Software integration in mobile robotics, a science to scale up machine intelligence

Stéphane Magnenat

École Polytechnique Fédérale de Lausanne

7 août 2010

Abstract

The present work tackles integration in mobile robotics. Integration is often considered to be a mere technique, unworthy of scientific investigation. On the contrary, we show that integrating capabilities in a mobile robot entails new questions that the parts alone do not feature. These questions reflect the structure of the application and the physics of the world. We also show that a succesful integration process transforms the parts themselves and allows to scale up mobile-robot intelligence in real-world applications.

In Chapter 2 we present the hardware. In Chapter 3, we show that building a low-level control architecture considering the mechanic and electronic reality of the robot improves the performances and allows to integrate a large number of sensors and actuators. In Chapter 4, we show that globally optimising mechatronic parameters considering the robot as a whole allows to implement SLAM using an inexpensive sensor with a low processor load. In Chapter 5, we show that based on the output from the SLAM algorithm, we can combine infrared proximity sensors and vision to detect objects and to build a semantic map of the environment. We show how to find free paths for the robot and how to create a dual geometric-symbolic representation of the world. In Chapter 6, we show that the nature of scenarios influences the implementation of a task-planning algorithm and changes its execution properties. All these chapters contribute results that together prove that integration is a science.

In Chapter 7, we show that combining these results improves the state of the art in a difficult application : autonomous construction in unknown environments with scarce resources. This application is interesting because it is challenging at multiple levels : For low-level control, manipulating objects in the real world to build structures is difficult. At the level of perceptions, the fusion of multiple heterogeneous inexpensive sensors is not trivial, because these sensors are noisy and the noise is non-Gaussian. At the level of cognition, reasoning about elements from an unknown

world in real time on a miniature robot is demanding. Building this application upon our other results proves that integration allows to scale up machine intelligence, because this application shows intelligence that is beyond the state of the art, still only combining basic components that are individually slightly behind the state of the art.

Keywords : miniature mobile robot, integration, machine intelligence, scalability, event-based architecture, SLAM, symbol grounding, HTN planning, autonomous construction, scarce resources, design methodology

Résumé

Ce travail étudie la question de l'intégration de systèmes en robotique mobile. L'intégration est souvent considérée comme purement technique, sans intérêt scientifique intrinsèque. Au contraire, nous montrons qu'intégrer des capacités dans un robot mobile suscite de nouvelles questions que ne posent pas les différentes parties individuellement. Ces questions reflètent la structure de l'application et la physique du monde. Nous montrons qu'un processus d'intégration, s'il réussit, transforme les parties elles-mêmes et permet d'étendre l'intelligence des robots mobiles dans leurs applications au monde réel.

Dans le chapitre 2, nous présentons les robots utilisés comme cadre expérimental. Dans le chapitre 3, nous montrons que la prise en compte des réalités mécaniques et électroniques du robot lors de la construction d'une architecture de contrôle bas niveau permet d'en améliorer les performances et d'intégrer un grand nombre de capteurs et d'actuateurs. Dans le chapitre 4, nous montrons qu'en considérant le robot comme un tout pour optimiser globalement ses paramètres mécatroniques, il devient possible d'implémenter du SLAM avec un capteur bon marché et une puissance de calcul limitée. Dans le chapitre 5, nous montrons que nous pouvons nous baser sur la sortie de l'algorithme de SLAM pour combiner les capteurs infrarouges de proximité et la vision pour détecter des objets et pour construire une carte sémantique de l'environnement. Nous montrons comment trouver des chemins libres pour le robot et comment créer une représentation duale du monde, géométrique et symbolique. Dans le chapitre 6, nous montrons que la nature des scénarios influence l'implémentation d'un algorithme de planification de tâches et change ses propriétés d'exécution.

Dans le chapitre 7, nous montrons qu'en combinant ces résultats, nous pouvons améliorer l'état de l'art d'une application difficile : la construction autonome dans un environnement inconnu avec des ressources rares. Cette application est ambitieuse à plusieurs titres : pour le contrôle bas niveau,

la manipulation d'objets pour construire des structures est difficile dans le monde réel. Au niveau de la perception, la fusion des données de multiples capteurs hétérogènes et bon marchés n'est pas triviale, car ces capteurs sont bruités d'un bruit non gaussien. Au niveau de la cognition, raisonner en temps réel sur un robot miniature à propos des éléments d'un monde inconnu est ardu. Construire cette application sur nos résultats précédents prouve que l'intégration permet d'étendre l'intelligence des machines. En effet, cette application démontre une intelligence qui dépasse l'état de l'art, en ne combinant que des composants de base en deçà de l'état de l'art.

Mots clés : robot mobile miniature, intégration, intelligence machine, extensibilité, architecture basée sur des événements, SLAM, ancrage de symboles, planification HTN, construction autonome, ressources rares, méthodologie de conception.

Acknowledgements

First and foremost, I deeply thank my supervisor, Francesco Mondada, for having welcomed me to make a PhD in his group. Francesco has given me a wise combination of freedom and excellent advices. This balance is very hard to attain, and Francesco is one of the few scientists of my knowledge who achieves it. For these reasons, and also because Francesco is a great human being who casts trust and friendship within his research group, I cannot imagine a better place than Mobots to have conducted my PhD work.

I thank the research project *Swarmanoid* (IST-FET grant 022888), as well as the EPFL, for funding me during this work. I also thank the European and Swiss taxpayers for financing scientific research.

I express my gratitude to the thesis committee members: Prof. Luca M. Gambardella, Prof. Alcherio Martinoli, Dr. Roland Philippsen and Prof. Eduardo Sanchez for accepting to evaluate this work. I am thankful for their careful reading of my work and for their insightful and constructive comments.

I thank Prof. Owen Holland who was kind enough to read and comment a draft of the proposal for this project. His comments allowed me to start quickly in a good direction.

I specially thank Martin Voelkle who helped me implementing Planner 9 during his free time and for proofreading the thesis report. Albeit currently not officially doing research, Martin has the disinterested curiosity which is at the core of intellectual progresses. In a scientific world where competition and short-term rewards are the norm, Martin restores my trust in human beings.

I thank Jean-Cédric Chappellier for our games of Go and for helping me with the proofs related to Planner 9. I thank Antoine Beyeler for being a great person to code with.

I am grateful to my lab mates: Michael Bonani, Philippe Rétornaz, Alexey Gribovskiy, Valentin Longchamp, Daniel Burnier, Frédéric Rochat,

Patrick Schoeneich, Daniel Burnier, Tarek Baaboura, Pierre Noirat, Fanny Riedo, Mariza Freire, Roderich Gross, Anne Remillet-Schaller and Florian Vaussard. You have always been helpful and have created a warm atmosphere of collaboration in the lab. Our level of collaboration is higher than the norm, and this has risen my quality of life during the last four years. The details of your individual contributions are available in Section 9.1. I specially thank Alexey Gribovskiy for your kindness as my office mate during the last years.

I would not have done this work nor become who I am without the influence of two creative and highly motivating people: Jean-Daniel Nicoud and Sebastian Gerlach. Thank you for sharing your knowledge and enthusiasm with me.

I am grateful to my close scientist friends: Cécile Grivaz, Emmanuel Eckard, Elisa Laurenti, Cyrille Dunant and Daniel Roggen. Your availability, your friendship, your kindness and your intelligence have made my PhD years a very happy time. I am specially grateful to Emmanuel Eckard for proofreading the thesis report.

I thank Jean-David Maillefer for the joyful Starcraft games that made me forget, for short durations, the thesis report to write. I thank Basilio Noris and Marion Haemmerli for your friendship and our discussions about art, life and the rest. I thank Lynda Metref and Etienne Dysli for your friendship and our good time together.

I thank Julien Pilet and Nelly Niwa for being kind and welcoming regardless of your location in the world.

I thank Vera Janowski for your kindness and your help with the German language.

I am deeply thankful to Jacques Paul Grivaz, for our unique collaboration and friendship. Jacques, making music with you restores the fragile life balance that research always jeopardizes.

I also thank all my friends who have not directly contributed to this work, but whose friendship and presence have made this work possible.

Finally, I thank my parents and my family for their kindness and for their unconditional support during my long studies.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

Chapter 1

Introduction

Mobile robotics has made indisputable progresses in the last decades. The improvements in energy storage now allow a simple and light Lithium battery to power demanding sensors and actuators for hours. The availability of inexpensive and highly integrated embedded application processors, a consequence of the massive market of mobile phones, has brought advanced capabilities to small mobile robots, such as high-resolution vision. Moreover, the improvements in electronics and material science have lowered the cost of sensors and actuators, and have improved their quality. The combination of all these advances has thus led to affordable robot hardware and allowed sophisticated control software.

Today—in research—autonomous mobile robots map unknown environments, detect objects and people, and run for hours. Yet there are few end-user applications of mobile robotics using such capabilities. In commercial applications, most mobile robots are either remotely steered or perform tasks requiring trivial reactive control. Commercial applications of mobile robotics only show a weak level of intelligence [1] because they fail to combine the advanced capabilities found in research.

Combining several capabilities is an integration problem; and following Occam's razor principle [2], researchers of a particular field usually simplify aspects related to other fields to the maximum. For instance, to study mapping one can combine a commercial mobile base, a standard laser scanner and a laptop computer. As a result, integration [3] is often considered to be a mere technique and thus unworthy of scientific investigation. Yet this is true only as long as methods and results of different

1. We define what we mean by intelligence in Section 1.1.3
2. "Plurality ought never be posed without necessity" [107]
3. We define what we mean by integration in Section 1.1.1

fields are compositional, that is, when the combination of the methods does not change the results. Most researchers quietly assume this as a premise.

On the contrary, our first hypothesis is that the different aspects of a mobile robot, such as its mechatronic structure, its sensors and actuators, its communication capabilities, its envisaged application, etc. all influence each other. For instance, the way sensors and actuators are distributed inside the robot influences the control architecture, or, in a multi-robot scenario, the number of robots influences the reasoning algorithms. Therefore, as the integration choices reflect the inner structure of the application and the physics of the world, and because the integration process modifies what is being integrated, integration is a science [4] of its own, and not a mere technique of assembly.

Hypothesis 1.1. *Integration is a science. Indeed, integrating capabilities in a mobile robot entails new questions that the parts alone do not feature. These questions reflect the structure of the application and the physics of the world. Moreover the integration process modifies the parts themselves.*

Let us now admit that integration is a science, that there are generic principles that govern how to combine things together to achieve a given result. In particular, we suppose that we understand and have models of how the different elements of a robot influence each other, and that we can put a figure on these influences. Thus we can alleviate the adverse effects of these influences and use them to create synergies instead. At the level of perception, we can take advantage of the complementarity between the different sensors to scale up the richness and the diversity of the world representation. At the level of action, we can work around the weaknesses of a particular platform and devise robust plans that allow an imperfect robot to achieve complex goals. Because of these improvements, the robot has a better picture of the world and a clearer understanding of its capabilities. Therefore, it can perform more sophisticated reasoning and thus displays a more intelligent behaviour. This leads us to our second hypothesis.

Hypothesis 1.2. *The science of integration allows to scale up mobile-robot intelligence in real-world applications.*

In this work, we test these hypotheses by identifying influences between the experimental scenarios, the mechanic and the electronic structure of

4. We define what we mean by science in Section 1.1.2

the robots, and the algorithms. We show that taking the mechatronics and the application scenario into account while implementing the algorithms improves the efficiency of the robots and unlocks new capabilities. We also show that combining these capabilities allows to create applications where the robot shows a high level of intelligence.

1.1 Definitions

Our hypotheses call for the clarification of the meaning of the terms *integration*, *science* and *intelligence* in the context of this work.

1.1.1 Integration

By *integration*, we mean the process of combining together different elements or components into a global system, typically a robot. Works in the middleware field have already extensively studied the software aspect, see for instance [14,19]. However, in our case we wish to extend the process of integration to the consideration of how mechanical and electronic specificities affect the software. So while we do not aim at a complete co-design methodology, we are interested in the influences of hardware on software. As we will see for instance in Chapter 3, these influences can be as deep as to force a re-definition of the control architecture.

1.1.2 Science

By *science*, we mean a process of gathering and streamlining knowledge into testable theories. In the context of robotic integration, this means looking for patterns that recurrently appear independently of the parts being integrated. The Popperian school considers that scientific theories must be falsifiable [87]. However, in this work we do not aim at theories that are directly falsifiable, but rather at finding recurrent patterns out of which we can make educated guesses for future designs of robotic control software. This corresponds to a Bayesian rather than to a Popperian view of science [16].

1.1.3 Intelligence

Intelligence does not admit a single authoritative definition. Therefore we fall back on a functional one. We propose to employ [43, p. 13]:

> A very general mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience. It is not merely book learning, a narrow academic skill, or test-taking smarts. Rather, it reflects a broader and deeper capability for comprehending our surroundings—"catching on", "making sense" of things, or "figuring out" what to do.

This definition encompasses what is generally accepted in robotics as elements critical to intelligence. These include both classical artificial intelligence (sense/plan/act paradigm) and more adaptive approaches (through the learning aspect).

1.2 Approaches to robotic integration

Albeit most roboticists consider integration to be a mere technical problem, several specific research fields have addressed questions related to integration. In these works, integration is sometimes considered explicitly, but often implicitly. Moreover, each of these communities has concentrated on a particular aspect of integration, and neglected the others. No current work in mobile robotics has addressed it as a whole, as a scientific question in itself. In this section we briefly survey the main existing approaches, to introduce enough background knowledge for the main discussion on our work.

1.2.1 Software middleware

Section 3.1 (p. 24) provides a survey of the literature on robotic software middlewares, in this section we discuss their properties. The middleware approach conceives the question of integration as a software engineering problem. It proposes to decompose robot control algorithms into self-contained modules with well-defined interfaces, called *components*. Proxy components endow sensors and actuators with compatible interfaces, such that application developers can connect them into control components. While we agree that components do ease the integration of complex controllers and promote reuse of code across applications, they only solve part of the problem. Indeed, first they ignore the mechatronic aspect of the integration problem. Building an effective robot controller demands more than programming algorithms: It depends on the correct spatial and temporal synchronisation between mechanical elements, and is tightly

linked with the capabilities of the robot's sensors, actuators and computer, and with the limitations of the buses connecting these together. These require testing and experimentation—or a complete modelling—and are not easily abstracted away by software interfaces. Second, components are fixed black boxes that we can only connect together. As we shall see later in this report, integrating algorithms together demands that we adapt these algorithms, and thus that we modify the internals of the components. Albeit components might provide parameters to control their internal processes, adjusting these parameters cannot address all possible situations. Finally, while components' implementations can often run distributed on a network, they typically require a fixed topology with reliable connections. They do not provide mechanisms to cope gracefully with abrupt disconnections. This currently does not make them suitable as the backbone software infrastructure for realistic collective-robotic applications. And yet most robotic applications would require several robots, or at least robots in communication with surrounding equipments.

1.2.2 Cognitive and control architectures

Cognitive and control architectures, if understood in the broad sense, are also attempts to address the problem of integrating intelligent behaviours. Contrary to middlewares, cognitive architectures do not explore how to connect components together, but rather what elements are needed to obtain intelligent behaviours. Early cognitive architectures, such as ACT-R [3] and Soar [60], stem from psychology and aim at modeling the human cognition. They consider that noiseless symbolic inputs are available and ignore the problems of perception and action. Some more recent works, such as [99], are closer to the robotic reality and have led to functional implementations in robots, for example [50]. However, even in these works, the cognitive architecture only solves the high-level problems, and there is a huge amount of ad hoc code and infrastructure to allow the robot to peform the task. Moreover, at the level of cognition the task is simple and thus current results do not provide a strong evidence of the integration capability of the architecture.

In total opposition to the abstract and symbolic approach of early cognitive architectures, the field of control architectures consider the problem from the perspective of the robot. Albeit cognitive architectures can control the robot as well, when we speak about control architectures, we restrictively mean the field that focuses on how to enable the robot to perform tasks, rather than to model what the robot should think. In this

field, some works such as the subsumption architecture [12] propose a parallel processing of the information flow with no explicit representation at the level of the architecture. Other works, such as HCS [2], propose a hierarchical decomposition with intermediate representations. These architectures are very generic and in this sense could be interpreted as models of middlewares. The subset of the multisensor data fusion research field that deals with robotics could also be considered as studying control architectures [94, ch. 25]. Works from this field carefully consider the technical characteristics of the sensors, but generally they do not address the methodology of the choice of the sensors.

Some works, such as the 3T architecture [9], have proposed to integrate both control and cognition into a unified layered architecture, with bi-directional connections between the layers. As with all existing architectures, 3T is a very general framework that does not approach the question of the integration of the mechatronic aspects. Yet we think that this direction is the most realistic for implementing robotic control between microcontrollers and a main computer (see Chapter 2), and we can interpret our application in the light of this architecture. In this case, aseba (see Chapter 3) implements the low-level layer while the main computer runs the other layers.

1.2.3 Embodiment and the exploitation of morphology

In opposition to the proponents of cognitive architectures, the advocates of embodiment claim that the cognitive capabilities depend deeply on the hardware of the robot. While this argument was originally used in favour of control architectures versus cognitive architectures, recently its proponents argued in favour of a more extreme exploitation of the physical body of the robot. Indeed, recent studies [113] show that the morphology and the material properties can deeply affect the efficiency of a system design. Based on this idea, researchers have exploited the morphology of the robot to perform tasks such as passive walking [18], fast running [57] and climbing almost vertical surfaces [111]. These results are interesting and show the importance of considering the mechanic of the robot while doing system design, however they do not approach cognitive tasks nor do they propose solutions to build versatile robots. We thus think that exploiting the morphology is a useful idea as part of a larger methodology, but that alone it is limited to trivial scenarios.

