Autonomous Real Time Architecture for High Performance Mobile Robots

Antonios E. Gkikakis[1], Dimitrios Kanoulas[2], and Roy Featherstone[3]

***Abstract*— Highly-dynamic robotic systems, such as hopping robots, require light, computationally and energy efficient on-board units for control. This paper presents such a computational unit together with a software architecture for achieving high-performance behaviors, such as balancing and hopping. These demanding behaviors require accurate dynamic calculations, high-bandwidth control, and fast real-time state estimation. The proposed system consists of cheap and off-the-shelf electronics that are detailed in this paper. The effectiveness of the presented approach is validated on a balancing machine called Tippy, which is able to achieve fast tracking of command signals while balancing. The experimental results of this paper demonstrate that reliable real-time software for demanding high-performance robotic applications, which require fast control loops and intensive calculations, can be achieved with light, low cost and energy efficient components, which can empower the widespread use and experimentation of high-performance robots worldwide.**

I. Introduction

Highly-dynamic robots have been the topic of research since the early 80s with the introduction of Raibert's hopping machines [1]. Some of the main reasons that robots must be fast are so that they can efficiently assist humans, and because swiftness is essential for adapting to rapid changes in their state of motion or their environment. An example of a highly dynamic behavior is legged locomotion, which requires fast and accurate system responses. Particularly, a fraction of a second is enough for the system to lose its balance and fall. Agile robots, such as Spot of Boston Dynamics [2], can reject disturbances, such as being pushed, and maintain their balance. Fast robots have the potential to cope with many undesired scenarios, and hence can be more robust and reliable. However, building such systems is not a trivial problem to solve, and has many pitfalls.

To achieve high speed, a robotic system requires accurate dynamic calculations, fast control loops, and real-time state estimation. Furthermore, agile motions may result in high-risk activity that might damage the robot or its environment. Thus, appropriate strategies and modules must be devised to ensure a reliable and safe operation. The aforementioned problems impose a tough obstacle for the computational system and the software that implements high-performance behaviors. Fast robotic arms already exist in confined industrial environments [3]; however, they come with a high cost and are restricted to controlled and certain environments. An open problem in bringing more dynamically operating robots into the real-world environments for everyday life applications is the need for cheap, reliable, and energy efficient solutions.

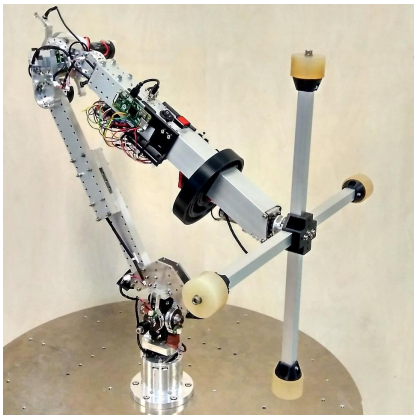

Fig. 1. Balancing machine named Tippy, that uses the presented computational unit and software architecture. Although capable of other motions, the machine is configured to behave as a planar reaction-wheel pendulum.

This paper focuses on a computational system and its software architecture to achieve high-performance motions, such as hopping and balancing. The contribution of this paper is to demonstrate a technological advancement, which shows that energy-efficient computational units for reliable and fast robots can be built at a low cost. Mobile robots have recently started to enter the market, such as Spot of Boston Dynamics [2]. It is evident that the requirements for these robots will increase to start matching the needs of new more demanding applications. To be more efficient, robots will also be required to complete tasks faster, and hence the need for developing cheap and reliable systems for controlling nimble robots will arise. Affordable and energy-efficient solutions can facilitate the widespread application and experimentation of robotics.

In this work, all the software responsible for the robot's behavior is implemented on-board, and hence the system is autonomous, compact (is small and efficiently uses its limited resources), robust, reliable (has appropriate fault detection modules), and has hard real-time implementations for dynamics, control, and state-estimation, which are essential for achieving demanding athletic behaviors.

