ASEBA: A Modular Architecture for Event-Based Control of Complex Robots

Stéphane Magnenat, *Student Member, IEEE*, Philippe Rétornaz, Michael Bonani, Valentin Longchamp, Francesco Mondada, *Member, IEEE*

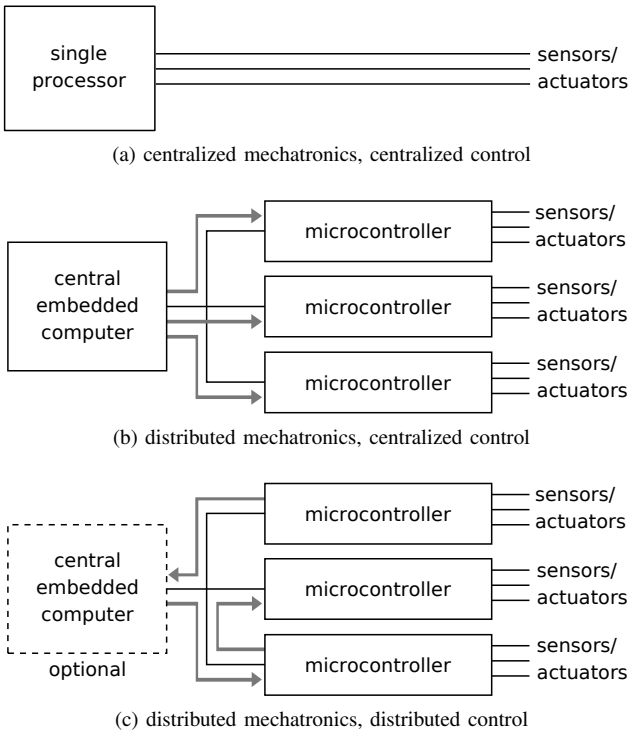
***Abstract*—We propose ASEBA, a modular architecture for event-based control of complex robots. ASEBA runs scripts inside virtual machines on self-contained sensor and actuator nodes. This distributes processing with no loss of versatility and provides several benefits. The closeness to the hardware allows fast reactivity to environmental stimuli. The exploitation of peripheral processing power to filter raw data offloads any central computer and thus allows the integration of a large number of peripherals. Thanks to scriptable and plug-and-play modules, ASEBA provides instant compilation and real-time monitoring and debugging of the behavior of the robots. Our results show that ASEBA improves the performance of the behavior with respect to other architectures. For instance, doing obstacle avoidance on the marXbot robot consumes two orders of magnitude less bandwidth than using a polling-based architecture. Moreover, latency is reduced by a factor of two to three. Our results also show how ASEBA enables advanced behavior in demanding environments using a complex robot, such as the handbot robot climbing a shelf to retrieve a book.**

***Index Terms*—Intelligent actuators, Intelligent sensors, Microcontrollers, Mobile robots**

I. INTRODUCTION

The complexity of mobile robots is continuously increasing. The rise in sensor quality and quantity is necessary to provide rich and diversified inputs to the increasingly sophisticated perception algorithms that research explores [1]–[3]. These algorithms in turn demand large computational power, which requires highly capable hardware [4]. In the field of miniature mobile robots, where space and energy are limited, one cannot integrate laptop-level processors or energy-hungry field programmable gate arrays (FPGA). One must thus exploit hardware architecture where powerful but energy-efficient processors from the smartphone industry provide a central embedded computer, while microcontroller-based peripheral nodes manage actuators and sensors in real time [5], [6]. A good example of such a robot is the s-bot [7].

However, this hardware architecture poses a scalability problem. While the real-time management of sensors and actuators is distributed, the control architecture is still centralized (Fig. 1). The robot controller runs in the central computer and reads sensors, processes the data, and sets actuators at regular intervals. In particular, the central computer must manage all read and write operations, and these must transit through a communication bus. The robots thus suffer from bus overloading and excessive latency to basic external stimuli, which limits the robots' speed of operation.



(a) centralized mechatronics, centralized control

(b) distributed mechatronics, centralized control

(c) distributed mechatronics, distributed control

Fig. 1: Different hardware and control architecture paradigms. ASEBA falls in category (c).

To solve these problems, we must distribute the controller as well, at least partially. In particular, we must filter information in the sensors themselves, and dispatch it to the rest of the robot only if and when the information is relevant to the application. This requires a shift in the behavior control paradigm: we must do event-based communication at the microcontroller level instead of polling hardware devices from the central computer. This shift in paradigm implies the definition of a set of new concepts within a new architecture. We have developed such an *event-based distributed architecture* called ASEBA.

In this article, we present the deployment of ASEBA in the handbot (Fig. 5) and the marXbot (Fig. 8), two complex miniature mobile robots that we have recently developed. Using

This work was supported by the Swarmanoid and the Perplexus projects, which are funded by the Future and Emerging Technologies programme (IST-FET) of the European Community, under grants IST-022888 and IST-034632. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

S. Magnenat is the corresponding author (email: stephane at magnenat dot net). All authors are affiliated with Mobots group - LSRO - École Polytechnique Fédérale de Lausanne (EPFL), Station 9 - CH-1015 Lausanne - Switzerland, Fax: +41 21 693 38 66 (email: firstname dot lastname at epfl dot ch)