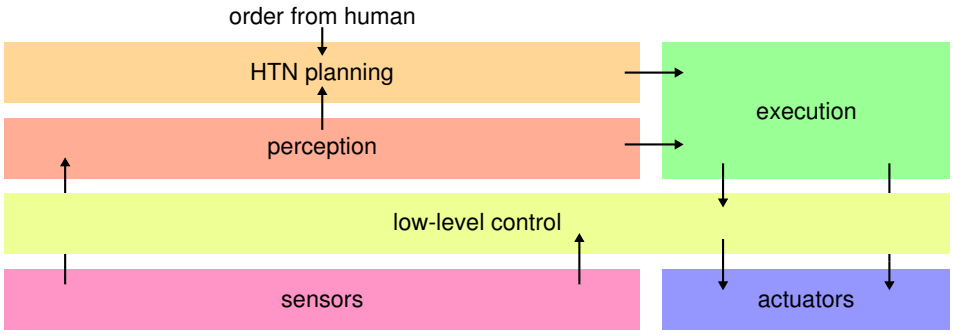


Figure 1.1 The bloc scheme of the autonomous construction application.

1.3 Outline

We begin this report by presenting the hardware in Chapter 2. In Chapter 3, we show that building a low-level control architecture considering the mechanic and electronic reality of the robot improves the performances and allows to integrate a large number of sensors and actuators. In Chapter 4, we show that globally optimising mechatronic parameters, for instance the error on the odometry, considering the robot as a whole allows to implement simultaneous localisation and mapping (SLAM) using an inexpensive sensor with a low processor load. In Chapter 5, we show that based on the output from the SLAM algorithm, we can combine infrared proximity sensors and vision to detect objects and to build semantic maps of the environment. We show how to find free paths for the robot and how to create a dual geometric-symbolic representation of the world. In Chapter 6, we show that the nature of collective-robotic scenarios influences the implementation of a task-planning algorithm and thus changes its execution properties. All these chapters contribute results that together prove Hypothesis 1.1.

In Chapter 7, we show that combining these results improves the state of the art in a difficult application: autonomous construction in unknown environments with scarce resources. In this application, a robot explores its environment and builds structures in it, following orders from a human. At the beginning, the robot does not know the geometry and the topology of the environment; and as the resources are scarce, it must use them parsimoniously. Figure 1.1 shows the system-level bloc scheme of this application; we will refer to this global picture throughout this report. This application is interesting because it is challenging at multiple levels: For low-level control, manipulating objects in the real world to build

structures is difficult. At the level of perceptions, the fusion of multiple heterogeneous inexpensive sensors is not trivial, because these sensors are noisy and the noise is non-Gaussian. At the level of cognition, reasoning about elements from an unknown world in real time on a miniature real robot is demanding. Building this autonomous construction application upon our other results proves Hypothesis 1.2, because we demonstrate an application showing machine intelligence that is beyond the state of the art, still only combining basic components that are individually slightly behind the state of the art.

In Chapter 8, we discuss the lessons learnt by doing this work. Finally, in Chapter 9, we summarise our contributions and provide outlooks.

Chapter 2

Hardware

As material to explore the question of integration, we use two robots that our laboratory, EPFL-LSRO-Mobots, has developed. The first robot is the *marXbot*, a modular ground robot. We have extended the marXbot with a magnetic manipulator, which allows the robot to displace objects and to act on its environment. The marXbot is our main experimental platform; Figure 2.1 shows the hardware part within the bloc scheme of the autonomous construction application. The second robot is the *handbot*, a climbing robot.

2.1 Common architecture

Our two robots share a common system architecture (Figure 2.2): microcontrollers manage sensors and actuators (Section 2.1.1) while a central embedded computer runs Linux and takes care of high-level tasks

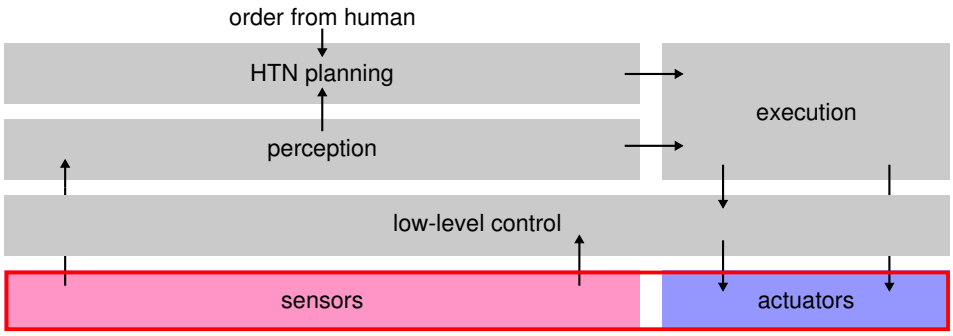


Figure 2.1 The hardware part within the bloc scheme of the autonomous construction application.

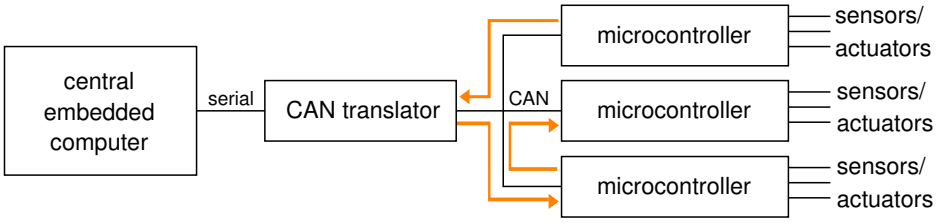


Figure 2.2 Overview of the architecture of our robots

such as vision (Section 2.1.2).

2.1.1 Microcontrollers

At the electronic level, most hardware peripherals—sensors and actuators—require sequences of input signals to control their functioning. For instance, a motor requires a pulse-width modulation and an infrared proximity sensor demands two subsequent acquisitions, one for the ambient intensity and one for the reflected intensity. The timings of these signals must be precisely set for the peripherals to work correctly. Moreover, as output the hardware peripherals produce streams or sequences of data, analog or digital. For instance, the output of an infrared sensor is an analog voltage, which must be converted into a digital value for use by the control algorithms.

To control hardware devices with precision and in real time, the best solution is to reduce wiring and thus to place microcontrollers physically close to the devices. A single powerful central embedded computer is not sufficient, because it does not provide enough interfaces such as timers, analog-to-digital converters or general-purpose input/output pins. Moreover, even if it did, the large number of wires required to connect the different locations of the peripherals to the central computer would prohibit this solution.

Thus, at the electronic level, our robots embed a multitude of microcontrollers distributed wherever they are needed. These microcontrollers are connected together through a shared communication bus. We decided to use the controller-area network (CAN) bus [102], because it is readily available in common microcontrollers and is capable of multi-master operations. We employ 16-bit microcontrollers from the Microchip dsPIC33 family, because they provide a rich variety of interfaces and incorporate instructions optimised for digital signal processing. To programme and take advantage of this network of distributed processors, we have developed a

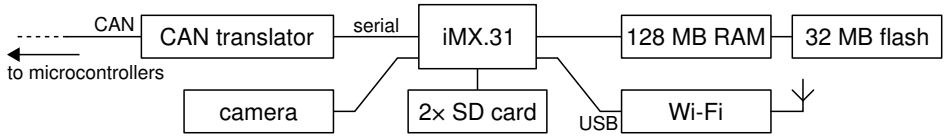


Figure 2.3 Schematic of the central computer of our robots

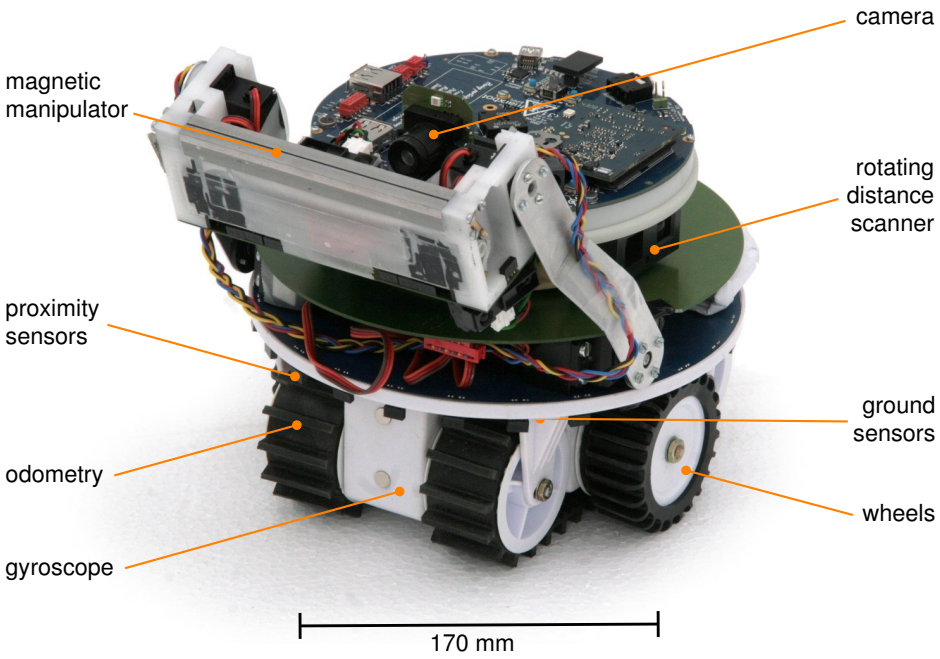
low-level event-based control architecture, detailed in Chapter 3.

2.1.2 Central computer

Microcontrollers are well-suited for managing peripherals with an output bandwidth in the order of some kB/s. However, some sensors such as cameras have a much higher bandwidth and produce data that require heavy processing. In addition, high-level tasks such as probabilistic map building or symbolic reasoning also require a large computational power. To address these needs, our robots embed a central computer. This computer is based on a 533 MHz Freescale i.MX31 application processor and 128 MB of RAM (Figure 2.3). The i.MX31 embeds an ARM 11 core, memory controllers, a camera interface and several serial ports. This computer runs Linux 2.6.33 and the OpenEmbedded development environment. The computer communicate with the microcontrollers through a translator that links the CAN network with a serial port. To provide high-quality vision to the robots, we have connected a three-megapixel camera to the i.MX31. The i.MX31 acquires the camera data in hardware using a direct memory access (DMA) controller, and thus the image is available in the main memory without any software processing.

2.2 MarXbot

In this section, we present the marXbot miniature mobile robot in the configuration that we use for our experiments (Figure 2.4). The marXbot is a modular robot; for our experiments we use a configuration with 4 modules. A base module provides energy, mobility, inertial measurement and short-range sensing (Section 2.2.1). Situated on top of the base, a magnetic manipulator module (Section 2.2.2) allows the marXbot to grasp, position and drop any light object whose side is ferromagnetic. Above the manipulator, a distance scanner module (Section 2.2.3) allows the robot to build a two-dimensional map of its environment. Finally, at the top of the robot, the main computer board runs Linux, manages vision and provides



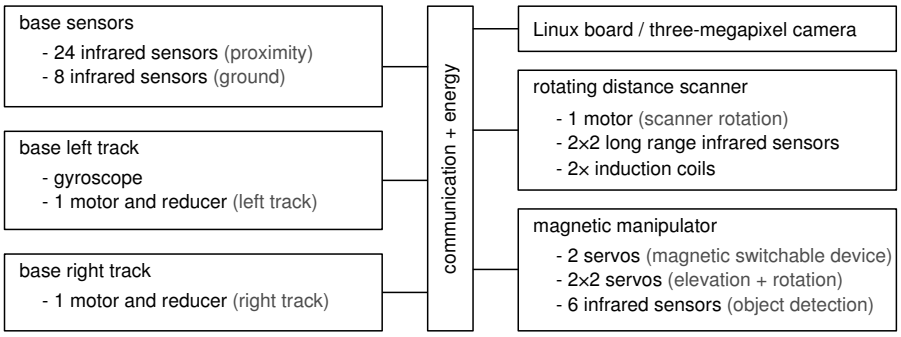


Figure 2.4 The marXbot (top) and an overview of its electronic schematic (bottom), both showing the configuration that we use for our experiments. All motors have position, speed and current sensors.

high-level cognitive capabilities. There exist other modules, for example for inter-robot assembling or localisation. We do not use these modules in this work and thus we do not describe them.

2.2.1 Base module

The marXbot is built upon a base module that provides rough-terrain mobility thanks to treels, a combination of tracks and wheels [72]. The treels provide good mobility in rough terrain at the expense of the precision of the odometry. In particular, the track part of the treel slips on the ground when the robot turns, which results in wrong odometry readings during rotations. To compensate, the base module embeds a gyroscope that can precisely measure the rotation of the robot for short durations. The base module embeds a ring of 24 short range infrared proximity sensors that allow the robot to perceive and to avoid obstacles. The base module also contains 8 infrared ground sensors that allow the robot to measure the distance to the ground and thus to avoid holes. For energy, the base module contains a slot for a swappable battery that powers the robot. This 38 Wh lithium polymer battery provides up to 7 hours of continuous operation when moving around and using the scanner. If the robot uses its manipulator a lot, the autonomy drops slightly but is still over 4 hours.

2.2.2 Magnetic manipulator

We designed the magnetic manipulator to endow the marXbot with the ability to grasp, displace and position small objects. This allows the marXbot to build structures and to manipulate its environment (Chapter 7). The magnetic manipulator features 6 infrared proximity sensors, which allows the marXbot to precisely align itself with the objects to grasp (Figure 2.5, top) As the marXbot can rotate on the spot and move freely on the ground plane, the manipulator has only three degrees of freedom. The robot can elevate and rotate its manipulator to position an object at a given altitude and pitch angle. The manipulator uses a magnetic switchable device [90] to implement the prehension. This device consists of a permanent magnet that rotates inside two pieces of metal (Figure 2.5, bottom). Depending on the orientation of the magnet with respect to the pieces of metal, the magnetic flux is either open or closed. When the flux is open, the device grasps external ferromagnetic objects; when the flux is closed, the device does not attract external objects. This device is

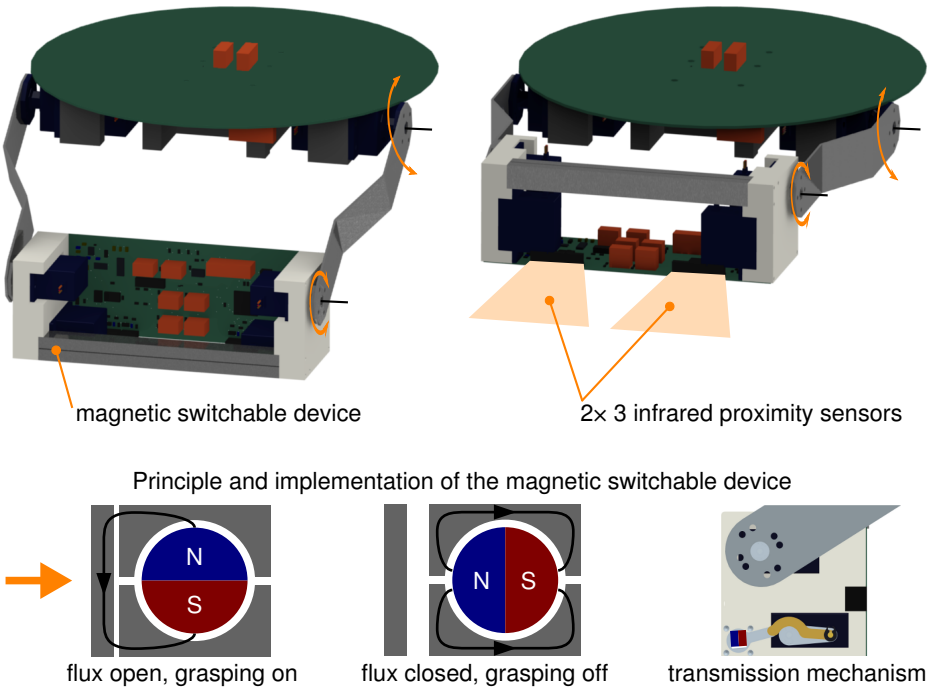
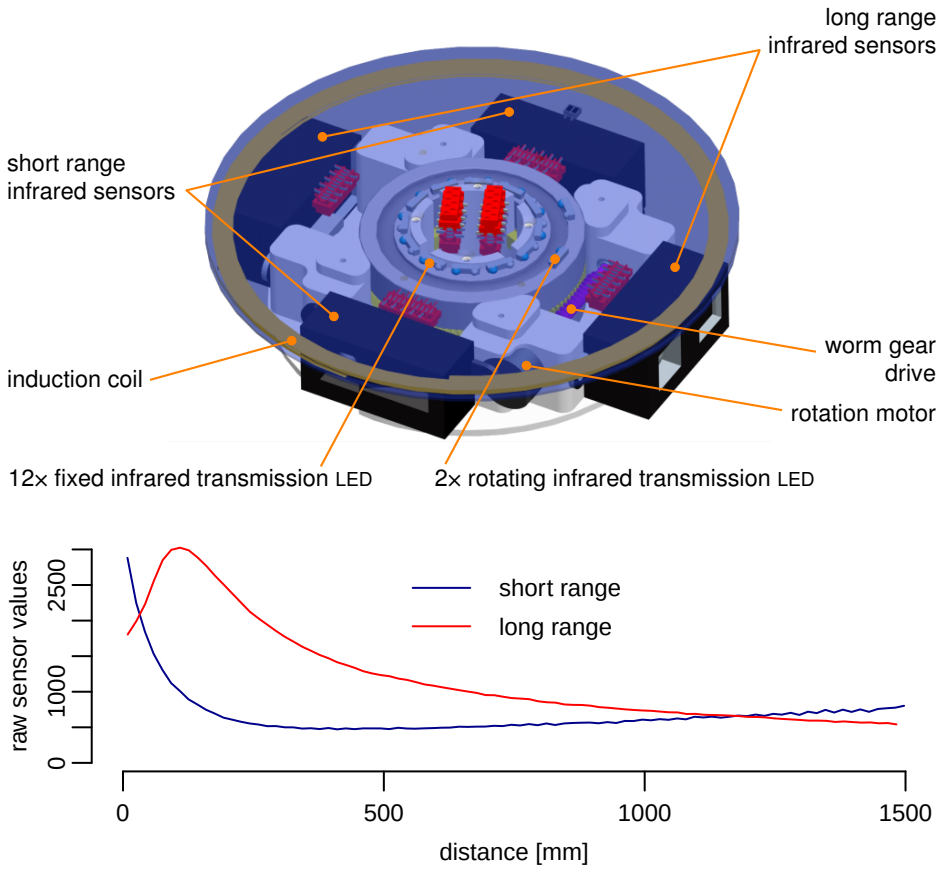


Figure 2.5 The magnetic manipulator module

bi-stable, and thus does not consume energy excepted when changing states (opening or closing). This operation lasts about 1 second.

