmental complexity* and *the robot speed* affect the number of generated events at each instance of time and consequently the event processing workload [15], [16].

As an example, Figure 1 shows the energy consumption per unit distance (referred to as *locomotion cost* in literature [17], [18], [19]) for different speeds of a mobile robot with an event-camera sensor, when running three different image processing tasks on the event data[2]. The Power Management Unit is responsible for reducing the computing power consumption as much as possible, while avoiding unacceptable throughput loss, by manipulating the voltage and frequency of the processing unit. The overall drained energy of the

[1]Sherif A.S. Mohamed, Mohammad-Hashem Haghbayan, and Juha Plosila are with Autonomous Systems Laboratory (ASL), University of Turku, 20500, Turku, Finland. {samoha, mohhag, juplos}@utu.fi

[2]Antonio Miele is with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, 20133 Milano, Italy. antonio.miele@polimi.it

[3]Mohammad-Hashem Haghbayan and Onur Mutlu are with ETH Zürich, 8092, Zürich, Switzerland. omutlu@gmail.com

[1]This happens due to changes in the captured image quality.

[2]Details on our applications are provided in Section II-A

robot is calculated as follows:

$$E = \int_0^T \left(p^v_{motor} + p^e_{cpu}\right) dt \quad (1)$$

where p_{motor}, p_{cpu} are respectively the power of the mechanical motors, $motor$, and computing unit, cpu, which are functions of robot speed, v, and captured events, e, respectively. The integral spans over the entire duration of the mission T to compute the energy consumption.

Figure 1 shows that increasing the speed to the maximum possible value does not necessarily result in an optimal energy consumption. In fact, if the robot's mechanical controller uses the *highest possible speed* to reduce the movement time, in addition to the requirement for high instantaneous power of the motors, computing a *large number of generated events* per time unit consumes a high amount of power. Having a low speed for the robot saves mechanical energy significantly. On the other hand, computing energy becomes dominant despite a lower frequency of generated events. The reason for this is that, by decreasing the speed and increasing the overall time of the travel, most of the energy consumed by the processing units is *static energy* that is used constantly during the waiting periods of the units. It can be seen that the lowest energy consumption is obtained at *a specific speed* of the robot. Another important fact that can be observed is that the most energy-efficient speed for different applications are not necessarily the same value. This is due to the different strategies in the Power Management Unit for different types of software executing on the computing platform. Therefore, to obtain energy-efficient control of the robot, both computational and mechanical controllers must be tuned *together at run-time*, to dynamically adjust the robot speed and voltage/frequency of the computing unit.

Given the above motivation, this paper presents a novel energy-efficient control technique to manipulate the mechanical and the computational actuators of a mobile robot together to obtain the best possible overall energy consumption. The actuators are the voltage of the motors, to change the speed, and the voltage/frequency of the CPU. To manipulate the actuators, the instantaneous power of the mechanical and computational parts are fed back to the controller. Then, the controller performs a fast hill-climbing algorithm at run-time to *iteratively improve* the system configuration to minimize the overall energy consumption.

In summary the novel contributions of the paper are:

- A controller framework capable of co-managing mechanical and computational parts of the robot system executing vision-based applications.
- A control policy to select the best motor speed and CPU voltage level configurations to guarantee application throughput while minimizing energy consumption.
- An experimental session showing an improvement in energy saving up to 50% w.r.t. the baseline solutions.

The rest of the paper is organized as follows. Section II provides the working scenario and the motivation for the proposed controller. In Section III, the proposed approach is described. Section IV provides the experimental results.

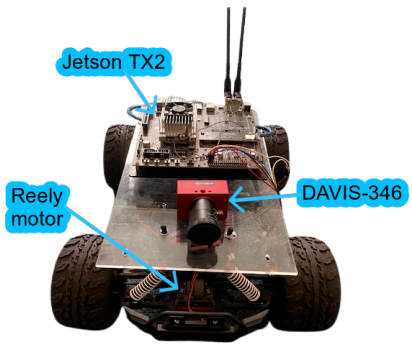


Fig. 2: Platform setup.

Section V is devoted to the related works. Finally, Section VI concludes the paper with some discussion and comments on future work.