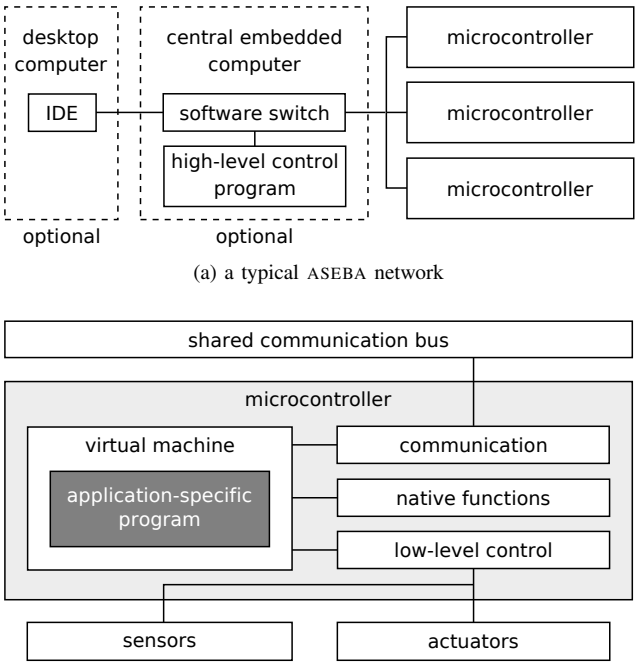
the handbot, we show that a behavior as complex as climbing a shelf can be implemented using ASEBA only (Sec. IV). Using the marXbot, we show that ASEBA improves the performance of the behavior with respect to a polling-based approach (Sec. V). We have previously presented the concept of ASEBA driving a robot in a simulator [8], the use of the ASEBA language for education [9], and the use of ASEBA to control a collection of single-microcontroller robots [10]. However, this paper is the first report of quantitative validations of ASEBA on multi-processors robots. ASEBA is open source (GPL v.3), and the community can use it and modify its source code free of charge. More information as well as the latest version are available at http://mobots.epfl.ch/aseba.html.

II. Related Work

The idea of distributing data processing to the sensors themselves was first explored by [11] almost 20 years ago. However, this work approached the question from a theoretical perspective and did not propose any concrete implementation. To do so, one must consider a complete hardware architecture, including a specific communication bus. In mobile robots, a good candidate is the CAN bus, as shown by several works that take advantage of its multi-master capabilities to let the sensors send data at some pre-defined [12] or adaptive [13] rate. ASEBA improves on these by providing an event-based architecture where the event emission policy is controlled by virtual machines (VM) inside microcontrollers. Previous work has shown that a VM can be lightweight enough to run even on tiny robots [14].

The event-based approach to multi-process communication has been extensively studied in general-purpose middleware literature [15]. Early theoretical works have shown the importance of strongly typing events [16]. More recent work has focused on using this type information to route events efficiently [17]. In the context of robotic applications, researchers have developed much middleware, and a fair number are capable of event-based communication. They all exhibit the same basic structure: a software architecture where distributed components interact through a communication layer. One of the main differences lies at the level of the communication layer: Orin [18] uses HTTP; Miro [19] and RtMiddleware [20] use CORBA; Orca uses Ice [21] while Orocos [22] and DCA [23] provide their own layers. Some provide additional features, such as Orocos, which provides a library to do Bayesian filtering, kinematics, and dynamics computation. Orccad [24] is a noteworthy approach that provides an integrated tool to build a behavior and prove its real-time constraints. Despite the diversity, these examples of middleware are all component-based architectures that run on one or more central computers, not on microcontrollers. These architectures thus suffer from the same bandwidth and latency problems as any polling-based system.

The need for a deeply embedded behavior control architecture has been recognized by researchers working on complex robots, such as robots with many degrees of freedom like modular self-reconfigurable robots [25]. In particular, researchers acknowledge the interest in defining an emission policy per microcontroller but stress its difficulty: "On the other hand, we could have all modules acting both as masters and slaves, letting the roles be determined at run time. Such a design would be very robust and flexible, if it works. However, our experience shows that the code would be very complex and difficult to debug" [26, Sec. 4]. We think that the cause of these problems is the lack of proper integration of the development and debugging process in the architecture itself. Indeed, to develop a complex behavior easily, one must be able to quickly perform trial-and-error experiments and to inspect what is happening inside the different elements of the system. Most architectures neglect this aspect and require a recompilation and reflashing of all the microcontrollers for any change in the controller code. On the contrary, ASEBA emphasizes efficient development tools and provides an integrated development environment (IDE) that allows instant changes in the behavior of the microcontrollers by loading new code to the VMs. This flexibility allows us to delegate higher-level functions to the microcontrollers, not only low-level hardware management. For instance, we can implement subsumption architectures [27] directly inside the microcontrollers. In the more demanding context of three-layer architectures [28], we can run the controller layer on the microcontrollers, which frees the central computer and allows it to concentrate on the sequencer and the deliberative layers.



(a) a typical ASEBA network

(b) a microcontroller in an ASEBA network

Fig. 2: ASEBA in miniature mobile robots.

III. Aseba

A. Distributed, Event-based Architectures

ASEBA is a modular, distributed, and event-based architecture, that is, a network of computers in which communication is asynchronous. All nodes send events and react to incoming events. An event consists of an identifier and payload data. In a small mobile robot, most of the nodes are microcontrollers,

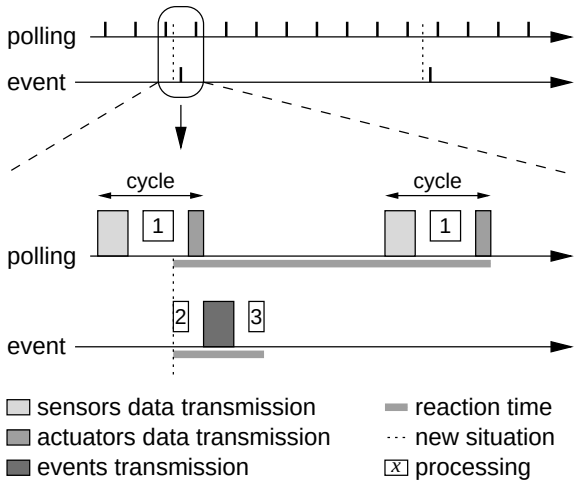


Fig. 3: A time-oriented comparison of polling versus events-based systems. (1) a central computer processing all sensors, (2) a microcontroller processing its local sensors, (3) a microcontroller processing the incoming event and setting actuators. Because processing is done locally in the microcontrollers, only useful data are transmitted, and the transfer occurs asynchronously. Thus bus load and reaction time are both reduced when using events.

and the communication layer can be any bus that is multi-master–capable. In our robots, we use the CAN bus [29] that provides this feature. In addition, the robots optionally embed a central computer, typically running Linux. In that case, a software switch extends the communication bus to local TCP/IP connections but also to any remote host. This also allows the developer to connect an IDE from a desktop computer (Fig. 2a).