The presented system is relatively cheap and uses a Raspberry Pi Zero W (RPI0) and a Texas Instruments (TI)

[1]Antonios E. Gkikakis is with Dept. of Advanced Robotics, Istituto Italiano di Technologia, Genoa, Italy and Dibris, University of Genoa, Italy antonios.gkikakis@iit.it

[2]Dimitrios Kanoulas is with Department of Computer Science, University College London, Gower Street, WC1E 6BT, London, UK d.kanoulas@ucl.ac.uk

[3]Roy Featherstone is with Dept. of Advanced Robotics, Istituto Italiano di Technologia, Genoa, Italy roy.featherstone@iit.it

development kit equipped with a real-time microprocessor. At the time this paper is written, the overall cost of the presented unit is less than 50 euros, and the framework implements a complete control system, which includes wireless communication between a user and the robot, high-bandwidth logging, high frequency sensor-sampling, state estimation, control loops, on-board real-time dynamics computations, and fault detection. The effectiveness of the system is demonstrated on a custom-built balancing machine named Tippy [4] (see Fig. 1), in the task of balancing and command tracking, an ability that inherently requires fast responses.

The paper is organized as follows. Firstly, the computational unit is presented (Sec. III), followed by the control hierarchy that is implemented in the system (Sec. IV). Next, the software architecture is detailed (Sec. V), and results of the system when applied on a real high-performance monopedal robot are presented (Sec. VI). Finally, future directions are discussed (Sec. VII) along with the conclusions (Sec. VIII).

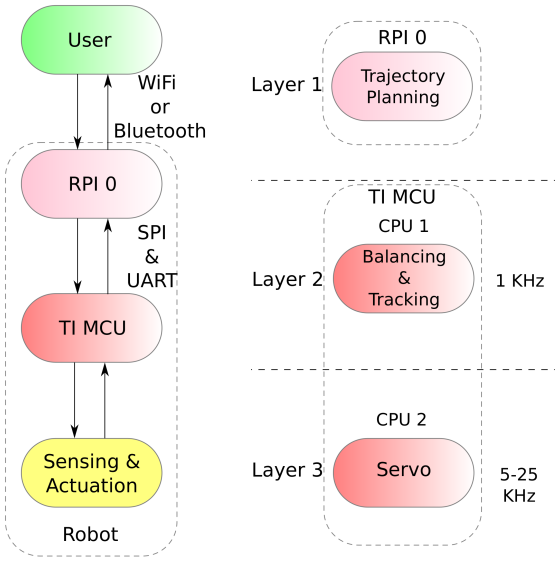

Fig. 2. System overview (left); control hierarchy (right).

II. Background

Balancing and dynamic walking are examples of high performance behaviors. Particularly, to perform hops, a robot might be required to reach a state of imbalance, where it has only a few fractions of a second to build the required momentum to propel itself upwards; because otherwise it will fall. The time when the robot can change its state of momentum is limited to when it is in contact with the ground (also called the stance phase). In the simulations of Gkikakis and Featherstone [5], where realistic simulations of various hopping behaviors of a high-performance monopod robot called Skippy are presented, the stance phase is less than 0.25 s. Similarly, in the work of Yim et al. [6], the stance phase of their miniature hopping robot Salto does not last longer than 0.3 s. During this short time frame, the robot must skillfully control the magnitude and the direction of the ground force to achieve the desired lift-off conditions. Both of these robots are relatively light; Salto weights a bit more than 0.1 kg, and Skippy is currently being designed to have a mass of 3 kg. The former achieves its athletic behaviors with an on-board real time microcontroller, that runs a 1 KHz servo for motor control and a balance controller at 500 Hz, but depends on an external computer for trajectory command generation, and the latter will be equipped with the system presented in this paper.

Cheap and lightweight electronics such as Raspberry Pi (RPI) have been recently popularized in control of mobile-robots [7]–[12]; however, RPI lacks the variety of hardware accelerators that are present in real-time microcontrollers, which can allow a much higher performance to be achieved since many time-costly tasks (communication, reading/writing from memory) can be handled by hardware, and hence the processor can spend almost all of its resources on computations.