2.2.3 Rotating distance scanner

The rotating distance scanner of the marXbot aims at being compact, inexpensive and energy efficient. The design is based on 4 infrared sharp distance sensors mounted on a rotating platform (Figure 2.6). These sensors have a limited range, a dead zone close to the device and a non-monotonic response function (Figure 2.6, centre). To disambiguate the readings, the scanner couples two sensors of different ranges (40–300 mm and 200–1500 mm). A probabilistic processing of the raw values will allow to retrieve the real distances (Chapter 4). The platform rotates continuously to make 360° scans; as it embeds two sensors of each type, the robot gets a full scan every 180°. A motor with a worm gear drives the rotation while two plastic ball bearings ensure the guidance. The motor is located in the rotor, to ease the synchronisation between the platform's position and the scanner's values. This location also fits well within the geometrical constraints of the robot and allows for a slim design. To minimise the wear



specification	value
diameter	130 mm
height	29 mm
weight	220 g
power consumption	2 W
cost	$<$ 300 € in small series
infrared distance sensors	2× GP2Y3A001K0F (4–30 cm)
	2× GP2Y3A002K0F (20–150 cm)
global operating distance	4 to 150 cm
maximum scan speed	2 scans/s
angular resolution	3° at 1 scan/s, 6° at 2 scan/s

Figure 2.6 The rotating distance scanner module. A CAD view (top), the response functions of the infrared distance sensors (centre) and the technical characteristics (bottom).

and maximise the life time of the scanner, the stator transfers energy to the rotor by induction; and the two parts exchange data using infrared light. This solution, albeit more difficult to implement than sliding contacts, is much more reliable and lasting. Induction is implemented directly on two printed circuit boards (PCBs) spaced by a gap of 0.8 mm.

The electronics of the scanner is distributed between the two PCB. The fixed PCB is connected to the rest of the robot and manages the energy transmission, while the rotating PCB acquires the sensors data and sends them back to the fixed PCB using infrared communication. For more details about the electronics of the scanner module, see [65].

2.3 Handbot

The handbot (Figure 2.7) is a climbing robot able to manipulate objects, aiming at service in indoor environments. The handbot takes advantage of synergies between climbing and object manipulation. This translates as the ability to climb common vertical office structures such as shelves. Within these structures, the robot gets small objects such as lightweight books or compact discs. The handbot is not mobile on the ground, and relies on other robots for transport, such as a modified version of the marXbot. In this thesis, we use the handbot as a testbed for implementing a complex behaviour with ASEBA: climbing a shelf to retrieve a book (Section 3.3.2, p. 36). Because the handbot operates in three-dimensional space and because we implement the climbing behaviour solely using ASEBA, we consider worth to describe the handbot and to present this experiment. In the rest of this section, we will present and explain the mechatronics of the handbot robot.

2.3.1 Design choices

The biggest constraint for climbing under gravity is to provide the vertical lift force; thus the handbot implements climbing by combining two techniques. Rolling a rope provides the vertical lift force while manipulators provide horizontal operations. The handbot fixes the rope to a ceiling and coils it around a reel. This mechanism is simple and can lift a large mass by using a strong motor, as shown by previous work [58]. To be autonomous, the robot must be able to attach its rope to a specific location, use it to climb and retrieve it afterwards. The handbot's launching mechanism is based on a strong spring (Figure 2.8) that projects a magnet to the ceiling. Though this approach works only in environments

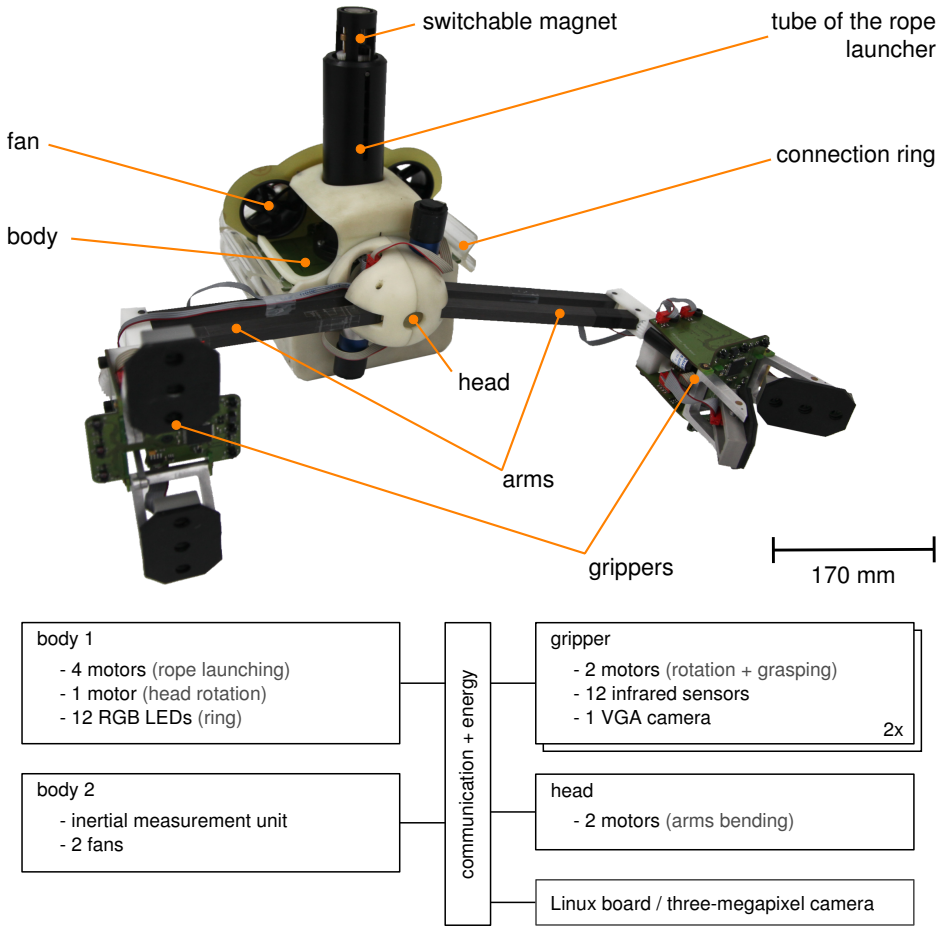


Figure 2.7 The handbot (top) and its electronic schematic (bottom). All motors have position, speed and current sensors.

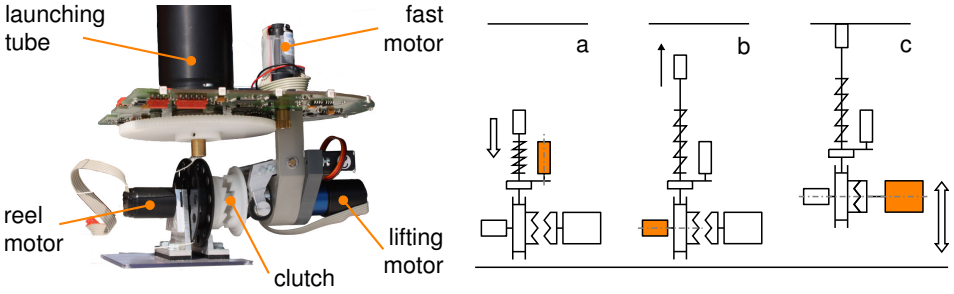


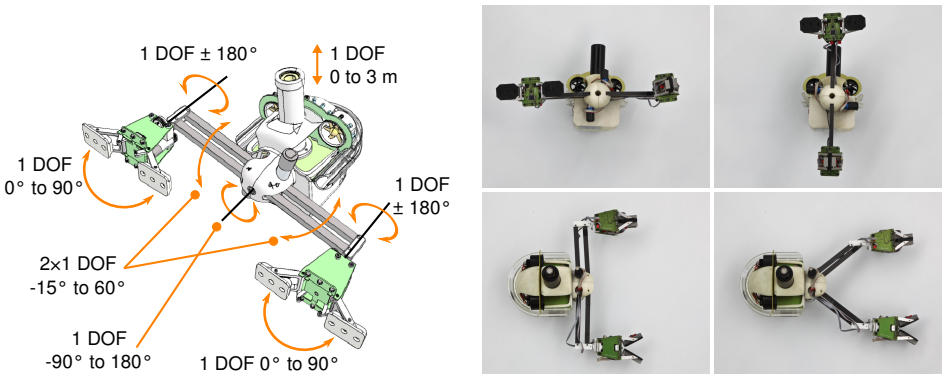
Figure 2.8 Left: mechanism of the rope launcher. Right: schematics of the launching sequence; the filled rectangles show the active motor: **a**, a motor compresses the spring, **b**, a fast motor maintains the tension in the rope while the spring launches the magnet, **c**, a clutch is engaged to let a strong motor lift the robot.

with a ferromagnetic ceiling, it is well understood and reliable. Moreover, depending on the type of ceiling, other attachment mechanisms such as plungers may be applicable. To detach from the ceiling, the attachment must be switchable. The handbot implements this feature using a magnetic switch that the robot can trigger by sending a predefined code through infrared.

When attached to the ceiling using the rope, the handbot has vertical mobility but is horizontally unstable. To stabilise and position itself on the horizontal plane, it needs manipulators to grasp the structure around it. The handbot has two arms, each with a gripper as manipulator (Figure 2.7). When using the two arms to climb, the robot maintains its stability all the time. It can also manipulate an object with one gripper while keeping the other one attached to the structure. That way, the handbot can manipulate objects precisely. Once on the ground, special versions of the marXbot robot can assemble with the handbot and displace it and the object it carries to a specific location. The marXbots attach to the handbot by grasping its translucent ring, which also contains 12 RGB LEDs.

2.3.2 Rope launcher

As we explained in the previous sections, the rope provides the main lifting force of the handbot. Figure 2.8 (left) shows a photo of the mechanism of the rope launcher and Figure 2.8 (right) shows the schematics of the launching sequence. To launch the rope, a motor compresses a spring using a small wagon. This wagon moves inside the launch tube and is driven by a worm gear. When the spring is fully compressed (at



(a) kinematic of the handbot. **(b)** extreme head and arms positions

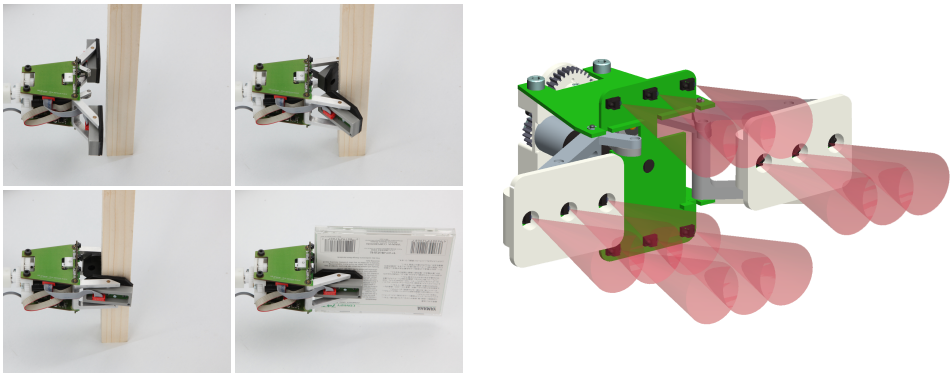
Figure 2.9 The degrees of freedom of the handbot

that point, it applies a force of 110 N), the wagon hits the bottom of the tube and liberates the spring, which launches the rope, the magnet, and the detachment mechanism up to an altitude of 270 centimetres. Two different motors drive the rope: a strong one provides the main lifting force and a fast one controls the tension of the rope, during launch and retrieval. To switch between these two motors, a servomotor activates a clutch. During launch, the fast motor brakes the reel: this is necessary to prevent the formation of knots in the rope. The launching control programme monitors the length of the uncoiled rope in real time. As soon as the magnet reaches the target altitude, the fast motor firmly coils back, which ensures the tension in the rope. If the magnet fails to attach, this action will coil back most of the rope and the handbot will know that the launch has failed. If the launch succeeds, the servomotor engages the clutch so that the strong motor drives the rope, and provides the main lifting force for the handbot. This force is strong enough to lift the robot by itself.

If the handbot hangs freely at the rope, because it holds an object or has failed to grasp an element of a structure, it can stabilise and orientate itself using an inertial measurement unit and two fans. The fans are located at the top of the body on both sides of the rope launcher tube.

2.3.3 Manipulators

The handbot has two arms on its front side. They can bend/extend forward—independently—and rotate with respect to the robot body (Figure 2.9). At the end of each arm, the handbot has a gripper (Figure 2.10a).



(a) passive compliance when grasping **(b)** locations of the infrared sensors

Figure 2.10 The handbot's gripper can open at 90° and close completely. When closing, its parallel compliance mechanism allows it to grasp objects of different thicknesses. The claws can apply a force up to 4 kg thanks to the high reduction of their worm drive.

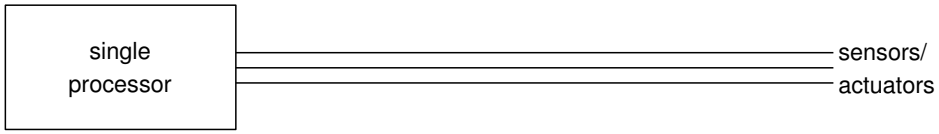
Each gripper can rotate with respect to its arm and can open and close its claws. To grasp objects and structures of different thicknesses, the grippers' claws have a parallel compliance mechanism. When no objects are present, a spring maintains a large opening angle between the claws. Once the claws squeeze an object, the points of contact are different than the points of rotation which generates a moment that aligns the claws in parallel with the object. This provides a strong force over a large range of thicknesses. To control grasping, each gripper can detect structures and objects at close range using 12 infrared proximity sensors on its perimeter (Figure 2.10b). These sensors have a range of 12 centimetres. In addition, the gripper has a VGA camera in its centre. This camera is capable of applying in hardware a Sobel filter [101] to the image, which eases line and object detection.

Chapter 3

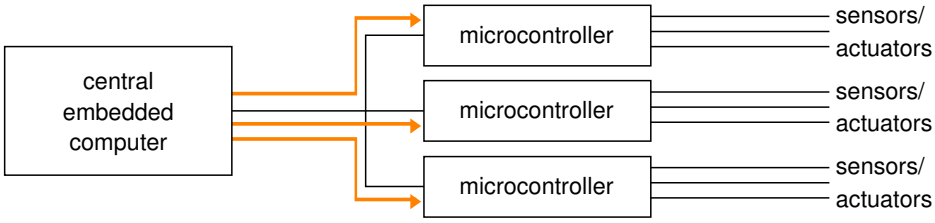
ASEBA, a low level architecture for event-based control of complex robots

As we have described in the previous chapter, a modern miniature mobile robot is all but a monolithic piece of engineering. Such a robot embeds a large quantity of sensors and actuators. These require far more input/output interfaces than what a typical embedded computer offers, and thus one must delegate the management of these peripherals to other processors (Section 2.1.1, p. 10). In most mobile robots that aim at complex tasks [73, 110], microcontrollers manage sensors and actuators while a central computer manages processing-intensive tasks such as vision and reasoning. A shared communication bus connects all these processing nodes together (Figure 3.1b). This architecture solves the connectivity problem at the electronic level; however, it opens new control challenges. Indeed, traditionally the control of miniature mobile robots was built around a simple see/process/act loop (Figure 3.2, polling part). But when peripherals are physically distributed, this control strategy does not scale up with the number and bandwidth of sensors. As the central computer manages all read and write operations, and these transit through the communication bus, the robot suffers from bus overloading and excessive latency to external stimuli. This limits the robot's speed of operation and the number of sensors and actuators the robot can embed.