Asynchronous events allow any microcontroller to transmit data when it wishes to. A basic behavior thus does not require a central computer. If one is present, it can delegate the management of reflexes to the microcontrollers and concentrate on high-level processing tasks, such as vision or cognition. A sensor would typically emit an event only when some relevant elements of information are available. For instance, a distance sensor could emit an event when an obstacle becomes closer than a certain threshold, and only when this transition occurs. The distribution of processing tasks to the microcontrollers thus reduces the load on the communication bus with respect to a polling-based architecture (Fig. 3, Sec. V-A). Moreover, when compared to polling with a fixed frequency, asynchronous events also decrease the latency, which improves the robot reaction time (Sec. V-B).

B. Events and Control

The choice of which event to send depends on the robot behavior: A robot engaged in obstacle avoidance would not need the same events as a robot following walls. Therefore, the behavior developer must be able to change the event control code easily; it must not be frozen in the firmware. In ASEBA, this flexibility is implemented by splitting the microcontroller code into two parts (Fig. 2b).

Proximity sensors microcontroller:

```
var vectorX[24] = -254, -241, ...
var vectorY[24] = -17, -82, ...
var threshold = 600
var activation

onevent sensors.updated

call math.dot(bumpers, vectorX, event.args[0], 0)
call math.dot(bumpers, vectorY, event.args[1], 0)
call math.dot(event.args[0..1], event.args[0..1],
  activation, 0)

if activation > threshold then
    emit ObstacleDetected event.args[0..1]
end
when activation <= threshold do
    emit FreeOfObstacle
end
```

Left motor microcontroller:

```
speed = 50

onevent ObstacleDetected
speed = 50 + event.args[0] - event.args[1]

onevent FreeOfObstacle
speed = 50
```

Right motor microcontroller:

```
...
speed = 50 + event.args[0] + event.args[1]
...
```

Listing 1: Example of ASEBA script implementing obstacle avoidance on a marXbot robot using potential fields. The `event.args` array corresponds to the payload data of the event.

First, sensor readings (for example, generating the timings for an infrared sensor), actuator low-level control (for example, the PID controller of a motor), and the communication layer are implemented in native code on the microcontrollers. This allows real-time, interrupt-driven handling of hardware resources.

Second, application-specific programs that control the events emission and reception policy run in a VM on the microcontrollers (Fig. 2b in the invert video). They are compiled out of a simple scripting language, which provides the necessary flexibility to allow the application developer to implement the event-based behavior.

C. Language

In ASEBA, we describe the robot behavior and the events emission and reception policy in a scripting language. The reception of an event triggers the execution of the associated part of the script, if any. The event can come from another microcontroller through the communication bus or from an internal peripheral of the microcontroller running the script. This association of code with events frees the programmer from managing the moments of execution of code.

Syntactically, the ASEBA language resembles MATLAB scripts; semantically, it is a simple imperative programming language with arrays of 16-bit signed integers as the only data type. Sensors' values and actuators' commands are seen as normal variables, which enables seamless access to the hardware. In addition to the usual *if* conditional, the ASEBA

language provides the *when* conditional, which is true when the actual evaluation of the condition is true and the last was false. This allows the execution of a specific behavior when a state changes, for instance, when an obstacle is closer than a threshold distance. To structure the code, the programmer can define subroutines that can be called from any subsequent code. To perform heavy computations, such as signal processing, microcontrollers provide native functions implemented in C or assembler. By default, a standard library provides vector operations and trigonometric functions.

Listing 1 shows an example of code that implements obstacle avoidance using potential fields. This code emits events only when it detects an obstacle and sends a preprocessed value instead of the sensors' raw values. To do so, the code uses the `math.dot` native function to compute the value to send and the `when` conditional to emit it only when the activation exceeds a threshold.

D. Integrated Development Environment

The efficiency of the development of a mobile robot behavior depends on easy inspection of what is happening inside the robot. In particular, we would like to inspect the values of the sensors, the state of the actuators, and the program execution flow. In an event-based architecture, we would also like to monitor the events that transit over the communication bus.

ASEBA provides an IDE that fulfills these requirements (Fig. 4). It communicates with microcontrollers through special events. For each microcontroller, the IDE provides a tab with the list of variables, a script editor, and debug controls. The variables' names and sizes are dynamically enumerated from the microcontroller. The list of variables also allows the real-time edition of the values of the sensors, the actuators, and the user-defined variables. The script editor provides syntax highlighting and on-typing compilation, that is, the editor compiles script into bytecode while the programmer is typing it and marks errors visually. If the script is free of compilation errors, the programmer can run it on the microcontroller in two clicks. An events log displays all the normal events and their data in real time. A distributed debugger lets the programmer set breakpoints and control the execution state of each node, for instance, to do step by step inside the script. If the bytecode on a microcontroller performs an illegal operation, such as division by zero, its execution is halted, and the faulty line is highlighted in the script editor of the corresponding tab. The microcontrollers run one separate debugger core each, but the IDE shows a unified interface to them. Thanks to these features, the IDE allows seamless development and debugging of all nodes in the network from a single place.

E. Virtual Machines

The ASEBA IDE compiles scripts into bytecode and loads them to the nodes through the communication bus. The nodes execute the bytecode in a lightweight VM. The use of a VM instead of native code ensures safe execution because no script mistake can disturb the low-level operations of the microcontrollers. For instance, if an array is accessed out of bounds, the VM will detect it and stop the execution of the faulty event. Moreover, the VM provides independence toward the microcontroller's vendor, as any 16-bit microcontroller or better suffices to run it. The VM can also write the bytecode into flash memory to run ASEBA code when the IDE is absent.

The overhead of the VM with respect to native code is acceptable in modern microcontrollers (Sec. V-C). In our dsPIC33 implementation, the compiled VM consumes 10 kB of flash memory and 4 kB of RAM, including all communication buffers. In demanding situations, we can shrink these requirements by adjusting the amount of bytecode and variable data, stack size, and number of breakpoints.

Moreover, the use of the ASEBA VM might even increase the performance of an application when compared to a C code from a lambda user. Indeed, the ASEBA VM provides native functions that efficiently compute common mathematical operations such as the dot product. As modern microcontrollers contain optimized instructions for such computations, carefully written native functions are faster than a naive C implementation. For instance, on the dsPIC, a dot product over 100 elements programmed in C and compiled with maximum optimization runs in 871 cycles.[1] The corresponding ASEBA native function, which uses the multiply and accumulate instruction, runs in 320 cycles.