In the work of Grimminger et. al [13] a 2.2 kg jumping quadruped robot also uses a TI evaluation board to perform high-level control at 1 KHz and at 10 KHz for motor control; however, the high level controller runs on a separate computational unit outside of the robot. Katz et. al [14] designed the mini-cheetah, a relatively small (has a mass 9 kg) and agile quadruped capable of athletic behaviors such as a back-flip. The robot has torque control loops running up to 4.5 KHz, control and state estimation at 1 KHz, and uses a more expensive computer unit than the one presented in this paper.

Larger robots are also capable of high-performance feats. HyQ has demonstrated high-performance balancing in the work of Gonzalez et. al [15], where the quadruped balances on a line using only two feet. The robot uses a two-layer control scheme running a balancing controller [16] at 250 Hz and a low-level controller at 1 KHz; which are frequencies lower than the aforementioned robots. The robot is also equipped with significantly more expensive computational units than the one presented in this paper.

III. System Overview

In this section, we describe the introduced system, whose overview appears in Fig. 2 (left). The user connects to the system via WiFi or Bluetooth and communicates with the robot through a custom-made user interface (UI) implemented in C language. The computational unit consists of a Raspberry Pi Zero W [17] (RPI0) and the LAUNCHXL-F28377S LaunchPad development kit, that is equipped with a dual core TI TMS320F28377S [18] real-time microprocessor (TI MCU). RPI0 was selected due to its compact size, low price, computational power, and its large community; the TI MCU was chosen based on the number of desired hardware accelerators, its affordable price, its power efficiency, and real-time control capabilities. The former is responsible for the following tasks:

1) providing an interface for the user,
2) storing log data,
3) trajectory generation and transmission, and

4) transmitting metadata to the microcontroller, such as controller gain values;

the latter is responsible for all the real-time computations, sensor sampling and motor control needs of the robot. It is essential for these computations to be performed in hard real-time to guarantee that the system will meet time requirements. RPI0 has a wireless interface and runs a standard Linux-based operating system named Raspberry Pi OS [17], through which the user connects to the robot via WiFi or Bluetooth and runs a script that initiates the communication with the real-time microcontroller.

The RPI0 is equipped with an SD card to allow the storage of several hours of logs. Furthermore, its 1 GHz clock allows it to perform high-level computations, such as trajectory planning, that we plan to implement in a future version of the system. (Currently only logging is performed and the CPU is IDLE for most of the time.) The results presented in [4] use precomputed trajectories.

The TI MCU has single-precision floating-point arithmetic, and has two independent cores that both run in parallel at 200 MHz. Details about the real-time implementation are presented over the next two sections.

IV. Control Hierarchy

Control is divided into three layers (see Fig. 2). In the first layer a trajectory is defined, that can be given in real-time or be predetermined. The trajectory consists of joint positions, velocities, and accelerations. Having high-order derivatives of the reference signal can improve tracking ability of the balance controller [16], but add additional complexity and increase the communication overhead. The signal is transmitted to the TI MCU via high-speed SPI communication.

The second and third layers are implemented in the real-time microcontroller. The trajectory is passed to the second layer, which has a C implementation of the planar balancing controller described in Featherstone [16]. The output of the balancing controller is then fed to the third layer, which implements a servo for controlling the motors. The balancing controller runs up to 1 KHz and the servo can reach frequencies up to 25 KHz. In the second layer real-time dynamics are calculated, and the control output is desired accelerations and joint torques, which are fed to the third layer that implements a PID controller for tracking these signals.

The second CPU (CPU 2) of the TI MCU is designed for executing tasks with low latency, and hence is a suitable choice for implementing time-critical tasks, such as low-level controllers and state estimation. Heavy-duty computations, such as dynamics and calculating the balancing controller output, are performed in CPU 1.

V. Software Architecture

The overview of the software architecture is presented in Fig. 3, together with the frequency value (or a frequency value range) of each of the presented events inscribed in the blocks. Depending on the various sensors used, the microcontroller can achieve different sensor sampling frequencies.