II. Working scenario and motivation

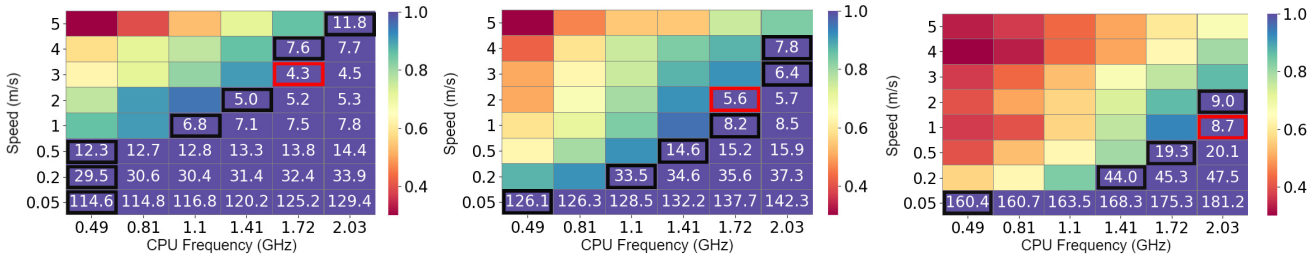
We present the overall robot architecture and a motivating example that shows how in different environmental situations the most energy-efficient values for voltage/frequency and robot speed changes. This motivational example demonstrates an overview of the dynamics of the system in different *environmental complexities* and motivates the proposed approach for an energy-efficient run-time controller.

A. Working scenario

System architecture. The platform in this work consists of three main parts, i.e., overall robot architecture, the architecture of the computing unit, and vision-based applications. Each of them is separately explained below.

Overall robot architecture. This includes the physical parts of the platform, such as an embedded system, an event-based camera, and a brushless dc motor, as shown in Figure 2. The Jetson TX2 board is a super power-efficient embedded board with two clusters: quad-core ARM Cortex-A57 and dual-core NVIDIA Denver. The event-based camera, i.e., DAVIS-346, can capture both intensity images and a stream of asynchronous events with a high temporal resolution of up to 10 million events in a second [20], [21]. A DC brushless motor is used to power up the robot. The motor has 3000 kV which can spin at 50000 RPM. The power supply of motors is calculated based on the current measurement circuit that measures the instantaneous electric current toward the motors.

The architecture of the CPU controller. For executing the applications, a mapping unit, as the middleware proposed in [10], allocates the tasks on the various cores inside the CPU clusters. The cores can run at a maximum frequency of 2 GHz. However, run-time DVFS is capable to reduce the voltage/frequency of the CPU cluster at various intermediate steps down to 300 kHz. The middleware is capable to report the throughput for each application that helps to show if the application is loosing some parts of the data due to overload or not. Throughput value ranges between 0 to 1



(a) Low-complexity environment (b) Medium-complexity environment (c) High-complexity environment

Fig. 3: The parameters configuration space for the robot while running three applications with different environmental complexity. The color of each cell represents the achieved application throughput (spanning from 0 to 1), while for valid configurations (i.e. with throughput equal to 1) the reported number is the energy consumption in Joule.

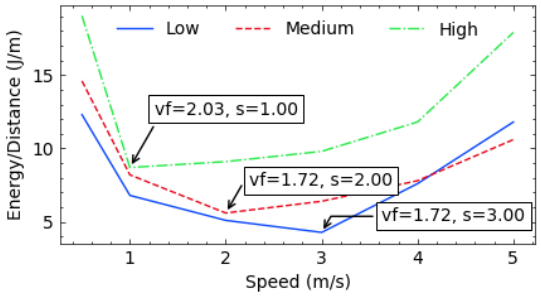


Fig. 4: Energy consumption per meter of the robot in three different environmental complexities, i.e, low, medium, and high-complexity.

where reported value 1 means all the captured data are fully being processed.

Application. Three different applications with different levels of complexity are used, that are image reconstruction, corner detection, and corner detection with filtering. The image reconstruction application [22] is considered to have low complexity and it can produce intensity-based images from a pure event stream using a high-pass filter. The second application is a corner detector [23], which extracts corners from a stream of events. The application is considered to have a high level of complexity since it requires computing the eigenvalues for each incoming event to decide whether the event is a corner or not. The corner detection application is used with a three-level filtering [24] to reduce the computational complexity and obtain a medium level of complexity.