Finally, the VM is easy to understand and thus easy to adapt and optimize. Its source code counts fewer than 1000 lines of C, including the debugger core.

IV. Application to Complex Mechatronics

This section presents the application of ASEBA to a complex mechatronic system, the handbot climbing robot (Fig. 5). We show how ASEBA provides distributed control for a tightly integrated behavior involving multiple degrees of freedom. The handbot climbs a shelf and retrieves a book. To help when climbing, the handbot uses a rope to compensate for the gravity force. The handbot launches the rope before climbing using a strong spring and a rolling mechanism. The details of the handbot, in particular the rope-launching mechanism, are available in [30].

The handbot uses four microcontrollers to climb a shelf. To move, the handbot rotates its arms with respect to its body in alternate directions (Fig. 5). Every 180 degrees, the free gripper attaches to the vertical board, and the other grip is released. The handbot repeats this sequence until the handbot reaches the height of the book, where the free gripper takes the book. Then the handbot detaches its second gripper and goes back down freely lying at the rope (Fig. 6). Each high-level, conceptual event such as alternating the attached grippers translates into series of concrete events that transit over the communication bus. For instance, to attach the left gripper and detach the right one, the microcontrollers exchange seven events (Fig. 7).

Thanks to ASEBA, we managed to implement, debug, and test the controller for climbing a shelf in less than two days. In particular, the IDE proved its usefulness by allowing us to inspect the values of the sensors and monitor the events in real time. The integrated debugger, through its step-by-step mode, allowed us to easily debug the finite state machines controlling

[1] C30 3.11, based on gcc 4.0.3, -O3 optimization flag

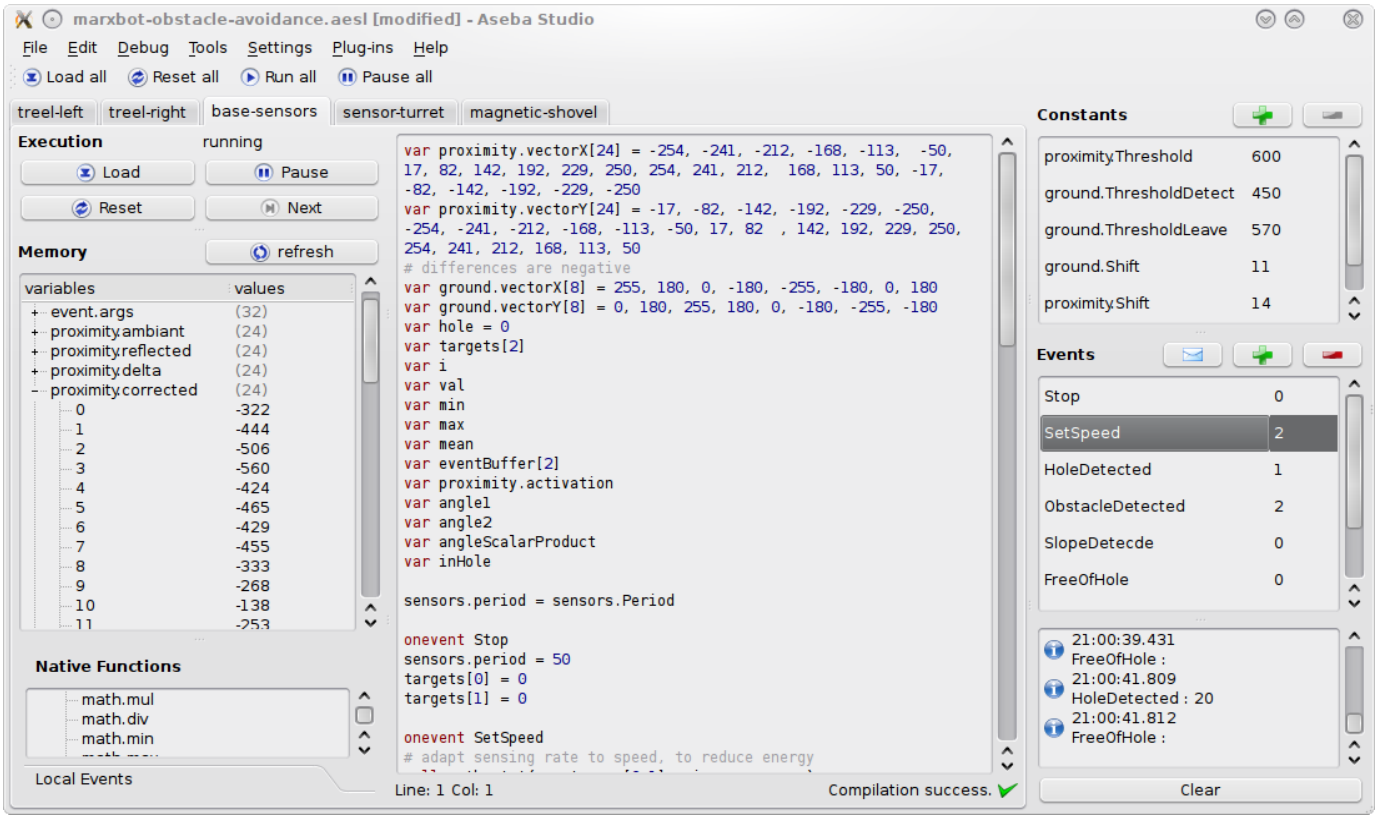


Fig. 4: Screenshot of the ASEBA IDE.

the actions of each microcontroller. Moreover, by allowing us to process information locally—for instance, the gripper decides itself to close when it senses the board—ASEBA enabled the distribution of the complex shelf-climbing behavior. As a result, the handbot is able to climb without a central computer.

V. Application to a Miniature Modular Robot

This section presents the application of ASEBA to a miniature modular robot, the marXbot (Fig. 8). We show experiments and measurements that demonstrate quantitatively the advantages of ASEBA. The base of the marXbot is a differential wheeled robot 17 cm in diameter with two wheels and two tracks. It is a robust, high-performance foundation for application-specific extensions. This base embeds three microcontrollers and provides a ring of infrared proximity sensors, infrared ground sensors, a three-dimensional accelerometers, an RFID reader, and a 40 Wh battery.