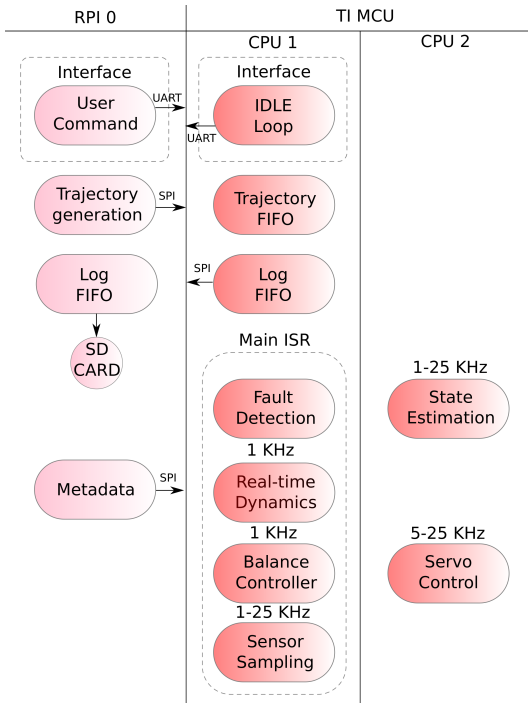

Fig. 3. Architecture of the system. Blocks represent tasks, and each column contains the tasks which the corresponding processor handles. For synchronous tasks their frequency appears on top of the block. The system consists of three processors: (1) the RPI0, (2) CPU 1 of the real-time microcontroller, where communication and heavy duty computations are performed, and (3) the second CPU of the microcontroller that runs independently and in parallel with CPU 1.

The RPI0 uses asynchronous communication (UART) to transmit user commands to the TI MCU, and the latter uses the same protocol for informing the RPI0 about the status of an ongoing experiment or any fault that appears (see Sec. V-A). Larger volumes of data are transmitted via the faster SPI synchronous communication; these data are logs, trajectories or metadata (see Sec. VI).

The software in the real-time microcontroller is interrupt-driven. A main interrupt service routine (ISR) is called at a rate between 10 and 25 KHz (depending on the requirements) in CPU 1 to perform the tasks mentioned in Figure 3 at the desired frequency. For example, if the servo control runs at 25 KHz, so does the ISR, and the balance controller computation happens every twenty-fifth call of the ISR. The ISR is also responsible for invoking the tasks of CPU 2 by simply changing a register value. Then, the tasks of CPU 2 happen in parallel and independently of CPU 1, and when the time comes CPU 1 checks if they terminated and retrieves the computed values via a shared memory pool.

A. Fault Detection

A crucial part of any real-time system is fault detection. Fast motions are difficult to stop, and for that reason the system must be designed not only to recover from but also to prevent undesired scenarios. The system must always meet time constraints, be certain of the proper functionality of all of its components as well as the available resources. Fast robots need keen perception of their environments, and that

means that all of their sensors must work as expected. These measures depend on the application, and for the system presented in this paper are:

1) watchdog timer to check if the system operates (it hasn't hung);
2) status check of sensors (e.g., header values and CRC);
3) communications must be completed within a given time frame and periodicity (i.e., sampling sensors);
4) motor saturation limits, for preventing the system to go out of control;
5) joint limits, for preventing the system from reaching undesired states;
6) power supply check (i.e., notify the user if the state of charge of the battery is below a certain threshold);
7) user-connection check, because all computations happen on-board the system does not depend on the user to operate, hence when the robot loses the connection with the user it returns to a home-balancing configuration until connection is re-established.

These measures are a necessity for increasing the reliability and the safety of the system. In addition, they help prevent catastrophic scenarios, where the robot might harm itself or its environment, and must be performed at frequent intervals. We consider measures 1–3 to be hard-faults (in measures 1–2 hard real-time assumptions are violated, and in measure 3 sensing is not properly functioning), because they may lead to unpredictable scenarios, and in case they are violated the system stops its operation.