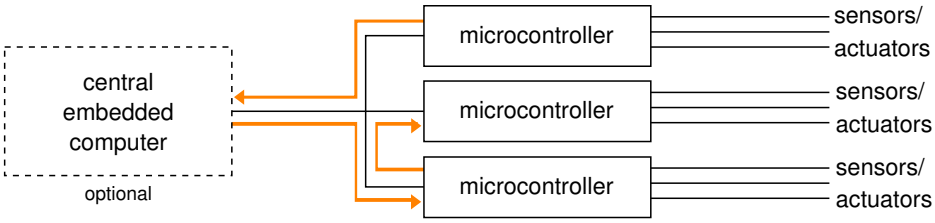
To solve these problems, we must distribute the controller as well, at least partially. In particular, we must filter information in the sensors themselves, and dispatch it to the rest of the robot only if and when the information is relevant to the application. This requires a shift in the



(a) centralised mechatronics, centralised control



(b) distributed mechatronics, centralised control



(c) distributed mechatronics, distributed control

Figure 3.1 Different hardware and control architecture paradigms. ASEBA falls in category c.

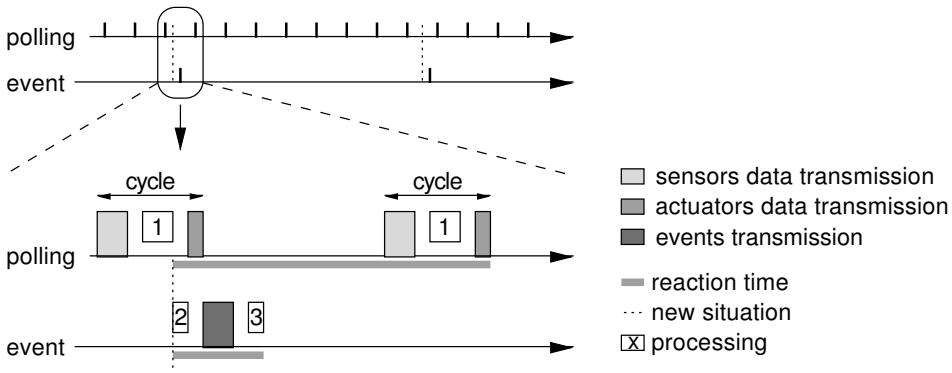


Figure 3.2 A time-oriented comparison of polling versus events-based systems. (1) a central computer processing all sensors, (2) a microcontroller processing its local sensors, (3) a microcontroller processing the incoming event and setting actuators. Because processing is done locally in the microcontrollers, only useful data are transmitted, and the transfer occurs asynchronously. Thus bus load and reaction time are both reduced when using events.

behaviour control paradigm: we must do event-based communication at the microcontroller level instead of polling hardware devices from the central computer. Yet this opens new challenges, as a microcontroller is far less convenient to programme than an embedded computer. Programming needs lengthy re-flashing, debugging requires special wiring, and inspecting the behaviour of the programme in real time often perturbs the operation, as code might run inside interrupts. Moreover, when one wants to debug a distributed system of such microcontrollers, and the synchronisation between them, one is often left with hacking one's own monitor and debug facilities. This is clearly not a satisfactory solution for programming robots.

To overcome these challenges, we have developed an *actuator and sensor event-based architecture* (ASEBA). ASEBA provides a clean and unified interface to the robot hardware by running user code inside a virtual machine (VM) on the microcontrollers and allowing the microcontrollers to communicate through events. To programme a network of microcontrollers from a unified place, ASEBA proposes an integrated development environment (IDE) which features instant code compilation, syntax highlighting, distributed debugging and monitoring. ASEBA improves the state of the art of robot low-level control architectures, and allows us to test the following hypotheses:

Hypothesis 3.1. *A* VM *is fast enough to run in a contemporary microcontroller, and with proper integration with programming languages, can provide better performances than code written by the robot application developer (Section 3.3.1).*

Hypothesis 3.2. *The use of a* VM *and user-friendly* IDE *allows rapid prototyping of complex distributed behaviours (Section 3.3.2).*

Hypothesis 3.3. *A low-level control architecture that takes mechatronic constraints into account should be event-based. This allows bandwidth saves (Section 3.3.3) and lower latencies (Section 3.3.4) compared to a polling-based approach.*

Hypothesis 3.4. *Let a robot with a main computer for high-level tasks and a low-level control architecture for managing its sensors and actuators; Integrating the low-level architecture with an existing middleware on the main computer allows to implement high-level tasks using a variety of programming languages (Section 3.3.5).*

This chapter presents ASEBA and gives experimental results that validate these hypotheses. It also discusses the lessons learnt, the future works

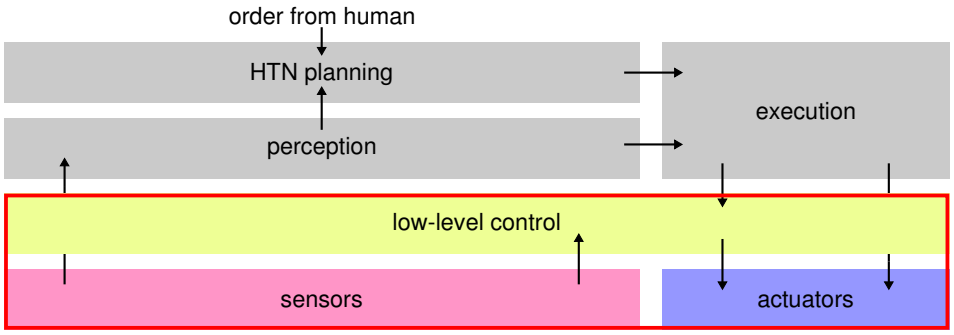


Figure 3.3 The ASEBA part within the bloc scheme of the autonomous construction application.

and situates ASEBA in the broader context of this thesis. As Figure 3.3 shows, in the context of the autonomous construction application, ASEBA allows to implement the low-level control in close interaction with the hardware.

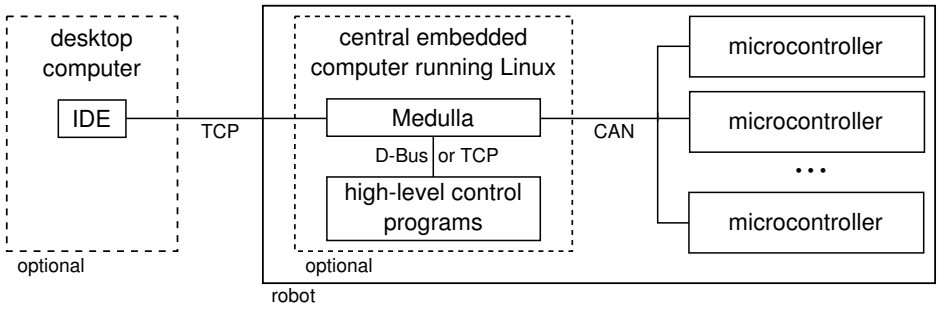
3.1 Related work

The idea of distributing data processing to the sensors themselves was first explored by [28] almost twenty years ago. However, this early 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 [42] or adaptive [44] rate. ASEBA improves on these by providing an event-based architecture where the event emission policy is controlled by a VM inside microcontrollers. Previous work has shown that a VM can be lightweight enough to run even on tiny robots [106].

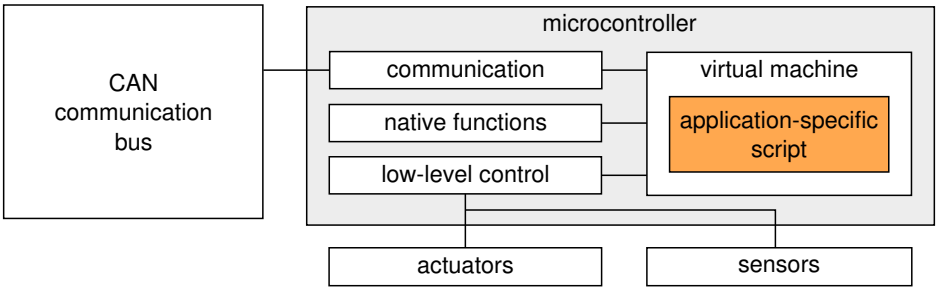
The event-based approach to multi-process communication has been extensively studied in general-purpose middleware literature [66]. Early theoretical works have shown the importance of strongly typing events [64]. More recent work has focused on using this type information to route events efficiently [85]. In the context of robotic applications, researchers have developed many middlewares, 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 [70] uses HTTP; Miro [112] and RtMiddleware [4] use CORBA; Orca uses Ice [52] while Orocos [14], Carmen [74], DCA [83], Player [17] and ROS [88] provide their own layers. Some middlewares provide additional features, such as Orocos, which provides a library to do Bayesian filtering, kinematics and dynamics computation; or Carmen, which provides localisation and mapping. ROS is also more than a middleware, as it is a robot software distribution. YARP [33] is a middleware that proposes several connection types, such as TCP, UDP, HTTP, etc. Orccad [97] is a noteworthy approach that provides an integrated tool to build a behaviour and prove its real-time constraints. Finally, GenoM [34] proposes an architecture close to ASEBA, with a low-level layer consisting of distributed modules, and a centralised decisional layer on top of it. Despite the diversity, these middlewares are all component-based architectures that run on one or more central computers, not on microcontrollers. In the context of miniature robots, these architectures thus suffer from the same bandwidth and latency problems as any polling-based system.

The need for a deeply embedded behaviour control architecture has been recognised by researchers working on complex robots, such as robots with many degrees of freedom like modular self-reconfigurable robots [93]. 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" [120, 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 behaviour 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 re-flashing of all the microcontrollers for any change in the controller code. On the contrary, ASEBA emphases efficient development tools and provides an IDE that allows instant changes in the behaviour of the microcontrollers by loading new code to the VM. 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 [12] directly inside the microcontrollers. In the more demanding context of three-layer architectures [39], we can run the controller layer on the microcontrollers. This distribution of work



(a) a typical ASEBA network in a robot



(b) a microcontroller in an ASEBA network

Figure 3.4 ASEBA in miniature mobile robots.

unloads the central computer and allows it to focus on the sequencer and the deliberative layers.

3.2 Concept and implementation

3.2.1 Architecture

ASEBA is an event-based architecture consisting of a network of processing units which communicate using asynchronous events. An event is a message with an identifier and payload data. All nodes send events and react to incoming events. In a small mobile robot, most of the nodes are microcontrollers and the communication layer can be any bus that is multi-master–capable. In our robots, we use the CAN bus [102] that provides this feature. Asynchronous events allow any microcontroller to transmit data when it wishes to. A basic behaviour 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 (Figure 3.2, Section 3.3.3). Moreover, when compared to polling with a fixed frequency, asynchronous events also decrease the latency, which improves the robot reaction time (Section 3.3.4).

In addition to the microcontrollers, the robot optionally embeds a central computer running Linux. The Linux runs a piece of software, Medulla, which extends the network through D-Bus to local programmes and through TCP/IP to any remote host, typically for the ASEBA IDE, as seen in Figure 3.4a. D-Bus is a middleware present by default under Linux [15]. It provides object-oriented inter-process communication and remote method invocation, using either synchronous or asynchronous calls. Moreover, D-Bus is language-independant and integrates with most programming languages such as C, C++, Java but also with scripting languages such as Shell, Perl and Python. On these last two, D-Bus takes profit of their dynamism to automatically create proxy objects corresponding to remote interfaces. This eases the communication with ASEBA from these languages, as shown in Listings 3.3 and 3.4. Medulla, by providing both a standard interface to Linux programmes and a generic ASEBA interface using TCP/IP, allows to run high-level controller software on the Linux and scripts on the microcontrollers concurrently.

3.2.2 Events and control

The choice of which event to send depends on the robot behaviour: A robot engaged in obstacle avoidance would not need the same events as a robot following walls. Therefore, the behaviour 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 (Figure 3.4b).

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 programmes that control the events emission and reception policy run in a VM on the microcontrollers (Figure 3.4b

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 behaviour.

3.2.3 Language

In aseba, we describe the robot behaviour and the events emission and reception policy in a scripting language. We can associate some code to an event so that the reception of the event triggers the execution of the code. 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 code execution timing.

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 behaviour when a state changes, for instance, when an obstacle is closer than a threshold distance. When the vm loads a code, it considers all *when* conditionals as initially false. 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; specific functions are also available depending on the microcontroller. Section A.1 lists the formal ebnf grammar of the aseba language.

Listing 3.1 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 if the activation exceeds a threshold.

3.2.4 Integrated development environment

The efficiency of the development of a mobile robot behaviour depends on easy inspection of what is happening inside the robot. In particular, we

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

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

onevent FreeOfObstacle
speed = 50
```

Listing 3.1 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.

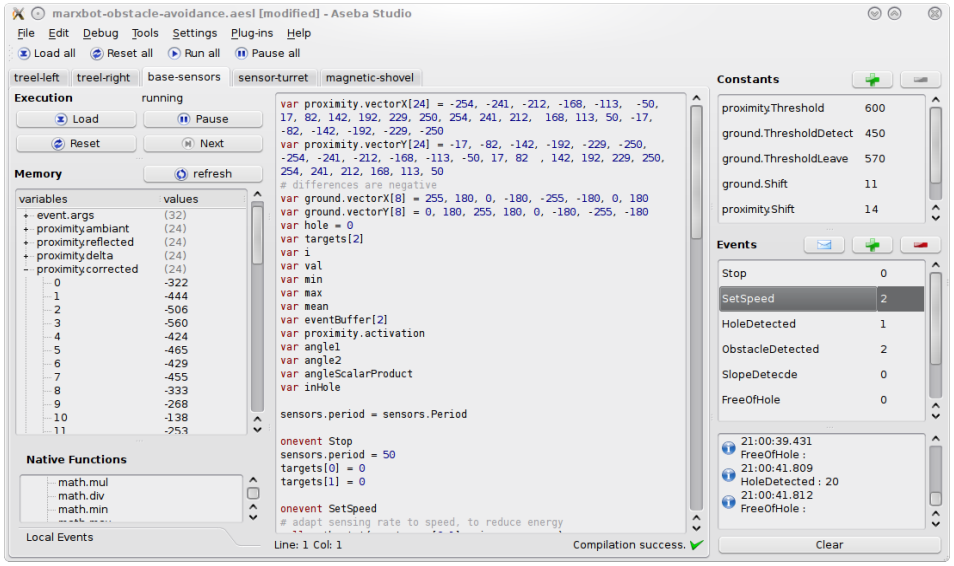


Figure 3.5 Screenshot of the ASEBA IDE.

would like to inspect the values of the sensors, the state of the actuators and the programme 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 fulfils these requirements (Figure 3.5). 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 let the programmer set breakpoints and control the execution state of each node, for instance, to run the script step by step. 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.

3.2.5 Virtual machine

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 this access and will stop the execution of the faulty event. Moreover, the vm provides independence towards 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 aseba vm implements a Harvard architecture and performs computations as a stack machine. Its state thus consists of programme memory (Table 3.1), data memory (Table 3.2), stack memory, programme counter, flags and the list of breakpoints. The vm implements events as a physical processor would implement interrupts. In the bottom of the programme memory, a table called *events vector* maps events' identifiers with addresses. When an event corresponding to an entry arrives, the vm executes the corresponding code until the latter reaches a *stop* bytecode or until the vm has executed too many steps.

The aseba compiler runs in the ide or in Medulla. It consists of a top-down hand-written parser which produces an abstract syntax tree. A type checker verifies this tree, and an optimiser performs simple improvements such as dead-code elimination, constant array access elimination, etc. The resulting simplified tree is transformed into bytecodes corresponding to each event, which are linked together into the final bytecode.

Each bytecode consists of one or more 16-bit words. In the first word, the 4 most significant bits encode the bytecode's type; the rest and the following words encode the bytecode's data. The execution of a bytecode increments the programme counter by its words count, excepted for bytecodes performing flow control which jump otherwhere. Table 3.3 shows all types of bytecodes. The bytecode can be flashed into the microcontrollers, to allow a network of microcontrollers to run autonomously.

The bottom of the data memory contains the exported variables, whose names and meanings are pre-defined per microcontroller. These include the identifier of the microcontroller and the payload data of the last event,

addresses (in 16-bit words)	content
`0x0000`	`evVectSize`
`0x0001`	`ev0Id`
`0x0002`	`ev0Addr`
...	...
`evVectSize` −2	`evLastId`
`evVectSize` −1	`evLastAddr`
`ev0Addr` ...	bytecode for first managed event
...	...
`evLastAddr` ...	bytecode for last managed event
... `bytecodeSize` −1	unused bytecode
`evVectSize` is the size of events vector. `ev0Addr` is the starting address of first managed event. `ev0Id` is the identifier of first managed event. `evLastAddr` is the starting address of last managed event. `evLastId` is the identifier of last managed event. `bytecodeSize` is the total number of bytecode words.	

Table 3.1 The programme memory layout of an Aseba VM.

addresses (in 16-bit words)	content
`0x0000` ...	exported variables
`exportedVarsLength` ...	user-defined variables
...	unused variables
... `variablesSize` −1	temporary variables to pass constants to native calls
`exportedVarsLength` is the length of the exported variables. `variablesSize` is the total number of variable words.	

Table 3.2 The data memory layout of an Aseba VM.

name	w.c.	function
stop	1	stop execution
small immediate	1	push a constant ($\leq$ 12 bits) onto the stack
large immediate	2	push a constant ($>$ 12 bits) onto the stack
load	1	push data from memory onto the stack
store	1	pop data from the stack into the memory
load indirect	2	push data from memory onto the stack using an offset from the stack
store indirect	2	pop data from the stack into the memory using an offset from the stack
unary arithmetic	1	unary arithmetic operation on the stack
binary arithmetic	1	binary arithmetic operation on the stack
jump	1	jump to another execution address
conditional branch	2	check a condition on the stack and jump depending on the result
emit	3	send an event
native call	1	call a native function
sub call	1	jump into a subroutine, store return address on the stack
sub ret	1	return from a subroutine, using return address from the stack

Table 3.3 The types of ASEBA bytecodes. The w.c. column indicates the number of 16-bit words that a bytecode of this type counts.