B. Motivating discussion

Figure 3(a-c) shows different values of the robot's overall energy and throughput levels while applying different (CPU voltage/frequency, robot speed) pairs for three kinds of environments, i.e., 1) low-complexity, 2) medium-complexity, and 3) high-complexity. It is worth noting that the throughput and energy of the mobile robot are measured for each point at run-time by enforcing the specific configuration. The vertical axis in the figure shows the speed of the robot and the horizontal axis shows the CPU voltage/frequency level[3]. The color of each (CPU voltage/frequency, speed) point shows the throughput of the applications, where the value 1.0, i.e., dark purple color, represents full throughput.

As it can be noticed from the colors in the figure the 3D surface representing the throughput has a hill shape where the only global peak is placed on the right-bottom side of each plot: in fact, higher CPU frequencies provide larger computation power and lower motor speeds reduce the number of events to process per unit of time. Moreover, throughput saturates to 1.0 since the application is designed to do not over-perform when enforcing higher CPU frequency levels or lower motor speed. Therefore, the top of the hill presents a plateau of purple points. When analyzing the three graphs together it can also be noticed that plateaus have not the same extension but at the opposite with the increase of the environmental complexity, the plateau shifts to the right-bottom side. This comes from the fact that the number of the generated events increases and more powerful processing is needed for obtaining real-time full-throughput outcomes.

We show inside each box the overall energy consumption of the robot. Since only the full-throughput cases are acceptable, energy values are reported for the areas with full-throughput, i.e., dark purple points. Here there is an opposite trend w.r.t. the throughput one. In fact, energy consumption decreases when motor speed is higher (since the overall trip takes a shorter time) and the CPU frequency is lower since consuming less static power consumption. Therefore, the energy surface forms a slope in the left-top direction.

Given these considerations, we state that the most energy-efficient (CPU voltage/frequency, robot speed) pair should be found at the frontier of the purple area. Figure 4 reports only the points on the frontier of the plateau in each one of the three cases. In particular, each line is composed of the points highlighted with the black box in Figure 3 (without loss of generality, to simplify the chart, we took a single point for each speed value). From the plot, we confirm the trend, and in particular the fact that there is a single global minimum in each case and no other local minima.

[3]Figures report only the frequency values since it is the knob actually tuned by the operating system; then the hardware DVFS controller selects the corresponding voltage level.

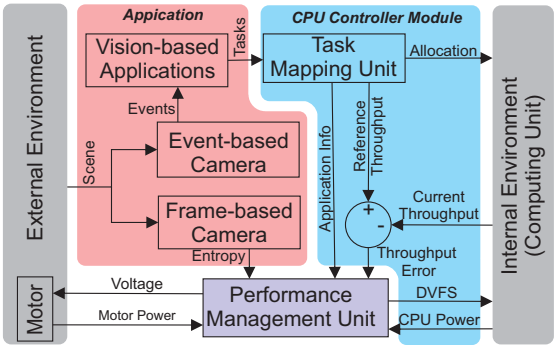


Fig. 5: The system architecture of the proposed cyber-physical automation for robots.

Moreover, since the plateau and the corresponding frontier change with the environmental complexity, it is not possible to determine at design time a unique best configuration for any working scenario/application. As a consequence, we here propose a control approach capable of optimizing the robot configuration at run-time by properly exploring the peculiarities of the considered problem space.

III. The Proposed Controller

A. Overall organization of the controller

The overall structure of the proposed throughput-aware run-time controller is depicted in Figure 5. The unit is hosted on the Performance Management Unit which controls the system based on the feedback from the Internal Environment, which is the computing unit, and External Environment, which is the environment of the mobile robot. The entropy of the vision data has been used as the metric for environmental complexity. The entropy is computed from intensity images captured by a frame-based camera.

We borrowed a fairly standard internal organization (as in [10]) for the application mapping unit. Different types of applications, e.g., image reconstruction and corner detection, are given to the Application Mapping Unit in run-time for execution. Each application has a pre-specified throughput requirement that is given to the mapping unit at design time. The Application Mapping Unit allocates the applications on the CPU cluster (ARM Cortex-A57 or NVIDIA Denver) selected as the best one at design time (future work is devoted to the automatic selection of the CPU cluster at run-time without any pre-profiling). This unit also provides information about the current running applications, *Application info*, and the required throughput for each application, i.e., *Throughput error* λ_e, for the Performance Management Unit. The observations for the Performance Management Unit are 1) current power consumption of the CPU, 2) current power of the robot motors, 3) entropy of the vision sensors, and 4) the difference between the current throughput of the applications and the desired throughput, i.e., *Throughput error*. Based on the feedback received from the external and the internal environments, the Performance Management Unit regulates the actuation values of the CPU voltage/frequency and motor voltage.