B. Real-time Dynamics and Control

Fast robots require accurate dynamics and fast controllers. Dynamic computations are usually costly, with the cost being increased depending on the robot's number of joints, existence of kinematic loops etc. The software architecture of our system is designed to optimally use its limited resources and to spend minimal time in time-costly tasks such as communication with the use of hardware accelerators, allowing the system to reach its true potential in terms of control bandwidth. Dynamic computations are performed in hard real-time on the TI microprocessor, which contains optimized C implementations for planar versions of the following algorithms:

1) Recursive Newton Euler Algorithm (RNEA) [19];
2) Composite Rigid Body Algorithm (CRBA) [19]; and
3) the balance controller proposed in Featherstone [16].

The aforementioned algorithms are executed at a rate of 1 KHz on the TI MCU CPU 1 (see right Figure 2). The output of the balancing controller is passed to the servo which runs at a faster rate than the former, and is responsible for making the motor track the command signals.

Algorithmic Optimization—the efficiency of these algorithms has been furtherly improved by exploiting Branch-Induced Sparsity. This phenomenon can result from branches in the kinematic tree of the robot, which lead to patterns of zeros appearing in the joint-space inertia matrix. Sparsity is exploited by using modified algorithms that iterate over only the nonzero elements. Exploiting sparsity can in some cases lead in substantial computational cost reductions in typical algorithms such as the CRBA from $\mathcal{O}(n^2)$ to $\mathcal{O}(nd)$ [20], which can allow efficient dynamic calculations even in complicated robots.

Memory Optimization—to maximize the usage of the limited available memory of the microcontroller a signal compression strategy is implemented. This strategy uses a simple cubic interpolation scheme to compress/decompress a command signal. In this way lengthy control signals can be stored in a compact way in the microcontroller's limited memory. In the experiments presented in [4] a command signal of 10 seconds (for 1 KHz 10,000 points are required) has been compressed to only 162 points (61.2 compression ratio), which are then used to reconstruct the original signal in real-time.

C. Sensing and State Estimation

For the robot presented in [4], state estimation consists of position and velocity estimation from joint encoders. For joint angle positions we use two iC-Haus' iC-MU150 18-bit [21] magnetic absolute encoders, and for the motor position we use Maxon's ENX 16 12-bit incremental encoder [22]. The absolute encoders are used by the balancing controller to measure the position of the crossbar, and the vertical direction (required for balancing) from a spherical joint at the base of the robot. The incremental encoder is used to measure the motor position and is used by the servo.

For velocity estimation we use a second-order Butterworth filter with a cut-off frequency of 500 rad/s for the absolute encoders and 3000 rad/s for the incremental encoder, which all run at CPU 2 (see Fig. 3). To sample these sensors, hardware accelerators are used, which are called the Enhanced Quadrature Encoder Pulse (eQEP) modules [18], where reading of the incremental encoders is handled completely by hardware that stores the values in registers, and hence does not add any communication overhead.

D. Communication and Logging

Logging of data is performed at 1 KHz. The MCU has limited memory (164 KB of RAM) so the logs must be transmitted quickly and reliably in frequent time intervals. For this reason a software FIFO queue is implemented for storing the logs.

To utilize all of the computational power of the microcontroller, and not waste time on communication and copying of data, the software uses Direct-Memory-Access (DMA), and hardware FIFO queues for transmitting/receiving data.

DMA copies data between memory locations while following specific rules that dictate when it is allowed to perform the operation (e.g., copy the data only when the queue has available space). We use DMA to copy data into/from the FIFO of the SPI channel. This operation is completely handled by hardware, and the software only needs to define the memory locations to copy from/to, and when the process starts. In this way, the software can focus all of its resources on computations. We implement a low-level