```
interface ch.epfl.mobots.EventFilter
{
  method Void ListenEvent(UInt16 eventId)
  method Void ListenEventName(String eventName)
  method Void IgnoreEvent(UInt16 eventId)
  method Void IgnoreEventName(String eventName)
  signal Event(UInt16 id, String name, Array<SInt16> payloadData)
}

interface ch.epfl.mobots.AsebaNetwork
{
  method Void LoadScripts(String fileName)
  method Array<String> GetNodesList()
  method Array<String> GetVariablesList(String nodeName)
  method Void SetVariable(String nodeName, String variableName,
    Array<SInt16> variableData)
  method Array<SInt16> GetVariable(String nodeName, String variableName)
  method Void SendEvent(UInt16 eventId, Array<SInt16> payloadData)
  method Void SendEventName(String eventName, Array<SInt16> payloadData)
  method ObjectPath CreateEventFilter()
}
```

Listing 3.2 The D-Bus interface of the aseba network, as exported by Medulla.

but also all the variables exported by the sensors and the actuators.

The processing overhead of the vm with respect to native code is acceptable in modern microcontrollers (Section 3.3.1). Moreover, it is a lightweight software. In a typical dsPIC33 implementation, it consumes 10 kB of flash memory and 4 kB of ram, including all communication buffers. We can adapt these requirements by adjusting the amount of bytecode and variable data, stack size and number of breakpoints.

Finally, the vm is easy to understand and thus easy to adapt and optimise. Its source code counts fewer than 1000 lines of C, including the debugger core. Section A.2 explains the procedure to deploy aseba on a new platform. In addition, the aseba source code, available at `http://mobots.epfl.ch/aseba.html`, provides a skeleton as well as examples.

3.2.6 Medulla, the D-Bus integration

Medulla presents the aseba network on D-Bus through a singleton object of interface `ch.epfl.mobots.AsebaNetwork` (Listing 3.2). Through this interface, a programme can retrieve information about the network, read and write variables, load scripts into the microcontrollers or send events. A programme can also receive events through medulla, but for the

sake of efficiency this requires a bit more machinery. Indeed, in a multi-application context, each programme is only interested in some events. To avoid waking-up every programme each time an event is received, Medulla implements event filtering. Each application that wants to receive events must call `CreatEventFilter()` to create an event filter. The latter exports the interface `ch.epfl.mobots.EventFilter`, which allows the application to choose which events it wants to receive. The application will then receive only those events through the `Event` signal.

3.2.7 Helper programmes

To ease experimentation, ASEBA provides several helper programmes. They connect to Medulla throught TCP/IP. A dump programme prints the type and the content of every message. A record programme stores every message along a timestamp, and a replay programme can stream again these messages, at real time or faster. A command line programme allows the user or a script to send a message.

3.2.8 Availability

ASEBA is written in C (microcontrollers) and C++/Qt (IDE). It is open-source (GPL v.3) and fully cross-platform. More information, as well as the latest version, are available at `http://mobots.epfl.ch/aseba.html`.

3.3 Experiments

This section presents the experiments that test Hypotheses 3.1 to 3.4. For each experiment, we present it, show the results and discuss the corresponding hypothesis in the light of the results.

3.3.1 Virtual machine

As we will see in this section, the VM of ASEBA achieves respectable performance, even if we have not yet micro-optimised 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 100 times, the delay between an incoming event and the resulting outgoing one is 1.71 ms. Such a loop

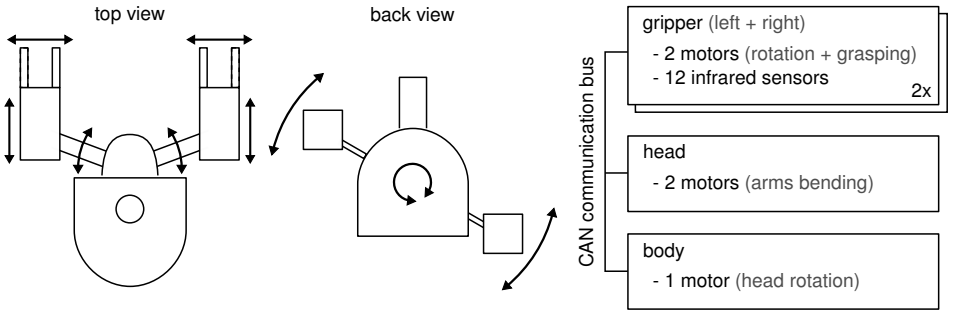


Figure 3.6 The degrees of freedom of the handbot and the distribution of functions between its microcontrollers for climbing a shelf.

executes around 1000 VM instructions, which results in a VM performance of about 600,000 instructions per second. The dsPIC microcontroller that we use runs at an instruction rate of 40 MIPS. This implies that the VM executes 70 dsPIC instructions per VM instruction.

The ASEBA 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. It is worth noting that the use of the ASEBA VM might even increase the performance of an application when compared to a C code from a lambda programmer. Indeed, the VM provides native functions to compute common mathematical operations such as the dot product. As modern microcontrollers contain optimised 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 optimisation runs in 871 cycles.[1] The corresponding ASEBA native function, which uses the multiply-accumulate instruction, runs in 320 cycles. The dot product is probably the most used mathematical operation for robot sensor processing, as it is required for vector and matrix operations, discrete Fourier and Z transforms, digital filters, etc. Our measures on the dot product, knowing its widespread use, prove Hypothesis 3.1 (p. 23).

3.3.2 Application to complex mechatronics

This section presents the application of ASEBA to a complex mechatronic system, the handbot climbing robot (see Section 2.3, p. 16). We show how ASEBA provides distributed control for a tightly integrated behaviour

1. C30 3.11, based on gcc 4.0.3, -O3 optimisation flag

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 handbot uses four microcontrollers to climb a shelf. To move, the handbot rotates its arms with respect to its body in alternate directions (Figure 3.6). Every 180 degrees, the free gripper attaches to the vertical board, and the other grip is released. The handbot repeats this sequence until the robot reaches the height of the book, where the free gripper takes the book. Then the handbot detaches its second gripper and goes back down the rope (Figure 3.7). 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 (Figure 3.8).

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. Through its step-by-step mode, the integrated debugger allowed us to easily debug the finite state machines controlling 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 behaviour. As a result, the handbot is able to climb without a central computer. This validates Hypothesis 3.2 (p. 23).

3.3.3 Bandwidth savings

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 Figure 3.9 (left). We have listed the full source code of the robot controller in Section B.1 (p. 139). We compare our experimental data to the requirements of a polling-based approach, which are the following:

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

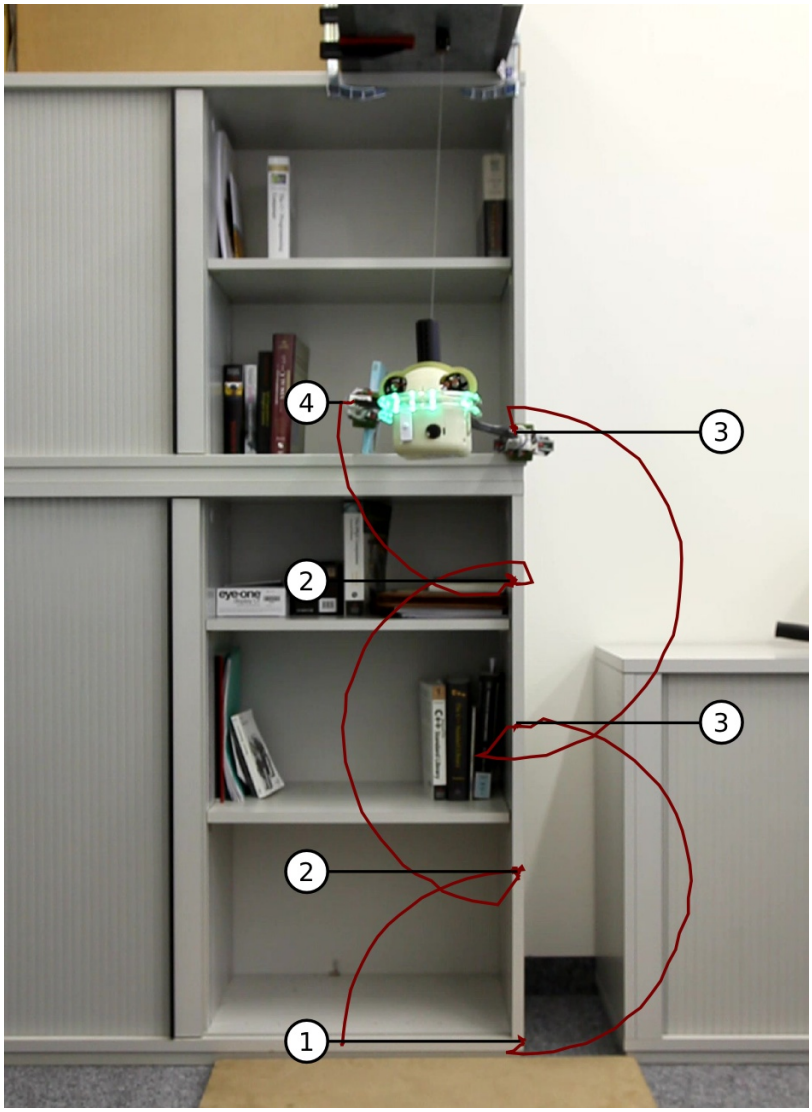


Figure 3.7 High-level events of a handbot climbing a shelf to retrieve a book. The burgundy lines show the traces of the grippers. The sequence of events is as follows: (1) the right gripper attaches, the head rotates clockwise; then alternatively (2) the left gripper attaches, the right gripper detaches, 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.

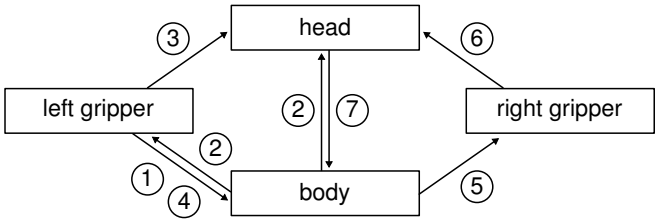
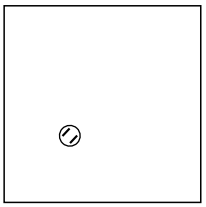


Figure 3.8 Low-level events of a handbot attaching its left gripper and detaching its right one. The sequence of events is as follows: (1) the left gripper detects a vertical board; (2) the body requests the head to extend the left arm and informs the arm; (3) when the left gripper is close enough to 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 body; (5) the body 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.

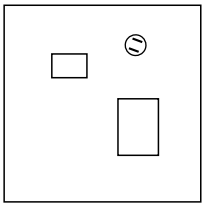
where b is the required bandwidth, f is the frequency of polling (10, 25 and 67 Hz), C is the number of sensors (24), S is the number of bytes to transmit per value (2) and O is the packet header overhead (source identifier + message type identifier, 3).

Figure 3.9 (bottom right) illustrates visually the variance of the bandwidth consumption with respect to the location of the robot. The latter is proportional to the number of obstacles encountered by the robot. The measures in Figure 3.9 (top right) 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 maximise the bandwidth savings. These results contribute to validate Hypothesis 3.3 (p. 23).

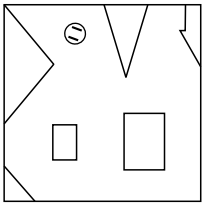
While the savings are significant for the sensors of the marXbot base, we expect them to be even higher in the case of high-bandwidth sensors such as sound source detection, vision-based features detection, or laser scanners. For instance, the dsPIC microcontroller that we use is powerful enough to process sound in real time [71], and thus through the use of native functions, ASEBA could process sound from several microphones and report the direction of the incoming sound. Moreover, if we imagine a robot with many sensors, it is probable that this robot would not use



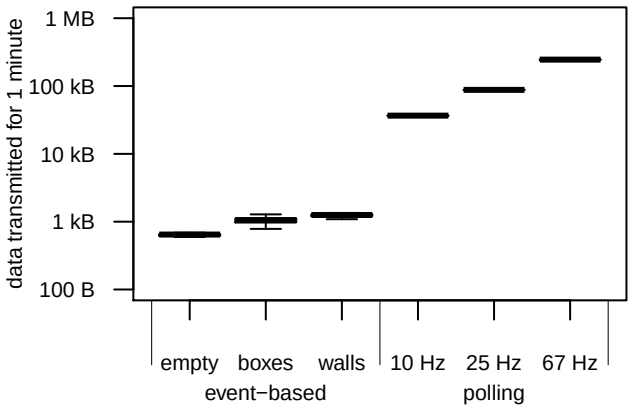
(a) empty environment



(b) boxes environment



(c) walls environment



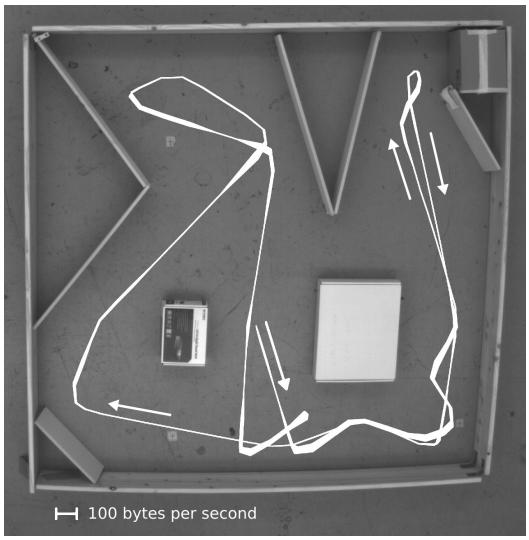


Figure 3.9 Measurement of bandwidth consumption. We performed this experiment in three different environments. Top: The box plot shows the bus load in these along with the theoretical values for polling. Bottom: Bandwidth consumption of events with respect to the location of the robot in the wall environment. The width of the white line is proportional to the bandwidth consumption.

all of its sensors at the same time. With a polling-based approach, we could choose not to poll some sensors. However, these sensors might still sometimes hold information useful to the application, for instance in abnormal situations. Using aseba, we could programme some sensors to send data only in case of emergency condition, to save bandwidth for the sensors currently being used. That way, all the perceptions are still taken into account, but with a reduced bandwidth consumption. This allows to scale the robot's capabilities as aseba permits to integrate a lot sensors and let them work together, sharing the bus depending on their needs of the moment.

3.3.4 Low latency

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 robot detects 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 Figure 3.10 show that the event-based control of aseba allows the robot to stop precisely at a long distance away from 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 transmission of the read command and to its aliasing with the update of the sensors' values. Because of Shannon's theorem, a polling rate of 134 Hz or beyond should remove these effects. 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 optimise its speed while operating safely. These results and the one from Section 3.3.3 validate Hypothesis 3.3 (p. 23).

3.3.5 Cross-language integration

This section presents a remote control application for the marXbot robot. This application allows a human to control the movement of the robot using a joystick. If we limit the magnitude of the speed command, the control is safe, that is, the robot will not crash into a wall

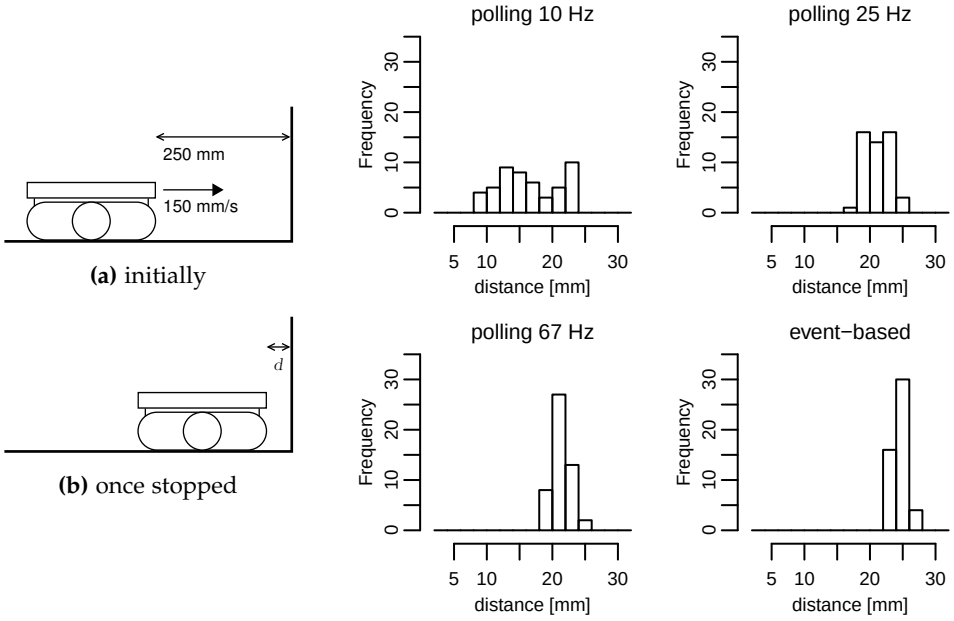


Figure 3.10 Measurement of latency. The robot moves towards a wall at the speed of 150 mm/s starting from a distance of 250 mm (a). When the robot's front sensors detect the wall at a distance closer than 30 mm, the robot stops. We then measure the distance d to the wall (b). The histograms show the distribution of d.