During the system operation, run-time events are produced via the Event-based Camera and are passed to the Computing Unit. Based on the types of running applications, the Computing Unit processes the events. The number of events relates to two factors: 1) the environmental complexity and 2) the rate of the change in the environmental information. The environmental complexity is measured through the entropy of the captured frame-based image in run-time. The rate of the change in the environmental information is measured by the speed of the robot, which is calculated by measuring the acceleration force toward the motors. If the number of generated events in the camera increases, that might be due to the increase in the complexity of the environment or the speed of the robot, the CPU load increases that affect the throughput. The current throughput of the application must always be higher than a pre-specified throughput for the application. In the case that throughput goes below the threshold due to a heavy workload, the Performance Management Unit is capable to increase the speed of the computation by increasing the voltage/frequency of the computational cores, or, decreasing the speed of the robot by decreasing the motor voltage.

B. The control algorithm

The Performance Management Unit works as a closed-loop controller to adjust the performance based on the feedback it gets from the power sensors on the motor and the CPU. Due to the peculiar characteristics of the problem space discussed in Section II-B, we have designed the controller algorithm by means of the hill-climbing algorithm, a simple and fast heuristic to search in a solution space based on local moves. Algorithm 1 shows the controller algorithm. The inputs of the algorithm are the entropy of the environmental information et, current power consumption of the CPU p_{cpu}, current power consumption of the motor p_{motor}, throughput error λ_e and the application info ai. The outputs of the algorithm are the CPU voltage/frequency, vf, and the motor voltage mv.

The algorithm is executed whenever working scenario changes, due to entering of new applications or leaving of currently executing ones. At the beginning, the CPU voltage/frequency and motor voltage are determined based on an initialization process (Lines 1-3). In this phase, the CPU voltage/frequency and the motor voltage are adjusted based on some *safe values* that approximately satisfies the required throughput; a conservative choice for our experiment is the maximum voltage/frequency level. The current entropy, $et_{current}$, is initialized to zero to be updated later in the algorithm. This initialization of entropy to zero is due to the fact that in zero entropy no event will be generated by the event-based camera and the best possible adjustment of actuations can be calculated in the initialization phase. Therefore there is no need for any optimization. However, if the captured entropy value from the environment is above zero, the process of tuning the CPU voltage/frequency and the motor voltage starts and continues until the application information is valid (Lines 4-19). During this period,

Algorithm 1 Performance Management algorithm

Input: Entropy: et; CPU power: p_{cpu}; Motor power: p_{motor}; Throughput error: λ_e; Application Info: ai;
Output: CPU voltage/frequency: vf; Motor voltage: mv;
Constant: Entropy threshold: et_{th}; Energy threshold: e_{th};

```
1:  (vf, mv) ← Initialization(ai)
2:  state_current ← (vf, mv)
3:  et_current ← 0
4:  while ai is valid do
5:      if |et_current − et| > et_th then
6:          et ← et_current
7:          repeat
8:              neighbors^list_state ← Get_neighbors (state_current)
9:              for all (vf, mv) in neighbors^list_state do
10:                 Apply (vf, mv)
11:                 Delay()
12:                 E_new ← Compute_energy(p_cpu, p_motor)
13:                 if E_new − E < e_th and λ_e = 0 then
14:                     state_new ← state
15:                     E ← E_new
16:         until state_current = (vf, mv)
17:         state_current ← (vf, mv)
18:     (vf, mv) ← state_current
19:     Apply (vf,mv)
```

whenever the input entropy, et, differs from the current entropy, $et_{current}$ (i.e., IF statement at Lines 5-18), the algorithm tries to find the optimal state of the system, i.e., CPU voltage/frequency and motor voltage, to minimize the overall power consumption. This ensures optimized actuations whenever the environmental complexity changes. If so, the current entropy, $et_{current}$, gets updated (Line 6), and the hill-climbing process in the 2D domain of CPU voltage/frequency and the motor voltage starts to find the optimal state (Lines 7-16). First, the neighbor states of the current state are determined (Line 8). After that, the algorithm systematically *tests* all the possible neighbors of the current (CPU voltage/frequency, motor voltage) pair to find the energy-efficient values. For each test, the new state will be applied to the robot, both CPU voltage/frequency and motor voltage, by *Apply* routine (Line 10).