A. Bandwidth Reduction by Preprocessing

In this experiment, we show that ASEBA, by transmitting only relevant data, consumes two orders of magnitude less bandwidth than a polling-based approach. We measure the amount of data that ASEBA requires to implement obstacle avoidance on the marXbot using the 24 proximity sensors at 67 Hz for 1 minute. We repeat this test 120 times. We do so for 3 environments of different complexities, as shown in Fig. 9. We compare these measures to the requirements of a polling-based approach, which are the following:

$$b = f \cdot ((\text{sensors data}) + 2 \cdot (\text{motor commands}))$$
$$\Leftrightarrow b = f \cdot ((C \cdot S + O) + 2 \cdot (S + O))$$

where b is the required bandwidth, f is the frequency of polling (10, 25, and 67 Hz), C is the number of sensors (24), S is the number of bytes to transmit per value (2), and O is the packet header overhead (source identifier + ASEBA packet type, 3).

The measures in Fig. 9 show that, even in the most complex environment, the median bus load is 193 times lower using ASEBA than using a polling-based approach. In the worst case, ASEBA is still 179 times more efficient. This excellent performance is due to the filtering that ASEBA scripts perform on the raw data inside the microcontrollers. Moreover, thanks to the use of scripting and VMs, the robot application developer can adapt the filtering code to each scenario to maximize the bandwidth savings. Fig. 10 illustrates visually the variance of the bandwidth consumption with respect to the robot's location. The latter is proportional to the number of obstacles encountered by the robot.

While the savings are significant for the sensors of the marXbot base, we expect them to be even higher in the case of high-bandwidth sensors such as sound source detection, vision-based features detection, or laser scanners. For instance, the dsPIC microcontroller that we use is powerful enough to process sound in real time, and thus, through the use of native functions, ASEBA could process sound from several microphones and report the direction of the incoming sound.

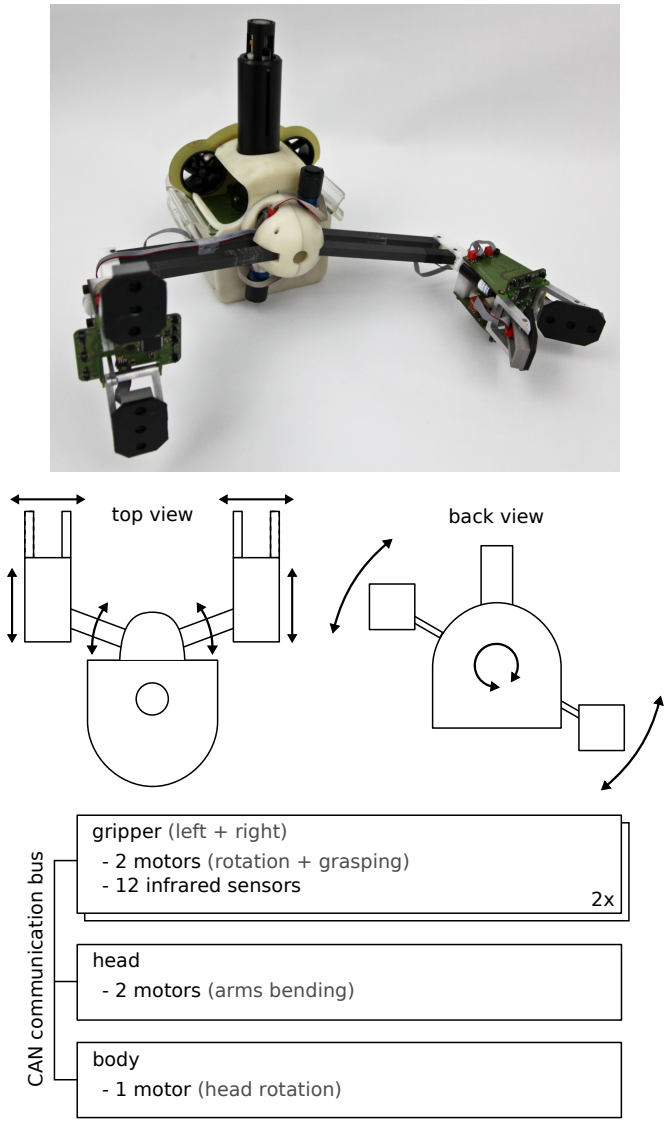


Fig. 5: The handbot robot, its degrees of freedom, and the distribution of functions between its microcontrollers for climbing a shelf. We have omitted the degrees of freedom, microcontrollers, sensors, and actuators for other operations such as launching the rope.

B. Low Latency for Fast Reactions

In this experiment, we show how ASEBA can reduce the latency between perception and action with respect to a polling-based approach. We show how a faster reaction allows the robot to stop at a larger distance from an obstacle.

The experimental setup is simple: the robot goes straight on a flat surface at 150 mm/s and stops when the front proximity sensors detect an obstacle at a distance less than or equal to 30 mm. We then measure the distance to the obstacle where the robot has finally stopped.

The measures in Fig. 11 show that the event-based control of ASEBA allows the robot to stop precisely at a long distance of the obstacle. In comparison, doing polling at the sampling frequency of the sensors, 67 Hz, results in a distance slightly smaller and more variable. We attribute this to the delay of the