```
#!/usr/bin/perl
use Net::DBus;
use Net::DBus::Reactor;

# gets stub of ASEBA network
my $bus = Net::DBus->session;
my $asebaService = $bus->get_service('ch.epfl.mobots.Aseba');
my $asebaNetwork = $asebaService->get_object(
  '/', 'ch.epfl.mobots.AsebaNetwork');

# loads scripts
$asebaNetwork->LoadScripts($ARGV[0]);

# creates an event filter and listen for an event
my $eventFilterPath = $asebaNetwork->CreateEventFilter();
my $eventFilter = Net::DBus::RemoteObject->new(
  $asebaService, $eventFilterPath, 'ch.epfl.mobots.EventFilter');
$eventFilter->ListenEventName('SetSpeed');
$eventFilter->connect_to_signal('Event',
  sub {
    # print the event
    my ($id, $name, $payloadData) = @_;
    print 'Event ' . $id . '/'. $name . ': ';
    for my $value (@$payloadData) {
      print "$value ";
    }
    print "\n";
  }
);

# starts event loop
my $reactor = Net::DBus::Reactor->main();
$reactor->run();
exit(0);
```

Listing 3.3 A Perl programme to load an ASEBA script and to log an event. This programme uses the `libnet-dbus-perl` library.

```
#!/usr/bin/python
import dbus
import dbus.mainloop.glib
import glib
import gobject
import pygame

def dbusReply():
  pass

def dbusError(e):
  print 'dbus error:'
  print str(e)
  loop.quit()

def scanJoystick():
  global ox, oy
  pygame.event.pump()
  x = joystick.get_axis(0) * 60
  y = -joystick.get_axis(1) * 60
  if x != ox or y != oy:
    asebaNetwork.SendEventName('SetSpeed',
      [y+x, y-x], reply_handler=dbusReply, error_handler=dbusError)
    ox = x
    oy = y
  # reschedule scan of joystick
  glib.timeout_add(20, scanJoystick)

if __name__ == '__main__':
  # inits main loop and joystick
  dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
  pygame.init()
  joystick = pygame.joystick.Joystick(0)
  joystick.init()
  ox = 0
  oy = 0

  # gets stub of ASEBA asebaNetwork
  bus = dbus.SessionBus()
  asebaNetworkObject = bus.get_object('ch.epfl.mobots.Aseba', '/')
  asebaNetwork = dbus.Interface(
    asebaNetworkObject, dbus_interface='ch.epfl.mobots.AsebaNetwork')