Then, the algorithm waits for a pre-specified *delay* to be able to measure stable values for the instantaneous power of the CPU and motor and the throughput of the application. In $compute\_energy$ routine, the new overall energy consumption is computed via instantaneous power feed-back from the Computing Unit and Motor, i.e., p_{cpu} and p_{motor}. Subsequently, the new computed overall energy consumption, E_{new}, is compared with the old measured energy value, E, to check whether the new test pair results in a less overall energy consumption or not. During this phase, throughput also should be checked since the climbing process might have a negative effect on throughput (Lines 13-15). Finally, the state that acquires the minimum obtained overall power will be considered as the new state. This process continues until the newly obtained state, $state_{new}$, is the same as the current state, $state_{current}$, which indicates that the hill-climbing process cannot improve the power consumption any more (Line 16).

It is worth noting that, even if simple, the proposed approach is effective, as also demonstrated later by the experimental results, due to the peculiar characteristics of the problem space discussed in Section II-B. Moreover, no scalability issues may raise as the problem space cannot grow sensibly for different system architectures. In fact, the number of motor speed steps and voltage/frequency levels in a general scenario is in the order of the considered working scenario. Nonetheless, we claim that the test of all the local moves, i.e., Line 9-15, may be replaced with a smarter selection of the neighbors of the current configuration based on the slope trend shown in Figure 3. This improvement is left as a future work.

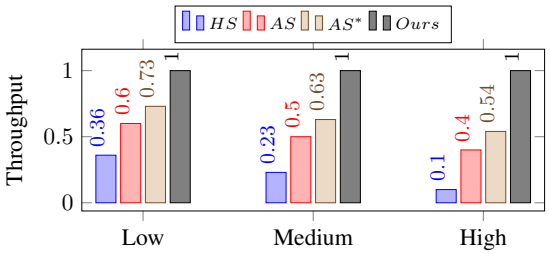


Fig. 6: Different obtained throughput for different control techniques in three different environmental complexities: low-, medium-, and high-complexity.

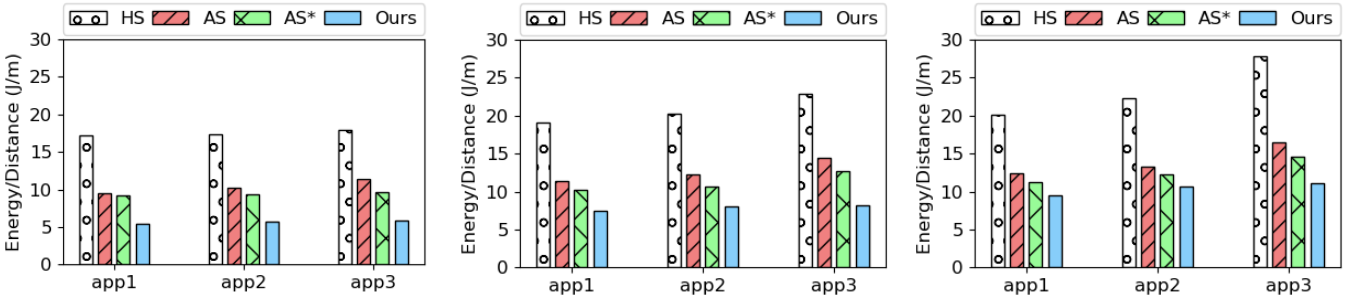
IV. Experimental Results

The proposed controller has been implemented in C++ that is executed as a middle-ware in Linux OS user space at run-time. We run the proposed control algorithm for three different environments, i.e., 1) low, 2) medium-complexity, and 3) high-complexity.