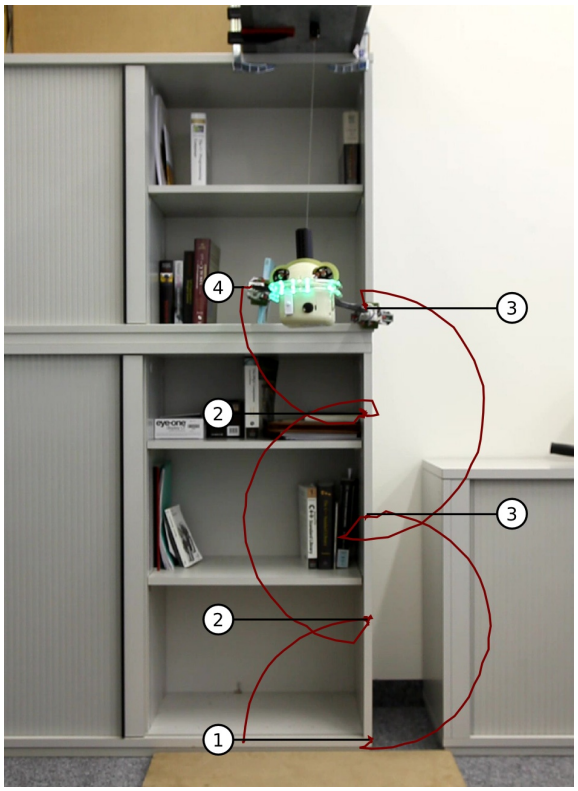


Fig. 6: High-level events of a handbot climbing a shelf to retrieve a book. The burgundy lines show the traces of the grippers. The sequence of events is as follows: (1) the right gripper attaches, the head rotates clockwise; then alternatively (2) the left gripper attaches, the right gripper detaches, the head rotates counterclockwise; (3) the right gripper attaches, the left gripper detaches, the head rotates clockwise; and finally (4) the left gripper takes the book, the right gripper detaches, the robot goes down using the rope.

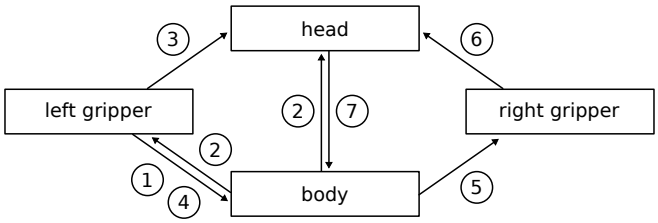


Fig. 7: Low-level events of a handbot attaching its left gripper and detaching its right one. The sequence of events is as follows: (1) the left gripper detects a vertical board; (2) the body requests the head to extend the left arm; (3) when the left gripper is close enough to the board, the left gripper instructs the head to stop extending the left arm; it begins grasping the board; (4) when the left gripper has firmly grabbed the board, the left gripper informs the head; (5) the head requests the right gripper to detach; (6) the right gripper informs the head that the gripper is not grasping the board any more, and the head retracts the right arm; (7) the head informs the body that the right arm is detached.

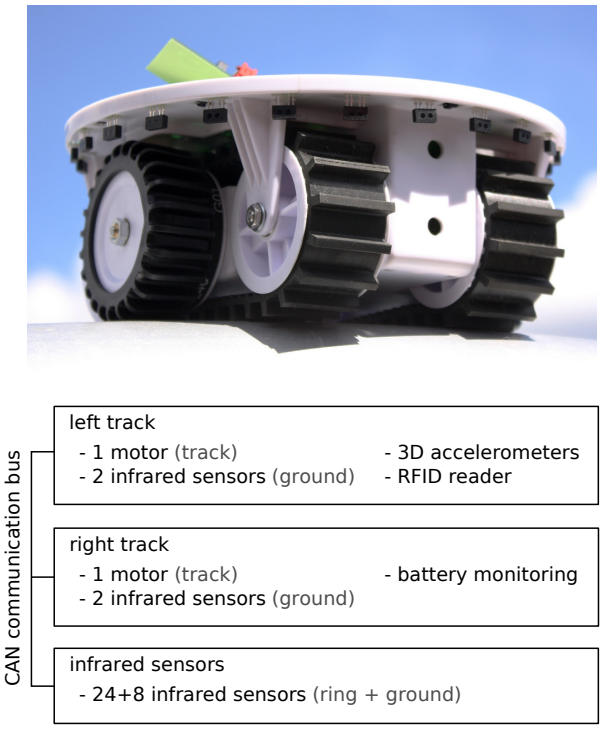


Fig. 8: The base of the marXbot robot and the distribution of functions between the microcontrollers.

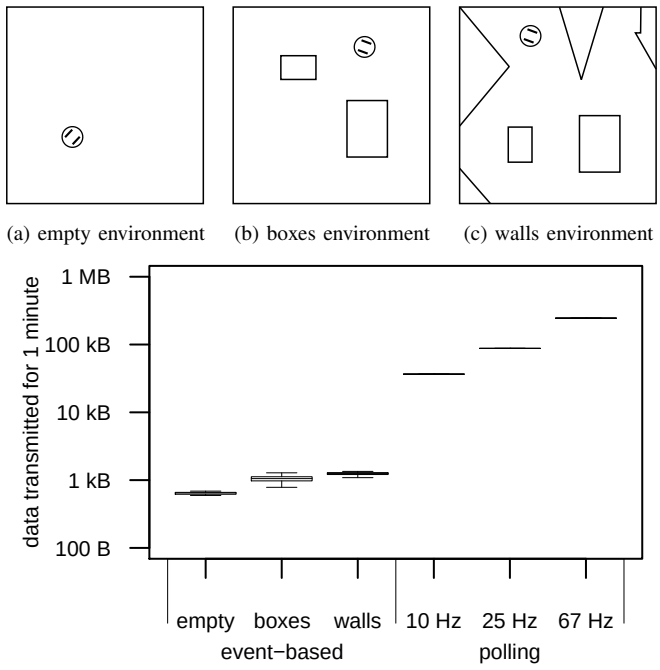


Fig. 9: Measurement of bandwidth consumption. We performed this experiment in three different environments. The box plot shows the bus load in these along with the theoretical values for polling.

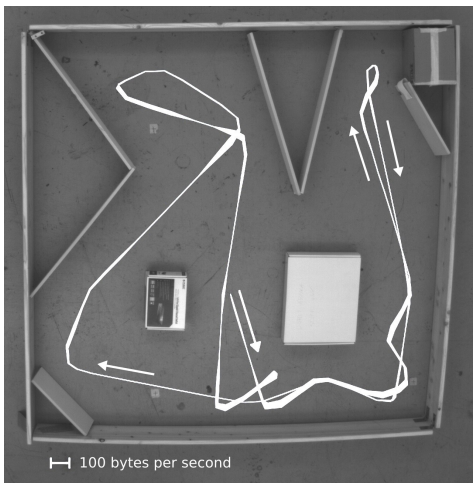


Fig. 10: Bandwidth consumption with respect to the robot's location in the wall environment (Fig. 9c). The width of the white line is proportional to the bandwidth consumption.