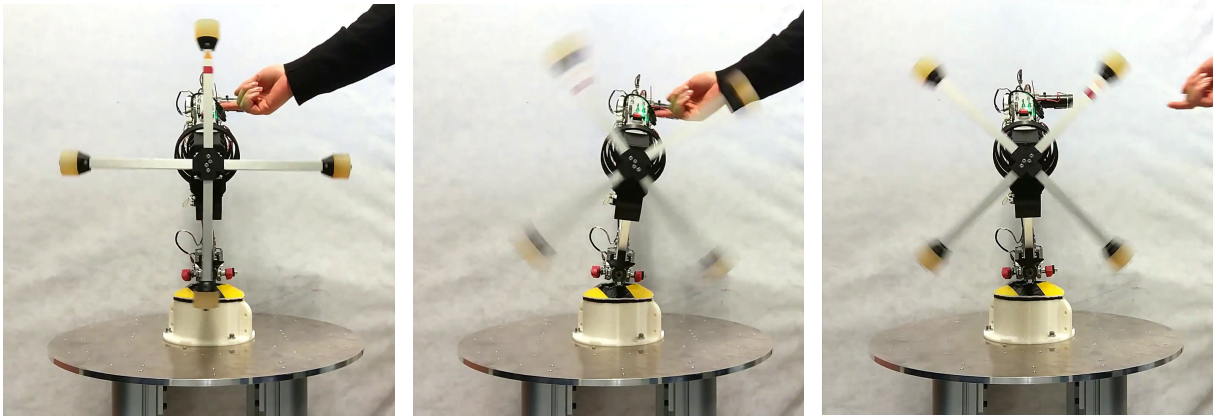

Fig. 4. Example of high-performance behavior demonstrated on Tippy, a balancing machine. The robot is connected via an unactuated spherical joint to the base; the robot balances by rotating its crossbar. Left: moments before external disturbance; middle: during a disturbance the robot needs to react in a very short amount time to prevent falling; right: the robot rejects the disturbance and continues operation.

communication protocol using SPI for transmitting logs from the microcontroller to the RPI0 at a rate of 1 KHz, which consist of 96 32-bit float values of interest.

VI. Results

The presented system is used on a 5-DoF balancing machine called Tippy, and experimental results are presented in Driessen et. al [4]. Tippy uses the balancing algorithm of Featherstone [16] to balance and track a command signal in 2 D. The robot is a full 3 D balancing machine, but in [4] it is configured as a 2 DoF planar reaction wheel pendulum (RWP). In this configuration a brushed Maxon DCX 22S 24V motor [22] is actuated for balancing and following a command signal, by rotating the crossbar (see Fig. 1). The robot is powered by a LiPo battery with a nominal voltage of 30 V.

In terms of power consumption the computational unit spends less than 2 W, and uses only $\sim 10\%$ of its computational power in CPU 1, and even less than that in CPU 2, meaning that most of the time it is inactive. This shows that much heavier computations can be performed, such as 3 D dynamics, or that the same system could be used in more complicated robots. A simple and cheap microcontroller such as the one used in this system [18] has enough computational power and modules to control at least 3 DC motors.

In the task of balancing only a few hundred milliseconds are enough for the robot to lose its balance and fall [16]. In the video [23] of the experimental results presented in [4], it can be seen that the proposed system can follow fast precalculated command signals, such as ramps, sinusoids and step position commands with its crossbar, while maintaining its balance, and can also respond to external disturbances without falling (see Fig. 4). To achieve such a behavior the balancing controller runs at 1 KHz and the servo at 5 KHz. Experimenting with servo control cycles below 5 KHz shows a reduction in the tracking and balancing response of our system, indicating that high control bandwidth is important for obtaining high-performance in this given task. For servo control, we have found that a simple velocity proportional controller works well in practice. The reference velocity to the servo is obtained by integrating the joint acceleration output of the balance controller. Our system has the potential to sample the absolute encoders at 15 KHz, and the incremental encoders at 25 KHz, but we were limited by noise in the signals.

An additional utility that was implemented and facilitated the experimentation process, is soft real-time gain tuning (block 'metadata' in Fig. 3). By using the UI, the user is able to modify values such as the controller gains and dynamic parameters of the model; a utility that is present in many robotic computational systems and could also be implemented in the low-cost computational unit that is presented in this paper.

Finally, the presented system has been tested to operate completely autonomously and balance for more than 40 minutes proving its reliability. During this period, a user was occasionally disturbing the system (see Fig. 4) and continuous logging of all the data was performed.