Effectiveness: To demonstrate the effectiveness of the proposed approach we compared it against three different baseline schemes in the experiments: 1) while the controller uses the highest possible speed for the robot and highest possible voltage/frequency, i.e., HS, 2) while the controller uses a medium speed and highest possible voltage/frequency, i.e., AS, and 3) while the controller uses the medium speed with a medium voltage frequency, i.e., AS^*. Three applications are used as vision-based tasks, i.e., 1) image reconstruction that is shown by $app1$, 2) corner detection, $app2$, and 3) corner detection without filtering, $app3$, please see Section II-A. The overall number of generated events for each environment is the same for all of these control techniques, including the proposed method. The reason is that the environment is the same for all techniques[4]. Figure 6 shows the average obtained throughput for different techniques. It can be seen that the proposed method obtains the highest average throughput in comparison with other methods. Here only the proposed controller can keep the throughput satisfiable for all the information environments. This is because of the fact that the motor speed affects the workload and should be tuned based on the workload of the system.

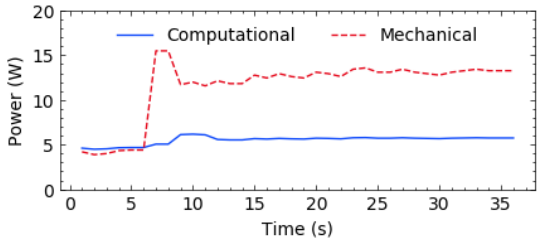
Efficiency. To show the efficiency of our technique, we used the *energy per meter* metric that shows the amount of energy

[4]In macroscopic level, the amount of perceptual information for a mobile robot in a specific path is constant

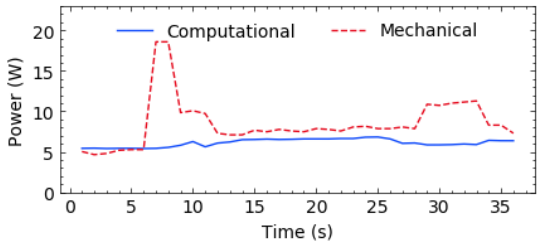


(a) Low-complexity environment (b) Medium-complexity environment (c) High-complexity environment

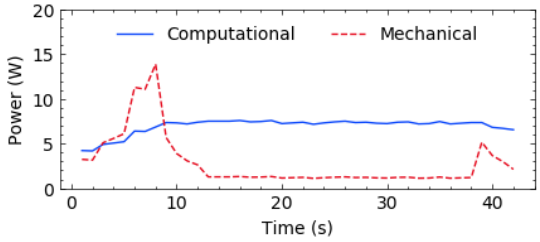
Fig. 7: Evaluation of the average energy per distance in three different environmental complexity.



(a) Low-complexity environment



(b) Medium-complexity environment



(c) High-complexity environment

Fig. 8: Evaluation of the power consumption during the time for the three different scenarios.

consumption per distance unit for a mobile robot running a specific application. Figure 7 shows the overall energy consumption per meter (J/m) of the different techniques for the three different applications and in three different environments. As can be seen, the proposed approach saves the energy of traveling per distance unit significantly compared with the other methods. Our method is able to save an average of 50.5%, 41%, and 31% of total energy in a low, medium, and high-complexity environment respectively.

Stability. To have a better understanding of the controller operation and to demonstrate the stability of the controller, the power and speed traces of two experiments in high, medium, and low-complexity environments, while *app*1 is executing, are shown in Figure 8. As can be seen, at the beginning, the controller starts the hill-climbing process to find the appropriate speed and adjusts the CPU's voltage/frequency. This causes fluctuations in speed during the start-up time. However, for the rest of the travel, the controller adjusts the speed, and CPU power is quite steady. The reason is that environmental information usually does not change rapidly. This gives the hill-climbing algorithm an opportunity to adjust the CPU voltage/frequency efficiently.

Execution times. The average execution time of each module is as follows: the event-based camera module: 10 us; the frame-based camera module: 2.4 ms; and the Performance Management Unit module: 50 ms that is the average time for finding the optimal solution. As can be noticed, the execution time for the algorithm is acceptable for a macroscopic size robot with a normal change of speed rate.

V. Related Works

The related works for energy-efficient robot control, or a swarm of robots, can be divided into two main categories: 1) those works that focus on energy-efficient computation for robotic applications, and 2) those works that focus on reducing the energy cost of the mechanical parts. From the computing perspective, several strategies have been used to control the instantaneous power and overall energy of the computation using different techniques acting at both architecture and application levels, such as DVFS [8], energy-efficient task mapping and scheduling [9], and software approximate computing [10]. The main idea in these works is to reduce the computing power/energy as much as possible while keeping the quality of service (in terms of the provided throughput) satisfactory. On the other hand, and from the robotics perspective, there have been several researches to optimize the energy cost and battery utilization of the mechanical parts that are ranging from conceptual studies such as bio-inspired theories based on least action principle (LAP) [25], to optimizing mechanical motion algorithms, e.g., reducing locomotion cost of different parts of the robot [7], [19] and energy-efficient path planning [1], [2].