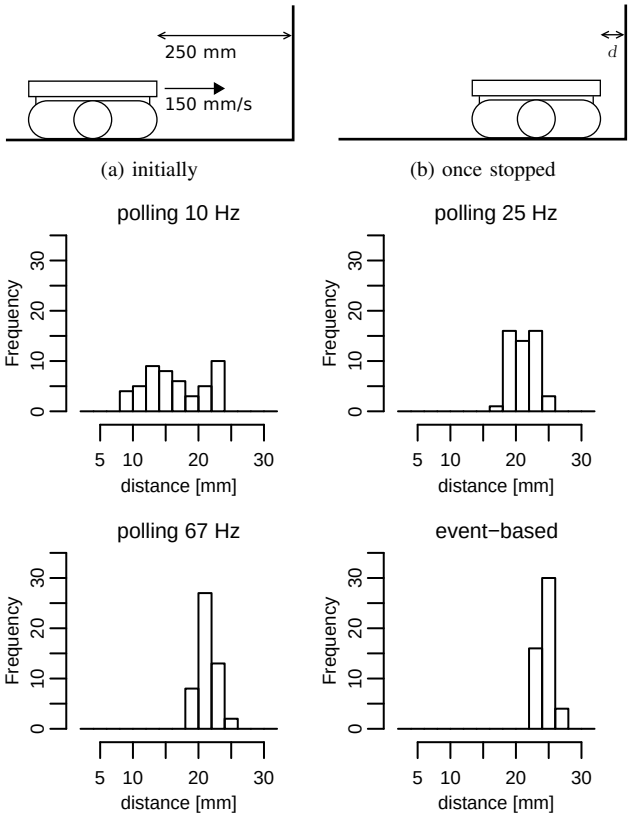


Fig. 11: Measurement of latency. The robot moves toward a wall at the speed of 150 mm/s starting from a distance of 250 mm (a). When the robot's front sensors detect the wall at a distance closer than 30 mm, the robot stops. We then measure the distance d to the wall (b). The histograms show the distribution of d.

transmission of the read command and to its aliasing with the update of the sensors values. When the frequency of the polling decreases to 25 Hz, the stop distance is clearly smaller and more variable. These effects are even more visible when polling at 10 Hz. These results show that the event-based control of ASEBA provides a reaction to input stimuli of lower latency than a polling-based control. This allows the robot to optimize its speed while operating safely.

C. Virtual Machine

The VM of ASEBA achieves respectable performance, even if we have not yet micro-optimized the VM for speed. We measured that the delay for an idle VM to react to an incoming event by sending an outgoing event, without any further processing, is 25 μs. This value will, however, slightly increase with the amount of event handling code, as the VM must find the address of the specific event in a table.

If we add a `for` loop that does an addition operation 100 times, the delay between an incoming event and the resulting outgoing one is 1.71 ms. Such a loop executes around 1000 VM instructions, which results in a VM performance of about 600,000 instructions per second. The microcontroller that we use, the dsPIC, runs at an instruction rate of 40 MIPS. This implies that the VM executes 70 dsPIC instructions per VM instruction.

The VM provides native functions for mathematical operations on arrays. Using such functions to perform 100 additions between the incoming and the outgoing events results in a delay of 60 μs.

VI. Lessons Learned and Future Works

The development of our robots taught us that the shared communication bus and its use are critical for the performances. ASEBA solves the bus overloading problems by filtering raw data inside the microcontrollers. We could further improve the scalability of ASEBA by segmenting the bus. Indeed, the compiler knows the source and destinations of all events. So the compiler could create routing tables such that events do not transit over segments of the bus that contain no destination. This does not apply to multi-master buses such as CAN, but would be of great interest for custom-tailored systems, such as the ones based on FPGA.

Our results show that a safe, event-based *scripting* language is a sound approach to the programming of real-time embedded systems. This fact provides insight into the question of scripting from a computer science point of view [31]. We could further increase the performances of ASEBA. Indeed, we could improve the compiler so that it proves more facts about the program. For instance, the optimizer could remove the boundary checks on array access in most cases. We could add timing analysis to prove that events would execute within a specified duration. In computer science, the field of research known as functional reactive programming proposes a theoretical approach to such challenges [32]. It would be interesting to explore whether and how the latter would apply to networks of microcontrollers. This would require a complete re-engineering of the language. Without such a radical redesign, we could still improve the language of ASEBA. It is currently limited by its data types, which consist only of 16-bit integers and arrays. We could extend it by adding more basic types (floating point values, 32-bit integers, ...) and arbitrary data structures. However, we must be careful when considering features that limit predictability, such as dynamic memory allocation. These could decrease the performance and the reliability of ASEBA in real-time applications.

We have discussed ASEBA in the context of miniature robots, but the results that we present apply to larger robots—or to industrial installations—as well. In these contexts, the distribution of the processing could also improve energy efficiency, because useless data are pruned early. As an additional benefit, an event-based approach could enable the creation of a failsafe behavior by using redundant hardware modules, as [11] showed.

VII. Conclusion

The experimental performances of ASEBA running in physical robots demonstrate that a modular, distributed, and event-based architecture is a pertinent solution to program the behavior of multi-processors' embedded systems.

Moreover, our results show that applying such architecture for low-level control improves complex robots in multiple ways. The closeness to the hardware allows fast *reactivity* to environmental stimuli. The exploitation of peripheral processing power provides *scalability* by filtering raw data and by implementing reflex-like control locally. This provides better behavioral performances for a given hardware and allows a smaller, cheaper, and less energy-consuming hardware to be used for equal performances. The exploitation of microcontrollers' processing power also *offloads the central computer*, when it is present. This leaves time for cognitive tasks such as path-planning or reasoning. Finally, the interactive IDE permits an efficient development process.

For all these reasons, ASEBA enables us to build more complex, more integrated, and smarter robots with today's hardware. We think that the latter characteristics are of premium importance to trigger a large-scale deployment of mobile robots in the real world.

Acknowledgment

We thank the three anonymous reviewers as well as the editors for their insightful comments. We thank Daniel Burnier and Florian Vaussard for their work on the marXbot; and Tarek Baboura for his work on the handbot. We also thank Daniel Roggen and Cyrille Dunant for their comments on the manuscript.