  # schedules scan of joystick
  glib.timeout_add(20, scanJoystick)

  # starts event loop
  loop = gobject.MainLoop()
  loop.run()
```

Listing 3.4 A Python programme to send speed commands. This programme uses the `python-gobject`, `python-dbus` and `python-pygame` libraries.

regardless of the command from the human. This applications uses three events: `SetSpeed` allows the human to control the robot's movements, while `ObstacleDetected` and `FreeOfObstacle` allow the robot to avoid obstacles regardless of the control command. These last two events allow to transmit data from the proximity sensors to the motors only when there is an obstacle. This lowers the bus bandwidth and reduces the processing load on the motor microcontroller, compared to a polling-based approach.

This application illustrates the vertical integration that aseba allows. At the low level, distributed among three microcontrollers, three aseba scripts implement the obstacle avoidance and its fusion with the control command (Listing 3.1). This script uses the ring of proximity sensors around the robot to detect obstacles in any direction. At the Linux level, a Perl programme loads this script, and then dumps all the `SetSpeed` events using an event filter (Listing 3.3). A Python programme sends `SetSpeed` events to the microcontrollers given a joystick command (Listing 3.4). This application shows that thanks to its vm and Medulla, aseba allows a straightforward scripting of the robot's behaviour at all levels. This proves Hypothesis 3.4 (p. 23).

3.4 Discussion

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 delegating to the microcontrollers the task of filtering raw data. 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. Using a robust communication bus, an event-based approach could also enable the creation of a fail-safe behaviour by using redundant hardware modules, as [28] showed.

Our results show that a safe, event-based *scripting* language is a sound approach to the programming of real-time embedded systems. This question was raised in the computer science literature [63], and aseba provides an affirmative answer to it. We could further increase the performances of aseba. Indeed, we could improve the compiler so that it proves more facts about the programme. For instance, the optimiser 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 [54]. 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. While this allows the embedding of the VM into most of the existing microcontrollers, it limits the types of data that Linux programmes can exchange through ASEBA. We could extend the language by adding more basic types (floating point values, 32-bit integers, etc.) and arbitrary data structures. If ASEBA would implement the same set of data structures as D-Bus does, we could further hide the difference between microcontrollers and Linux programmes. This would correspond to strongly typing events as in [64]. However, we must keep in mind that much of the ease of programming in ASEBA comes from the static memory allocation of data inside the VM. The compiler knows the address of each variable globally, which allows the compiler to perform useful checks at compile time. Some richer data structures, for instance those requiring dynamic memory allocation, would reduce the user-friendliness of the environment. We could alleviate this drawback by adding run-time checks and by improving the reasoning done by the compiler. While the former would reduce the execution speed and increase the bytecode size, the second is promising but requires state-of-the-art techniques from research in compilers. Moreover, it might not be desirable to lure the user into thinking that she is working with workstation-level processors, where floating-point operations are cheap and the memory is virtually unlimited.

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. To be suitable for industrial applications driving large–and potentially dangerous—pieces of equipment, ASEBA should provide execution timing guarantees. It is possible to improve the compiler such that it proves timing constraints for a subset of possible programmes, and notifies the developer when it cannot [53]. However this is difficult to do in general, also because in the analysis of the compiler should consider the timing constraints/guarantees of the communication bus. Addressing these challenges would be a rich research direction based on the foundation laid down by ASEBA.

3.5 Conclusion

The experimental performances of aseba running in physical robots demonstrate that a modular, distributed and event-based architecture is a pertinent solution to programme the behaviour 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 enhances the *scalability* by filtering raw data and by implementing reflex-like control locally. This provides better behavioural 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 tasks such as path-planning or reasoning. Thanks to the easy to use scripting language and the ide, aseba brings these advantages without compromising the flexibility nor diminishing the efficiency of the development process. At the level of the central embedded computer, aseba seamlessly interacts with other programmes through its integration with D-Bus, the Linux standard messaging middleware. One can thus script the high-level behaviour using languages such as Python or Perl. Therefore, aseba allows vertical integration between the various software layers of a multi-microcontrollers robot and brings the flexibility and reusability of middleware deep into the robot's mechatronic.

We have validated all our hypotheses by experiments with physical robots. As a contribution to the fundamental hypothesis of this thesis, Hypothesis 1.1 (p. 2), aseba shows that building a software control architecture knowing the electronic constraints of the robot greatly improves the capabilities of this robot.

Chapter 4

Global optimisation for SLAM

Most research works on SLAM concentrate on the mathematical formulation of the SLAM problem and do not address the integration questions. Typically, relatively large robots ($> 50\,\text{cm}$) embed expensive industrial laser scanners [95]. Recently, some works have explored the use of vision as an alternative to laser scanners [67, 96]. Cameras are compact and passive sensors, and so are well-suited for integration into small mobile robots. However, visual SLAM requires an environment populated with well-illuminated features and a large processing power [86]. Some applications do not enjoy such excellent conditions, and must fall back on distance measurements to perform SLAM. For instance search and rescue robots often operate in dark environments with no clear visual features. We therefore believe that there exists a strong need for non-visual SLAM techniques that one can integrate into cheap, small and low-power robots.

To integrate SLAM within a compact robot with modest sensor quality, it is important that the parameters of the robot's motion model are correct. For instance, if the values of the error model are not correct, the SLAM algorithm might diverge. Moreover, as the computational power is also limited, we must allocate at best the processing resources to the different steps of the SLAM algorithm. However, the relation between the different parameters is complex and non linear. To find these parameters, we propose to use an evolution strategy [8], which is a robust global optimisation method. Our hypothesis is that this optimisation exploits at best the capabilities of the robot, and thus allows SLAM on small, inexpensive and low-power robots.

Hypothesis 4.1. *A global optimisation of the robot's intrinsic parameters and of the allocation of processing resources allows real-time* SLAM *on small, inexpensive and low-power robots.*

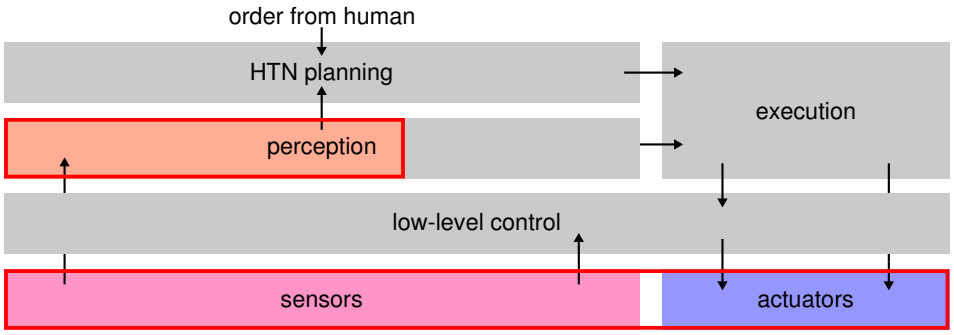


Figure 4.1 The SLAM part within the bloc scheme of the autonomous construction application.

This chapter presents our SLAM implementation, details the global optimisation methodology and gives experimental results that validate this hypothesis. It also shows the contribution of the SLAM results to the main hypothesis of the thesis. Figure 4.1 shows the SLAM algorithm in the context of the autonomous construction application. By providing absolute positioning, the SLAM algorithm forms the basis of the perception subsystem.

4.1 Related work

The reliance of most SLAM implementations on bulky and expensive laser scanners hinders their diffusion into small and cost-effective robots. Several projects have therefore explored the use of cheap and low-resolution distance sensors for the SLAM.

Schroter et al. [92] used the array of sonar sensors which equips the SCITOS A5 robot. Their work focused on reducing the memory footprint of particle-based gridmap SLAM by sharing the map between several particles. The resulting implementation runs in real time on laptop-level computers.

Yap et al. [118] also used sonar sensors. They worked with the ActivMedia P3-DX robot, which has less sensors than SCITOS A5. To cope with this sparseness, their SLAM implementation uses a map of line segments instead of a gridmap. Together with a strong assumption that the walls are orthogonal, their solution was able to reconstruct large indoor environments. Their article does not report any performance measurement. In the same direction, Abrate et al. [1] used line extraction to apply SLAM to a Khepera II robot, which only embeds 8 short-range infrared proxim-

ity sensors. Like in the work of Yap et al., the environment consists of orthogonal walls, in this particular case only in small numbers.

These projects are representative of a line of research which focuses on developing SLAM algorithms that fit the features of specific sensors. They all succeed in performing SLAM in loopy environments thanks to robust algorithms. However, these are too computationally intensive to run in an embedded computer, requiring at least laptop-level performances. Gifford et al. [41] have proposed a global approach to address these limitations. They have both designed a robot and implemented a distributed SLAM algorithm which uses a scanning sensor. Their SLAM algorithm uses a particle filter, and they report real-time performances using 15 particles and 3 seconds per update. The authors conclude that their scanner, a simple set of infrared distance sensors mounted on top of a servomotor, does not provide enough information in sparse environments. They also underline the difficulty in finding the right SLAM parameters to fit within the available computational power. Recently, Grzonka et al. [45] performed SLAM experiments on an autonomous indoor flying robot. Albeit they use a laser scanner, their SLAM implementation runs in real time on the computer of their small flying robot.

Our contribution is to use a global optimisation algorithm to find the intrinsic parameters of the robot's motion model and to allocate processing resources. We have implemented FastSLAM 2.0 using the slim and inexpensive rotating distance scanner of the marXbot (see Section 2.2.3, p. 14), thus fitting the size and cost requirement. The resulting algorithm runs faster than real time on the marXbot robot.

4.2 Model and implementation

We have adapted FastSLAM 2.0 [75] to the specificities of our hardware (see Section 2.2.3). This SLAM implementation estimates the pose of the robot and incrementally builds a 2D occupancy-grid map [29] of its surrounding environment. A time step corresponds to a full 360° scan by the rotating scanner (half a turn). Each cell of the occupancy-grid map holds the log odds ratio of the belief that this cell is an obstacle [108, p. 94, p. 286]. Our SLAM algorithm consists of a particle filter, where each particle k at each time step t contains the robot pose $\boldsymbol{x}_t = (x, y, \theta)^T$, the associated weight w_t and a full map of the environment m_t. The algorithm updates these three values with the new measurements acquired by the scanner in four phases. These phases are:

A. the pose update (Section 4.2.1),
B. the measurement to map matching (Section 4.2.2),
C. the occupancy-grid update (Section 4.2.3),
D. the particles resampling (Section 4.2.4).

4.2.1 A. Pose update

The rotating scanner only produces enough data for a relevant map matching every half turn. Moreover, we must perform the estimation of the robot pose at the same rate as the measurement to map matching. Yet during a half turn of the scanner, the robot receives several odometry measurements. To cope with this discrepancy, we store all the measurements and delay the computation of the robot pose. To compute the pose, we reconstruct the trajectory by iteratively applying the odometry measurements.

The update of the pose $\boldsymbol{x}_t^{[k]}$ knowing the pose at the previous time step $t-1$ and the command $\boldsymbol{u}_t$ (odometry measurements) that steered the robot between the two time steps is described by the motion posterior:

$$p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}^{[k]}, \boldsymbol{u}_t) \tag{4.1}$$

The marXbot is roughly equivalent to a differential wheeled robot at the level of the motion model. We approximate this motion model by an odometry model in which we decompose the interval $(t-1, t]$ into an initial rotation δ_{rot1}, a translation δ_{trans}, and a final rotation δ_{rot2}. We can directly get δ_{trans} from the motors' encoders as the average of displacement of each treel. However, the tracks introduce non-linear slipping depending on the speed, the acceleration and the type of surface. The slipping particularly affects the odometry when the robot rotates. We therefore use the gyroscope integrated in the base of the marXbot to measure the changes in orientation.

4.2.2 B. Measurement to map matching

We compute the weight of a particle $w_t^{[k]}$ which is proportional to the likelihood of the measurement $\boldsymbol{z}_t$:

$$w_t^{[k]} \approx p(\boldsymbol{z}_t|\boldsymbol{x}_t^{[k]}, m^{[k]}) \tag{4.2}$$

To compute the likelihood of each measurement, we project 4 rays oriented like the 4 sensors of the scanner at the time of the measurement

onto the particle's internal map and compare the distance measured by the sensor and the one found by reading the map. We back-propagate all the measurements along the trajectory computed at phase A such that matching is done with the estimated robot pose at the time of the measurement. The final likelihood is the product of the likelihood of each measurement along the trajectory of the robot in $(t-1, t]$. Since the response functions of the sharp sensors are not injective (Figure 2.6, p. 15), we ignore their values when they correspond to invalid distances. The probabilistic nature of the map is sufficient to disambiguate wrong readings from correct ones. We manage to cover the whole $(0, 1]$ m range by dropping the values of short range sensors over 35 cm and the values of long range sensors below 35 cm.

We optimise the robot's pose knowing the measurement by performing a scan-matching step using a small Monte-Carlo localisation. For each particle, we explore a small space around the final pose computed in phase A, following a Gaussian distribution. We perform the measurement to map matching for each candidate pose and keep the best match. This operation improves the positioning, and is comparable to having more particles, yet without the memory expense of one distinct map per particle. However, we must project more rays per particle.

4.2.3 C. Occupancy-grid update

We compute the map m which is represented by the posterior:

$$p(m|\boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t}) \tag{4.3}$$

We represent m by the set of all grid cells $m = \{m_i\}$, where m_i is a binary random variable with $p(m_i = 1)$ representing the probability that an obstacle occupies a cell. This independence assumption allows us to approximate the posterior of the map:

$$p(m|\boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t}) = \prod_i p(m_i|\boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t}) \tag{4.4}$$

Note that for simplicity and for compatibility with the literature we use a single index i but remember that the map is two-dimensional. Each cell holds the log odds ratio of the probability that it contains an element [108, p. 94, p. 286], with a resolution of 16 bits:

$$l_{t,i} = l(m_i|\boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t}) = \log \frac{p(m_i|\boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t})}{1 - p(m_i|\boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t})} \tag{4.5}$$

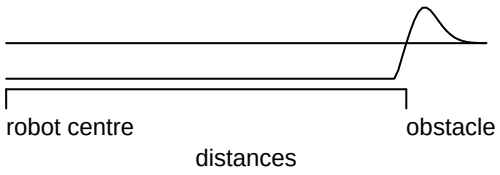


Figure 4.2 The update function, whose values are added to the occupancy-grid map.

This implementation choice is convenient because computing the posterior given a measurement corresponds to summing the log odds ratios:

$$l(m_i|\boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t}) = l(m_i|\boldsymbol{z}_t) + l(m_i|\boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t-1}) - l(m_0) \tag{4.6}$$

where $l(m_0)$ is the log odds ratio of the prior probability that an obstacle occupies the cell, $l(m_i|\boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t-1})$ the previous value of the cell, and $l(m_i|\boldsymbol{z}_t)$ the probability that an obstacle occupies the cell given the most recent measurement. Note that the trajectory $\boldsymbol{x}_{1:t}$ is known within a particle and that $l(m_0) = 0$ because we choose 0.5 for the prior probability of a map cell to be an obstacle.

For the sake of efficiency, we update the map for each sensor measurement using a pre-computed update function dependent on the sensor value (Figure 4.2). We have pre-computed tables for all sensors values for every sensor (512 kB of data). These tables store $l(m_i|\boldsymbol{z}_t)$. Like in phase B, we cast a ray from the estimated robot pose into the direction of the measure. On this ray, the update function adds information that cells before the measured distance are free of obstacles and that cells at the measured distance are occupied by an obstacle. It adds no information to cells beyond the measured distance. We back-propagate the estimated poses of the measures along the robot trajectory in $(t-1, t]$.

4.2.4 D. Particles resampling

The particles resampling frequency is a parameter of our algorithm. When it resamples particles, the algorithm first sorts them according to their weight. It then draws a new set of particles out of the previous set, with a probability proportional to the weight of the particle. The new set may contain many times the same particle, as particles with a large weight have a strong probability to be drawn more then once. However, as the pose update step introduces randomness, the particles will quickly differentiate.

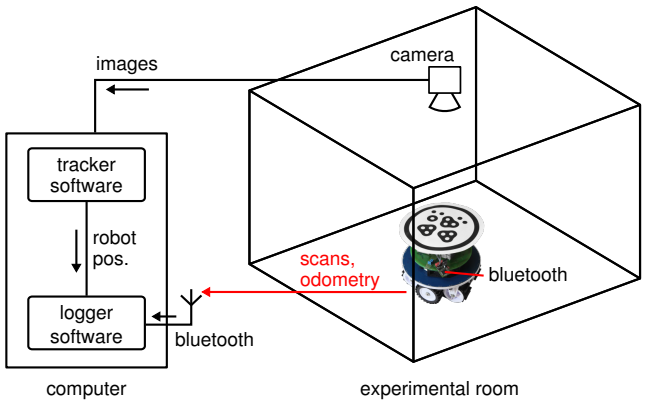


Figure 4.3 The experimental setup for SLAM experiments

4.3 Experimental methodology

4.3.1 Measuring the quality of slam

We run experiments in a room with an overhead camera connected to a robot tracker (Figure 4.3). We built the tracker using `libfidtrack` from the reacTIVision project [1] [7]. The tracker and the robot share a common ASEBA network so we simply use the ASEBA logging facility to record the estimated robot poses along their ground truth. The tracker also measures the experiment time and records images of the arena for further analysis. The tracking camera has a resolution of 3000×2208 pixels. This camera covers an area of approximately 4×3 meters. To calibrate the tracker, we have placed the robot at 12 different known positions (approximately the centres of each squared meter). Then, we have optimised the parameters of the camera projection matrix using a global optimisation algorithm. The average error for the known positions, at the end of the optimisation, is 4 mm. The maximum error is 25 mm.

We measure the quality of the SLAM by comparing the average squared difference between the reconstructed trajectory of the best particle and the real trajectory ($\boldsymbol{T}_{\text{slam}} = \{\boldsymbol{T}^i_{\text{slam}}\}$ and $\boldsymbol{T}_{\text{real}} = \{\boldsymbol{T}^i_{\text{real}}\}$, for i iterating over $S_{\text{T}} = |\boldsymbol{T}_{\text{slam}}| = |\boldsymbol{T}_{\text{real}}|$ tracked positions). However, as both trajectories are expressed in different coordinates, we must first find the set of parameters $\boldsymbol{\theta}_T = \{\theta_\alpha, \boldsymbol{\theta}_d\}$ for the transformation $A(\boldsymbol{T}^i_{\text{slam}}, \boldsymbol{\theta}_T) = \boldsymbol{R}(\theta_\alpha)\boldsymbol{T}^i_{\text{slam}} + \boldsymbol{\theta}_d$ (knowing that $\boldsymbol{R}(x)$ is 2D rotation matrix of angle x) to minimise the