To the best of our knowledge, all of the studies in these two areas are considering the optimization of the computing and mechanical parts independent from each other, i.e., co-optimization has not been investigated in practice. In [25], the authors have proposed a *new theory* in multi-robot

environment in which computing and mechanical parts are considered together under the umbrella of overall *entropy measurement* of the system. It is stated that the main goal of an autonomous system should be to save energy by reducing the entropy as much as possible. In [26], the author has proposed an *architectural characterization* for cyber-physical agents that consists of *internal and external environments*, and by evaluating measurements from these two environments the controller should decide the values of *internal and external actuators*. In this paper, the internal and external environments correspond to the computing platform and the robot environment respectively, and, internal and external actuators correspond to voltage/frequency scaling for the computing unit and controlling the robot speed respectively.

VI. Conclusions

In this paper, a novel control approach is proposed to intelligently tune the CPU voltage/frequency and motor voltage of a mobile robot running vision-based applications. The proposed approach uses the complexity of the environment, instantaneous power of the motor, and CPU to compute the overall energy consumption of the system and manipulate the CPU voltage/frequency and the motor voltage of the robot at run-time. A run-time hill-climbing algorithm has been proposed to find the near-optimal energy-efficient solution for the controller. Experimental results show that by utilizing our method a mobile robot equipped with a Jetson TX2 board can save an average of 50.5% of the energy in a low-complexity environment, 41% in a medium-complexity environment, and 30% in a high-complexity environment. As future work, we intend to extend our approach by considering the energy consumption of the path planning module.

Acknowledgment

This work has been financially supported by the Academy of Finland funded projects 335512 - ADAFI (Adaptive-Fidelity Digital Twins for Robust and Intelligent Control Systems) and 330493 - AURORA (Autonomous Performance Management in Digital Manufacturing), and by Nokia Jorma Ollila Grant.

References

[1] T. Kundu and I. Saha, "Energy-Aware Temporal Logic Motion Planning for Mobile Robots," in *Proc. Intl. Conf. on Robotics and Automation (ICRA)*, 2019, pp. 8599–8605.

[2] C. Cunningham, J. Amato, H. L. Jones, and W. L. Whittaker, "Accelerating energy-aware spatiotemporal path planning for the lunar poles," in *Proc. Intl. Conf. on Robotics and Automation (ICRA)*, 2017, pp. 4399–4406.

[3] Yongguo Mei, Yung-Hsiang Lu, Y. Hu, and C. Lee, "A case study of mobile robot's energy consumption and conservation techniques," in *In Proc. Intl. Conf. on Advanced Robotics*, 2005, pp. 492–497.

[4] M. Gabiccini, A. Artoni, G. Pannocchia, and J. Gillis, "A computational framework for environment-aware robotic manipulation planning," in *Robotics Research: Volume 2*, A. Bicchi and W. Burgard, Eds. Springer, 2018, pp. 363–385.

[5] V. Duchaine, S. Bouchard, and C. Gosselin, "Computationally efficient predictive robot control," *IEEE/ASME Trans. on Mechatronics*, vol. 12, pp. 570 – 578, 11 2007.

[6] D. E. Orin and W. W. Schrader, "Efficient Computation of the Jacobian for Robot Manipulators," *Intl. Journal of Robotics Research*, vol. 3, no. 4, pp. 66–75, 1984.

[7] S. Abiko and G. Hirzinger, "Computational efficient algorithms for operational space formulation of branching arms on a space robot," in *In Proc. Intl. Conf. on Intelligent Robots and Systems*, 2008, pp. 3312–3317.

[8] A. M. Rahmani, B. Donyanavard, T. Mück, K. Moazzemi, A. Jantsch, O. Mutlu, and N. D. Dutt, "SPECTR: Formal Supervisory Control and Coordination for Many-core Systems Resource Management," in *Proc. of Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018, pp. 169–183.