References

[1] R. Siegwart and I. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. MIT Press, 2004.

[2] S. Thrun, "Probabilistic robotics," *Commun. ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[3] P. Newman, D. Cole, and K. Ho, "Outdoor SLAM using visual appearance and laser ranging," in *IEEE International Conference on Robotics and Automation*, pp. 1180–1187, IEEE Press, 2006.

[4] H. Liu, P. Meusel, G. Hirzinger, M. Jin, Y. Liu, and Z. Xie, "The modular multisensory dlr-hit-hand: Hardware and software architecture," *Mechatronics, IEEE/ASME Transactions on*, vol. 13, pp. 461–469, Aug. 2008.

[5] Y. Yoon and D. Rus, "Shady3d: A robot that climbs 3d trusses," in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 4071–4076, IEEE Press, 2007.

[6] M. Mataric, N. Koenig, and D. Feil-Seifer, "Materials for enabling hands-on robotics and STEM education," in *AAAI Spring Symposium on Robots and Robot Venues: Resources for AI Education*, AAAI Press, 2007.

[7] F. Mondada, G. C. Pettinaro, A. Guignard, I. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L. Gambardella, and M. Dorigo, "SWARM-BOT: a New Distributed Robotic Concept," *Autonomous Robots, special Issue on Swarm Robotics*, vol. 17, no. 2–3, pp. 193–221, 2004.

[8] S. Magnenat, V. Longchamp, and F. Mondada, "Aseba, an event-based middleware for distributed robot control," in *Workshops DVD of International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[9] S. Magnenat, B. Noris, and F. Mondada, "Aseba-challenge: an open-source multiplayer introduction to mobile robots programming," in *Proceedings of International conference on Fun and Games*, Springer, 2008.

[10] S. Magnenat, P. Rétornaz, B. Noris, and F. Mondada, "Scripting the swarm: event-based control of microcontroller-based robots," in *SIMPAR 2008 Workshop Proceedings*, 2008.

[11] H. Durrant-Whyte, B. Rao, and H. Hu, "Toward a fully decentralized architecture for multi-sensor data fusion," in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pp. 1331–1336, IEEE Press, May 1990.

[12] J. Gil, A. Pont, G. Benet, F. Blanes, and M. Martínez, "A CAN Architecture for an Intelligent Mobile Robot," in *Proc. of SICICA-97*, pp. 65–70, 1997.

[13] I. Gravagne, J. Davis, J. Dacunha, and R. Marks, "Bandwidth reduction for controller area networks using adaptive sampling," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 5, pp. 5250–5255, April-1 May 2004.

[14] M. Szymanski and H. Worn, "Jamos - a mdl2ϵ based operating system for swarm micro robotics," in *Swarm Intelligence Symposium, IEEE*, pp. 324–331, IEEE Press, 2007.

[15] Q. Mahmoud, ed., *Middleware for Communications*. Wiley, 2004.

[16] J. Luckham, D.C.; Vera, "An event-based architecture definition language," *Software Engineering, IEEE Transactions on*, vol. 21, pp. 717–734, 1995.

[17] P. R. Pietzuch and J. Bacon, "Hermes: A distributed event-based middleware architecture," in *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pp. 611–618, IEEE Computer Society, 2002.

[18] M. Mizukawa, H. Matsuka, T. Koyama, and A. Matsumoto, "ORiN: Open Robot Interface for the Network, a proposed standard," *Industrial Robot: An International Journal*, vol. 27, no. 5, pp. 344–350, 2000.

[19] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro - middleware for mobile robot applications," *Robotics and Automation, IEEE Transactions on*, vol. 18, pp. 493–497, Aug 2002.

[20] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W.-K. Yoon, "Rt-middleware: distributed component middleware for rt (robot technology)," in *International Conference on Intelligent Robots and Systems (IROS)*, pp. 3933–3938, IEEE Press, 2005.

[21] M. Henning, "A new approach to object-oriented middleware," *Internet Computing, IEEE*, vol. 8, pp. 66–75, Jan–Feb 2004.

[22] H. Bruyninckx, "Open robot control software: the orocos project," in *International Conference on Robotics and Automation (ICRA)*, pp. 2523–2528, IEEE Press, 2001.

[23] L. Petersson, D. Austin, and H. Christensen, "DCA: A Distributed Control Architecture for Robotics," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pp. 2361–2368, IEEE Press, 2001.

[24] D. Simon, B. Espiau, K. Kapellos, and R. Pissard-Gibollet, "Orccad: software engineering for real-time robotics: A technical insight," *Robotica*, vol. 15, pp. 111–115, 1997.

[25] W.-M. Shen and M. Yim, eds., *Mechatronics, IEEE/ASME Transactions on, special issue on self-reconfigurable robots*. IEEE Press, 2002. vol 7, issue 4.

[26] Y. Zhang, K. Roufas, and M. Yim, "Software architecture for modular self-reconfigurable robots," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, pp. 2355–2360, IEEE Press, 2001.

[27] R. A. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, 1986.

[28] E. Gat, "On three-layer architectures," in *Artificial Intelligence and Mobile Robots* (D. Kortenkamp, R. P. Bonnasso, and R. Murphy, eds.), pp. 195–210, MIT/AAAI Press, 1997.

[29] I. Standards, *Road Vehicles Interchange of Digital Information - Controller Area Network - ISO 11898*. International Organization for Standardization, 1993.

[30] M. Bonani, S. Magnenat, P. Rétornaz, and F. Mondada, "The Handbot, a Robot Design for Simultaneous Climbing and Manipulation," in *Proceedings of the Second International Conference on Intelligent Robotics and Applications* (M. Xie et al., ed.), vol. 5928 of *Lecture Notes in Computer Science*, pp. 11–22, Springer-Verlag, 2009.

[31] R. P. Loui, "In praise of scripting: Real programming pragmatism," *Computer*, vol. 41, no. 7, pp. 22–26, 2008.

[32] P. Hudak, A. Courtney, H. Nilsson, and J. Peterson, "Arrows, robots, and functional reactive programming," in *Summer School on Advanced Functional Programming 2002, Oxford University*, vol. 2638 of *Lecture Notes in Computer Science*, pp. 159–187, Springer-Verlag, 2003.