VII. Future Work

The LAUNCHXL-F28377S LaunchPad development kit is an evaluation and development tool, that does not provide access to all of the pins of the microcontroller. Fig. 5 shows a computational unit stack, that consists of an RPI0 and a custom made PCB for the TI TMS320F28377S microprocessor. This unit will be the brain of Skippy, which is presented in [5], that will allow the autonomous operation of the high-performance legged robot. Its overall size is $4.4 \times 6.5 \times 2$ cm, it weighs only 33 g, and is responsible for all the computational needs of the robot.

VIII. Conclusion

This paper presents an off-the-shelf computational unit and its software architecture that is aimed for the control of high-performance robots. The aim of this paper is to demonstrate that high performance can be achieved with low cost, and energy efficient electronic computational units and with existing off-the-shelf microcontrollers.

The computational unit consists of a Raspberry Pi Zero W and a Texas Instrument real-time microcontroller. Despite the low price of the unit (costs less than 50 euros) it is able to

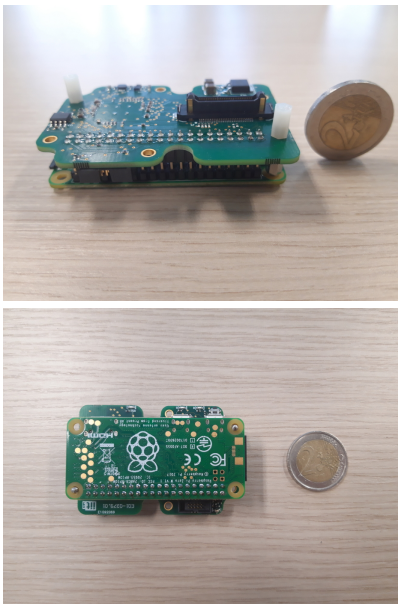

Fig. 5. Computational unit stack next to a two-euro coin for size comparison. The unit consists of a Raspberry Pi Zero W and a custom-made PCB for the TI TMS320F28377S microprocessor.

achieve reliable and high bandwidth control. Specifically, the system has been tested to reach servo cycles up to 25 KHz; perform real-time planar dynamics computations and achieve control loops at 1 KHz, all on the real-time microprocessor. Furthermore, sensor sampling, state estimation, fault detection and high-bandwidth logging is implemented. The unit is untethered and can be accessed via WiFi or Bluetooth making it a completely autonomous system where all important computations happen in real-time and on board. Additional utilities to assist experimentation were implemented such as real-time gain tuning of parameters during operation.

The unit was used to implement a high-performance balancing and tracking behavior on a 2 D simplified version of a 5 DoF under-actuated balancing robot.

The proposed approach utilizes the available technology to its maximum potential with a thorough software architecture. Specifically, the proposed architecture utilizes hardware accelerators, task parallelism, a signal compression scheme, scheduling strategies, and algorithmic optimization of dynamics algorithms. The system was optimized to efficiently use its resources (memory, computational power, energy consumption) and we envision that it can be used in energy-efficient and high-performance robots with higher degrees of freedom. The selected microcontroller can easily be replaced with its newer versions, that have more functionalities and higher computational power, and in addition, multiple microcontrollers can be stacked together. Finally, we will use the unit in a successor robot named Skippy with 9 DoF and two actuators, that will also be capable of high-performance hopping, a behavior which also requires very fast responses.

The results of this work demonstrate that autonomous, reliable, efficient and fast robots can be built with cheap electronics and carefully designed software that efficiently utilizes the available resources. The presented system architecture can be used as a seed for the design of more complicated systems, that are energy-efficient and have low cost electronics, with the aim to empower the widespread use of robotics in developing countries and contribute to unleashing global robotics-related opportunities.

Acknowledgment

Special thanks to Phil E. Hudson, Bajwa Roodra Pratap Singh, and Juan David Gamba Camacho for their contributions in the system's development. This work was partially supported by the Italian Workers' Compensation Authority (INAIL).