[9] M. Haghbayan, A. Rahmani, A. Y. Weldezion, P. Liljeberg, J. Plosila, A. Jantsch, and H. Tenhunen, "Dark silicon aware power management for manycore systems under dynamic workloads," in *In Proc. of Intl. Conference on Computer Design (ICCD)*, 2014.

[10] A. Kanduri, A. Miele, A. M. Rahmani, P. Liljeberg, C. Bolchini, and N. Dutt, "Approximation-aware coordinated power/performance management for heterogeneous multi-cores," in *Proc. of ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[11] L. Matignon, L. Jeanpierre, and A.-I. Mouaddib, "Distributed Value Functions for Multi-Robot Exploration: a Position Paper," *Proc. Intl. Conf. on Robotics and Automation (ICRA)*, pp. 1544–1550, 02 2013.

[12] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya, "Some observations on optimal frequency selection in DVFS-based energy consumption minimization," *Journal of Parallel and Distributed Computing*, vol. 71, no. 8, pp. 1154–1164, 2011.

[13] D. Gehrig, H. Rebecq, G. Gallego, and D. Scaramuzza, "EKLT: Asynchronous Photometric Feature Tracking Using Events and Frames," *Intl. Journal of Computer Vision*, pp. 1–18, 08 2019.

[14] C. Forster, M. Faessler, F. Fontana, M. Werlberger, and D. Scaramuzza, "Continuous on-board monocular-vision-based elevation mapping applied to autonomous landing of micro aerial vehicles," in *Proc. of Intl. Conf. on Robotics and Automation (ICRA)*, 2015, pp. 111–118.

[15] U. M. Nunes and Y. Demiris, "Entropy Minimisation Framework for Event-Based Vision Model Estimation," in *Proc. of European Conf. on Computer Vision (ECCV)*, 2020, pp. 161–176.

[16] S. A. S. Mohamed, M. H. Haghbayan, J. Heikkonen, H. Tenhunen, and J. Plosila, "Towards Dynamic Monocular Visual Odometry Based on an Event Camera and IMU Sensor," in *Proc. of Intl. Conf. on Intelligent Transport Systems (INTSYS)*, 2020.

[17] S. Gao and H. Jia, "Integrated configuration and optimization of electric vehicle aggregators for charging facilities in power networks with renewables," *IEEE Access*, vol. 7, pp. 84 690–84 700, 2019.

[18] N. V. der Noot, A. J. Ijspeert, and R. Ronsse, "Bio-inspired controller achieving forward speed modulation with a 3D bipedal walker," *Intl. Journal of Robotics Research*, vol. 37, no. 1, pp. 168–196, 2018.

[19] D. Zarrouk and R. S. Fearing, "Cost of locomotion of a dynamic hexapedal robot," in *Proc. of Intl. Conf. on Robotics and Automation (ICRA)*, 2013, pp. 2548–2553.

[20] G. Gallego, T. Delbruck, G. M. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based Vision: A Survey," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, pp. 1–1. Early access, 2020.

[21] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240x180 130dB 3µs Latency Global Shutter Spatiotemporal Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, 2014.

[22] C. Scheerlinck, N. Barnes, and R. Mahony, "Continuous-time Intensity Estimation Using Event Cameras," in *Asian Conf. Comput. Vis. (ACCV)*, 2018, pp. 308–324.

[23] S. A. S. Mohamed, J. N. Yasin, M. H. Haghbayan, A. Miele, J. Heikkonen, H. Tenhunen, and J. Plosila, "Asynchronous corner tracking algorithm based on lifetime of events for DAVIS cameras," in *Proc. of Intl. Symp. on Advances in Visual Computing (ISVC)*, 2020, pp. 530–541.

[24] S. A. S. Mohamed, J. N. Yasin, M. H. Haghbayan, A. Miele, J. Heikkonen, H. Tenhunen, and J. Plosila, "Dynamic Resource-aware Corner Detection for Bio-inspired Vision Sensors," in *In Proc. Intl. Conf. of Pattern Recognition*, 2020.

[25] S. Fox, "Active inference: Applicability to different types of social organization explained through reference to industrial engineering and quality management," *Entropy*, vol. 23, no. 2, 2021.

[26] J. Sifakis, "Autonomous Systems – An Architectural Characterization," in *Models, Languages, and Tools for Concurrent and Distributed Programming*, M. Boreale, F. Corradini, M. Loreti, and R. Pugliese, Eds. Springer, 2019, pp. 388–410.