References

[1] M. H. Raibert, *Legged Robots that Balance*. MIT press, 1986.
[2] Boston Dynamics, "Spot," 2021. https://www.bostondynamics.com/spot, accessed Jun. 2021.
[3] H. A. Almurib, H. F. Al-Qrimli, and N. Kumar, "A Review of Application Industrial Robotic Design," in *9th International Conference on ICT and Knowledge Engineering*, pp. 105–112, 2012.
[4] J. J. M. Driessen, A. E. Gkikakis, R. Featherstone, and B. R. P. Singh, "Experimental Demonstration of High-Performance Robotic Balancing," in *IEEE ICRA*, pp. 9459–9465, 2019.
[5] A. E. Gkikakis and R. Featherstone, "Realistic Mechanism and Behaviour Co-design of a One Legged Hopping Robot," in *2021 IEE Int. Conf. on Computer, Control and Robotics (ICCCR)*, pp. 42–49, 2021.
[6] J. K. Yim, B. R. P. Singh, E. K. Wang, R. Featherstone, and R. S. Fearing, "Precision robotic leaping and landing using stance-phase balance," *IEEE RA-L*, vol. 5, no. 2, pp. 3422–3429, 2020.
[7] S. C. Gomez, M. Vona, and D. Kanoulas, "A Three-Toe Biped Foot with Hall-Effect Sensing," in *IEEE/RSJ IROS*, pp. 360–365, 2015.
[8] A. U. Bokade and V. R. Ratnaparkhe, "Video Surveillance Robot Control using Smartphone and Raspberry Pi," in *Int. Conf. on Communication and Signal Processing (ICCSP)*, pp. 2094–2097, 2016.
[9] M. Güleçi and M. Orhun, "Android based WI-FI controlled robot using Raspberry Pi," in *UBMK*, pp. 978–982, 2017.
[10] G. O. E. Abdalla and T. Veeramanikandasamy, "Implementation of spy robot for a surveillance system using internet protocol of raspberry pi," in *2nd IEEE RTEICT*, pp. 86–89, 2017.
[11] S. Bisi, L. De Luca, B. Shrestha, Z. Yang, and V. Gandhi, "Development of an emg-controlled mobile robot," *Robotics*, vol. 7, no. 3, p. 36, 2018.
[12] S.-E. Oltean, "Mobile robot platform with arduino uno and raspberry pi for autonomous navigation," *Procedia Manufacturing*, vol. 32, pp. 572–577, 2019.
[13] F. Grimminger, A. Meduri, *et al.*, "An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3650–3657, 2020.
[14] B. Katz, J. Di Carlo, and S. Kim, "Mini Cheetah: A Platform for Pushing the Limits of Dynamic Quadruped Control," in *2019 ICRA*, pp. 6295–6301, IEEE, 2019.
[15] C. Gonzalez, V. Barasuol, M. Frigerio, R. Featherstone, D. G. Caldwell, and C. Semini, "Line walking and balancing for legged robots with point feet," *arXiv preprint arXiv:2007.01087*, 2020.
[16] R. Featherstone, "A simple model of balancing in the plane and a simple preview balance controller," *The International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1489–1507, 2017.
[17] Raspberry, "Raspberry Pi Zero W," 2021. https://www.raspberrypi.org, accessed Jun. 2021.
[18] Texas Instruments, "TMS320F28377S," 2021. https://www.ti.com/, accessed Jun. 2021.
[19] R. Featherstone, *Rigid body dynamics algorithms*. New York: Springer, 2008.
[20] R. Featherstone, "Efficient factorization of the joint-space inertia matrix for branched kinematic trees," *The International Journal of Robotics Research*, vol. 24, no. 6, pp. 487–500, 2005.
[21] iC-Haus, "iC-MU150," 2021. https://www.ichaus.de/ic-mu150, accessed Jun. 2021.
[22] Maxon Group, "DCX32L 24V," 2021. https://www.maxongroup.com/, accessed Jun. 2021.
[23] Roy Featherstone, "Tippy RWP Balancing video," 2021. http://royfeatherstone.org/skippy/index.html, accessed Jun. 2021.