1. `http://reactivision.sourceforge.net/`

distance:

$$\mathrm{d}(\boldsymbol{T}_{\mathrm{slam}}, \boldsymbol{T}_{\mathrm{real}}, \boldsymbol{\theta}_T) = \sum_i^{S_\mathrm{T}} (\|A(\boldsymbol{T}^i_{\mathrm{slam}}, \boldsymbol{\theta}_T) - \boldsymbol{T}^i_{\mathrm{real}}\|^2) \tag{4.7}$$

We find the optimal set $\hat{\boldsymbol{\theta}}_T$:

$$\hat{\boldsymbol{\theta}}_T = \underset{\boldsymbol{\theta}_T}{\operatorname{argmin}}(\mathrm{d}(\boldsymbol{T}_{\mathrm{slam}}, \boldsymbol{T}_{\mathrm{real}}, \boldsymbol{\theta}_T)) \tag{4.8}$$

We implement the argmin optimisation using a simple evolution strategy [8] using 32 individuals over 128 generations. This evolution strategy operates on the real-valued space of $\boldsymbol{\theta}_T$. We do not perform self-adaptation, and after each generation, we keep the best 25 % individuals. For each of these individuals, we make three mutated copies to replace the discarded 75 % individuals. During evolution, we perform simulated annealing by varying the mutation factor from 100 % to 20 %. The initial values are 0 for all elements of $\boldsymbol{\theta}_T$. For mutations, the standard deviation of the normally-distributed added value is 50 for all elements of $\boldsymbol{\theta}_T$. The quality of the trajectory is the inverse of the mean of the residual errors:

$$\mathrm{q}(\boldsymbol{T}_{\mathrm{slam}}, \boldsymbol{T}_{\mathrm{real}}) = -\frac{1}{S_\mathrm{T}} \sum_i^{S_\mathrm{T}} (\|A(\boldsymbol{T}^i_{\mathrm{slam}}, \hat{\boldsymbol{\theta}}_T) - \boldsymbol{T}^i_{\mathrm{real}}\|^2) \tag{4.9}$$

4.3.2 Optimising parameters for the slam algorithm

The SLAM algorithm depends on multiple parameters (Table 4.1). A first set of parameters is related to the error model of the motion model of the robot. They are the constant error, the proportional error, and the minimal uncertainty on pose. A second set of parameters concerns the processing power allocation policy. We have observed that tracing rays on the map consumes most of the processing power (>95 %). Thus to perform SLAM in real time we have a limited ray budget. On the marXbot, when performing 1.5 scan/s, this budget is 65000 rays per scan for a load of 100 %. The parameters related to this budget allocation are the particle count, the minimal angle between scans for measurement to map matching and the particle resampling frequency. The minimal angle between scans allows the scan-matching algorithm to match only a subset of the acquired scan data, to improve the speed of the matching process. This corresponds to resampling the scan data in polar coordinates.

These parameters affect the quality of the SLAM, but they are not obvious to measure nor compute. We thus propose to learn them from

parameter	initial value	mut. σ	best of	best of	best of	best of
ray budget	n.a.	n.a.	8125	16250	32500	65000
dist. error ratio	0.05	0.01	0.13	0.10	0.10	0.12
dist. error const	0.01	0.002	0.014	0.010	0.016	0.006
angle error ratio	0.05	0.01	0.045	0.002	0.064	0.10
angle error const	.01°	0.002°	0.01°	0.01°	0.02°	0.01°
min pos uncertainty	2	0.4	0.41	0.28	0.02	0.05
min angle uncertainty	5°	1°	4.3°	6.9°	8.4°	0.98°
particle resampling f.	1	0.5	1	1	2	5
angle between scans	0	2.5°	1.9°	2.9°	1.8°	3.7°
number of particles	1	1	1	1	1	1

Table 4.1 Parameters for the SLAM algorithm (left) and their values after optimisation (right, best individual of last generation). The particle resampling frequency is irrelevant when the particle count is 1.

experimental data. Our experimental setup allows the recording of the robot's odometry and scanner data (Figure 4.3). We have synchronised the tracker with this telemetry using ASEBA, which allows us to replay any experiment with any set of parameters. We utilise this feature to optimise the set of parameters. To do so, we implement a simple evolution strategy [8]. This evolution strategy operates on the real-valued space of the parameters. We do not perform self-adaptation, and after each generation, we keep the best 25 % individuals. For each of these individuals, we make three mutated copies to replace the discarded 75 % individuals. During evolution, we perform simulated annealing by varying the mutation factor from 100 % to 20 %. Table 4.1 gives for every parameter its initial value and the standard deviation of the normally-distributed mutation added value. Note that we evaluate an individual over several runs. For each run, we evaluate the quality of its reconstructed trajectory using Equation 4.9. To do so, we find $\hat{\boldsymbol{\theta}}_T$ for each run using the method we explained in Section 4.3.1.

The quality measure $\mathrm{q}(\boldsymbol{T}_{\mathrm{slam}}, \boldsymbol{T}_{\mathrm{real}})$ from 4.9 is well suited for human interpretation. However, it is highly non-Gaussian: if the robot looses itself during the map creation, the quality will be orders of magnitude worse than in a successful map reconstruction. To alleviate this effect, we let the

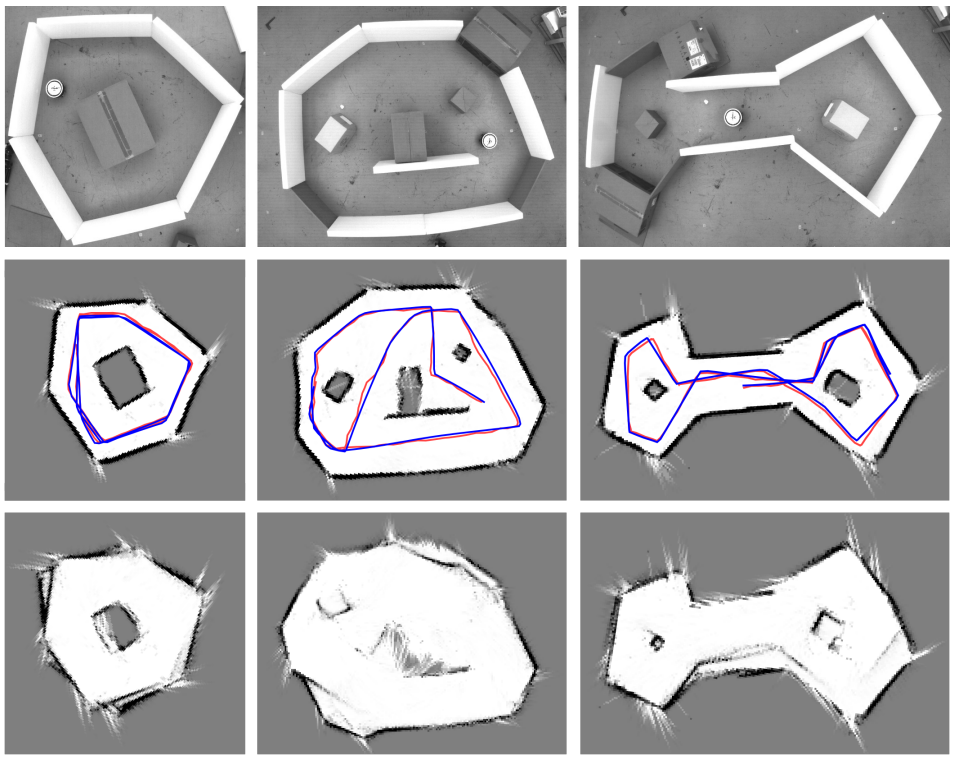


Figure 4.4 The different experimental environments (top), and the maps built by our SLAM implementation (middle), using a budget of 65000 rays. The SLAM trajectory is in light red while the real trajectory (tracker) is in dark blue. The bottom shows the map reconstruction while ignoring the phase B (measurement to map matching) of our algorithm.

evolution strategy minimise the following term instead of the quality:

$$e(\boldsymbol{T}_{\text{slam}}, \boldsymbol{T}_{\text{real}}) = \log(1 - q(\boldsymbol{T}_{\text{slam}}, \boldsymbol{T}_{\text{real}})) \tag{4.10}$$

This results in a smoother evolution, because we evaluate each parameter set over 5 recorded experiments in three different environments and take the mean in a log scale.

4.4 Results

We run 5 experiments of 5 minutes each, in 3 different environments (Figure 4.4). We let the marXbot move freely and avoid obstacles using its short range proximity sensors. We recorded the robot's scans, odometry and absolute position from the tracker. We then evolved the parameters

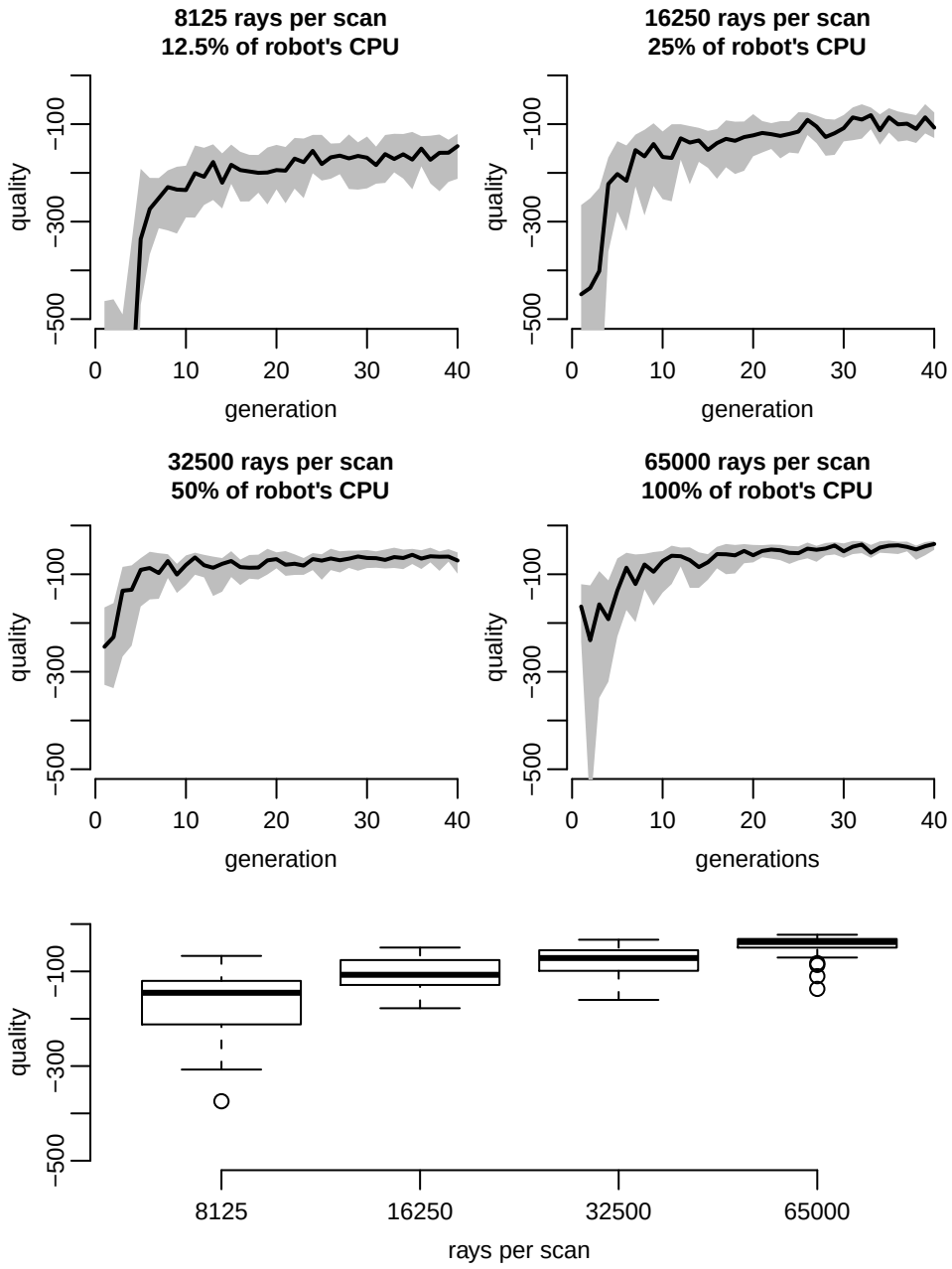


Figure 4.5 Optimisation of the parameters for different ray budgets. We have evolved populations of 48 individuals, over 40 generations, by evaluating each parameter set over 5 recorded experiments for 3 different environments. The top plots show the four evolutions. The black line represents the median and the grey area represents the interquartile range of the quality (see Equation 4.9). The bottom boxplot shows a comparison of the last generation.

ray budget:	16250	32500	65000
8125 rays; 12.5% of robot's CPU	6.6e-7	2.1e-14	2.2e-16
16250 rays; 25% of robot's CPU		1.0e-4	2.2e-16
32500 rays; 50% of robot's CPU			6.1e-11

Table 4.2 P-values of the Mann–Whitney U statistical test between all individuals of the last generation of evolutions for different ray budgets. The alternative hypothesis is that the error on trajectory reconstruction is the same for different ray budgets.

corresponding to allocating 1, $\frac{1}{2}$, $\frac{1}{4}$, and $\frac{1}{8}$ of our processing budget to SLAM. As Figure 4.5 shows, allocating more processing resources leads to statistically significantly better maps (Table 4.2).

The evolution was free to use several particles, to the expense of the quality of the robot's pose optimisation during phase B of the SLAM algorithm. Yet the evolution always kept a single particle, and adapted the minimum uncertainty on position in regards to the available computational power (Table 4.1). The more rays were available, the smaller uncertainty the evolution kept. It seems that in our setup, a small number of particles do not hold enough different possibilities to be worth the investment in computational power. Moreover, the scan-matching step reduces the need for particles, as it locally simulates several particles. We cannot rule out that a longer evolution, with a larger population, and with more experiments per evaluation would lead to the use of more particles.

All our environments are small, with respect to the range of the rotating scanner. It would be interesting to allow more computational power and to increase the size of the environment to see when multiple particles would get used. In particular, in environments with large open areas, scan-matching would not be enough to maintain the consistency of the map. In these cases, the robot would have to solely rely on odometry to estimate its pose, and multiple particles would be very helpful to track multiple location and map hypotheses. Unfortunately, the minimum number of particles to correctly reconstruct such environments will increase with the size of the environment, in particular with the size of the open areas. A finite number of particles would not solve such situations in general, adding more particles would only extend the set of environments in which the SLAM algorithm works.

At the qualitative level, we see that all our three environments are well reconstructed (Figure 4.4). One exception are the corners, which our

scanner tend to see as holes in the walls. This is due to the orientation of the sharp sensors and their triangulation-based distance measurement. Corners create reflections which lead to wrong readings from the sensors. We could alleviate this effect by mounting the sensors vertically, but that would triple the height of the scanner. We could also post-process the grid map knowing that walls are flat [118] and thus work around the problem of the corners. Finally, if the robot physically approaches the corners, it can use its proximity sensors to see the walls and thus to compensate for the missing information from the scanner.

Several researchers have proposed to take profit of a global optimisation algorithm to perform SLAM [26,32]. However, these works employ the algorithm to estimate the posterior probability distribution over trajectories or maps, which is taken care by our particle filter. To our knowledge, there is no previous work about the use of a unified optimisation algorithm to find the parameters of the robot's motion model and to allocate processing resources.

4.5 Lessons learnt

The choice of orienting the sensors horizontally, to save vertical space, has the effect of introducing important errors in the perception of the corners, as seen in Figure 4.4. Given that all obtuse angles are not seen correctly, we think that the orientation choice for the sensors was sub-optimal.

While developing the SLAM algorithm, we have noticed that constant bias are critical for the convergence of the algorithm. At some point in the development, we made a sub-pixel rounding error in the transformation between the integer map pixel and the floating-point world coordinates. This error prevented the convergence of the SLAM, as it created drifts in the map. This example highlights the difficulty of developing complex algorithms for real robots, for which one must consider both their abstract mathematical properties and the tiny details of their implementations.

4.6 Conclusion

In this chapter, we have demonstrated a global optimisation of a robot's intrinsic parameters and of the allocation of processing resources. This optimisation allows an inexpensive sensor coupled with a low-speed processor to perform SLAM in simple environments in real time, which

validates Hypothesis 4.1 (p. 49). We note that the optimisation has opted for a single-particle SLAM, thus using solely scan matching to build a consistent map. It would be interesting to increase the ray budget and the difficulty of the environment to see when the optimisation would switch to multiple particles and how many it would use. Unfortunately we do not have the physical space for much larger areas nor do we have the time and the computational resources to conduct such a research, as optimising over the parameter space demands a lot of computing power. In the current setup, for each individual of each generation, the optimisation must run 15 SLAM of 5 minutes each. Nevertheless, the fact that the optimisation decided to drop the particle filter in favour of a robust scan matching is a major contribution to the fundamental hypothesis of this thesis, Hypothesis 1.1 (p. 2), because the structure of the world and the processing-power constraints did influence the core of the algorithm.

Chapter 5

Semantic maps and symbol grounding

In the previous chapter, we have seen that the SLAM algorithm is able to build a map of the environment. In the case of the marXbot and the rotating distance scanner, this map is an occupancy grid showing whether there are obstacles at the height of the scanner (9 cm). However, there is a richer variety of elements in the world than what the scanner is able to capture. In particular, to build an application beyond simple exploration, the marXbot must use its diverse sensors to perceive the environment. As the final demonstration of this work, we are interested in enabling the marXbot to autonomously build structures in a world of unknown geometry and topology (Chapter 7). In this chapter, we show how to combine together the outputs from the SLAM algorithm and from the different sensors to build a representation of this environment (Figure 5.1).

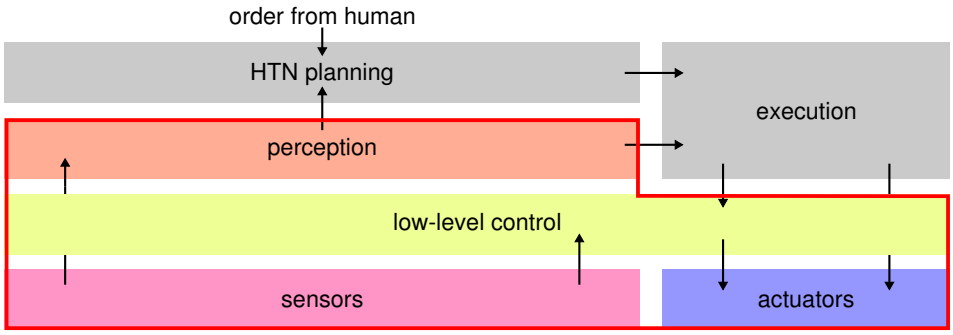


Figure 5.1 The semantic-map part within the bloc scheme of the autonomous construction application.

As we can see in Figure 5.2 (top), the terrain consists of multiple areas, separated by valleys of different widths and depths. The robot knows that the smaller the width of the valleys the shallower their depths. It uses this information by planning bridges where the distance between two areas is the closest. The environment also contains resources, which are small cubes of expanded polystyrene.

The aim of the perception subsystem is to create a symbolic representation of the world that is grounded in reality. Indeed, to reason about the world the robot needs a symbolic representation, but to execute the results of this reasoning this representation must be linked with geometric information. This leads to a dual symbolic-geometric representation of the world in which symbols are grounded by geometric data. We think that for our application, this representation is the best suited:

Hypothesis 5.1. *A dual symbolic-geometric representation of the world fits the needs of a construction scenario.*

This model is similar to what other researchers working on semantic maps have proposed [59, 79, 119]. However, probably because in these works the robot does not manipulate the world, it does not keep as much geometric information as our robot does. Moreover, the software presented in these works is not designed to run on embedded processors, so they are relatively free to use heavy computations, where in our case we must take into account the limitations of our embedded processor.

5.1 The perception process

As Figure 5.3 shows, the perception process is made of three layers of increasing abstraction. The first layer consists of three probabilistic maps, for the walls, the valleys and the resources. For the second layer, we combine the maximum likelihood estimations of the probabilistic maps to create two segmentation maps, one for the ground areas and one for the resources. The third layer contains the dual symbolic-geometric representation. We build the latter by analysing and filtering the segmentation maps to extract symbolic elements, representing the areas, their connectivity, the resources, their associated area and the robot itself.

5.2 Probabilistic maps

The lowest layer of perception consists of three probabilistic maps. These maps are two-dimensional occupancy-grid maps [29] of walls, val-

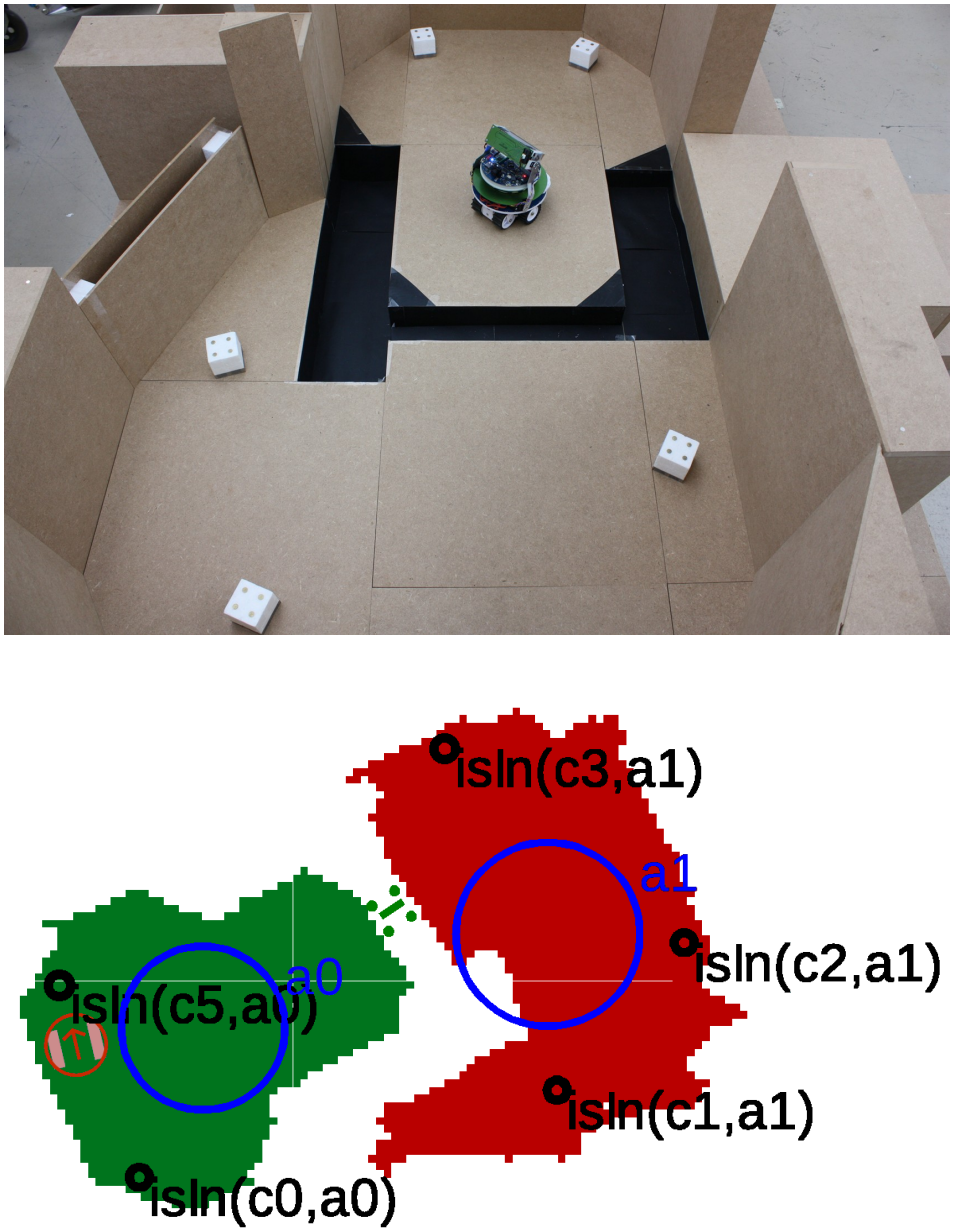


Figure 5.2 The environment of which the marXbot builds a representation. Top: a photography of the real arena, the black zones are valleys. Bottom: a screenshot of the software which builds a representation of this environment. The solid areas represent the two sides of the valley. The small black circles represent the resources. The blue circles are located on the centres of mass of the areas and their radius are proportional to the surfaces of the areas.

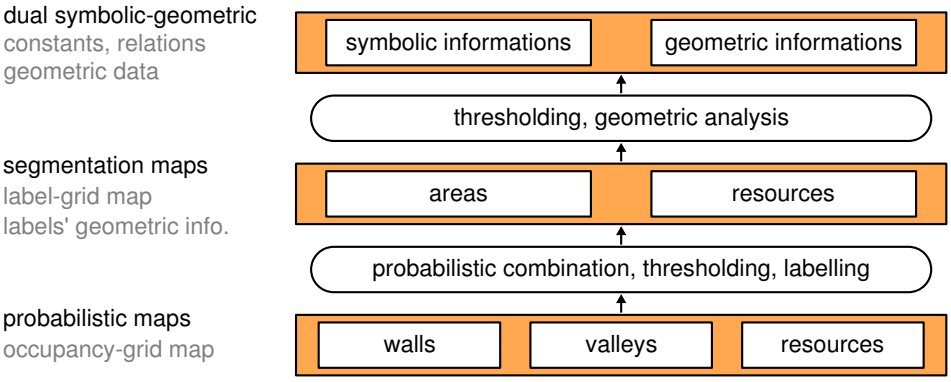


Figure 5.3 The three layers of the perception process

leys and resources. They have a spatial resolution of 2 cm. We represent each map m by the set of all grid cells $m = \{m_i\}$, where m_i is a binary random variable with $p(m_i = 1)$ representing the probability that the cell i contains the corresponding element: a wall, a valley or a resource. Note that for simplicity and for compatibility with the literature we use a single index i but remember that the maps are two-dimensional. We assume that each cell is independent, which allows us to approximate the posterior of the map as:

$$p(m) = \prod_i p(m_i) \tag{5.1}$$

This assumption is a major simplification of the reality, because most objects and free areas span multiple cells. However, a more refined assumption would greatly complicate the probabilistic model; for this reason, most works in the literature employ this simplistic assumption. For the same reason, we use it as well. At the implementation level, each cell holds the log odds ratio of the probability that it contains an element [108, p. 94, p. 286], with a resolution of 16 bits:

$$l(m_i) = \log \frac{p(m_i = 1)}{1 - p(m_i = 1)} \tag{5.2}$$

This implementation choice is convenient because computing the posterior given an observation corresponds to summing the log odds ratios. If $o_{1:t}$ is the vector of all observations, with o_t the observation at time t, then:

$$l(m_i|o_{1:t}) = l(m_i|o_t) + l(m_i|o_{1:t-1}) - l(m_0) \tag{5.3}$$

where $l(m_0)$ is the log odds ratio of the prior probability that an element occupies the cell, $l(m_i|o_{1:t-1})$ the previous value of the cell, and $l(m_i|o_t)$

the probability that an element occupies the cell given the most recent observation. As all cells are independent, in the rest of this chapter we will omit the index i when discussing operations on maps that apply to every cell.

The log odds representation cannot express complete certainty, as $\lim_{p(X=1)\to 0} l(X) = -\infty$ and $\lim_{p(X=1)\to 1} l(X) = \infty$. Thus, we choose a small value ε that corresponds to the smallest probability that we can express with our representation. The choice of ε leads to bounds for l:

$$\begin{aligned} p(X=1) < \varepsilon &\implies l(X) = l_{\min} \\ p(X=1) > 1-\varepsilon &\implies l(X) = l_{\max} \end{aligned} \tag{5.4}$$

Often the probability to observe something with a sensor depends on what has been sensed by another sensor. For instance, when an infrared proximity sensor detects an obstacle, this reading may be due to a resource or to a wall. In this case we must use the wall map to disambiguate the reading. Formally, in this example the probability to detect a resource is the product of the probability to detect an obstacle and the probability that this obstacle is not a wall: $p(\text{resource}) = p(\text{obstacle})\,(1 - p(\text{wall}))$. Albeit this formula is simple, it does not take a simple closed form when expressed in log odds ratio. Indeed, given three binary random variables X, Y, Z, such that $p(Z=1) = p(X=1)p(Y=1)$, there exist no real numbers a, b, c such that $l(Z) = a \cdot l(X) + b \cdot l(Y) + c$. However if we take $a = 1, b = 1$ and $c = l_{\min}$, we can approximate the $l(Z)$:

$$p(Z=1) = p(X=1)p(Y=1) \implies l(Z) \approx l(X) + l(Y) + l_{\min} \tag{5.5}$$

Figure 5.4 shows the superposition of the true and the approximate $l(Z)$. These two functions are not too different, in particular around $l(Z) = 0$, which means that there are only few cases where the belief is increased when it should be decreased, and reversely. Because it only performs two additions, the approximate function is very fast, and thus well-suited for real-time processing on a real robot. In our implementation of sensor data fusion, we use the approximate function instead of the true function.

5.2.1 Sensing and data fusion

We create the three maps by fusing the data of the multiple sensors shown in Figure 2.4 (p. 12). The wall map is the output of the SLAM algorithm, which is explained in Section 4.2 (p. 51). The valley map is the result of the fusion of the wall map and the outputs of the ground sensors and the vision. The resource map is the result of the fusion of the

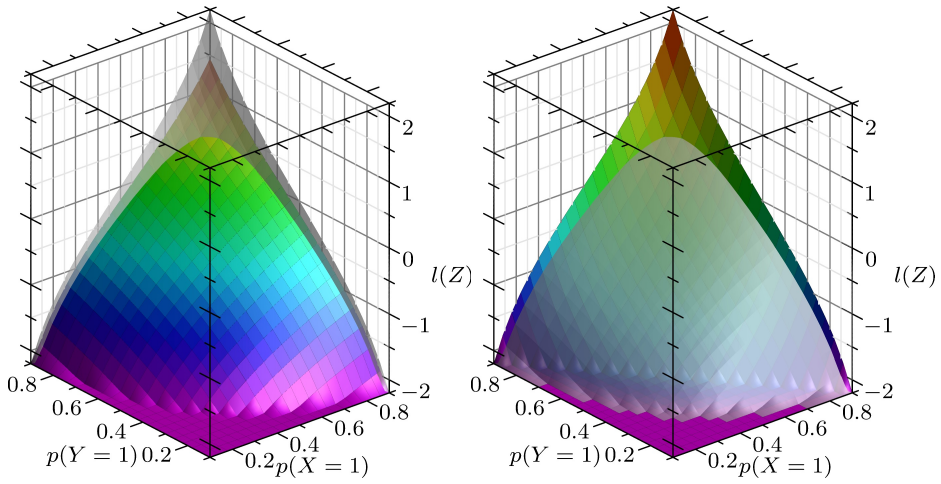


Figure 5.4 Simplification of the log odds ratio for the product of probabilities. Given three binary random variables X, Y, Z such that $p(Z = 1) = p(X = 1)p(Y = 1)$, these plots show true and the approximate $l(Z)$. Left: the true function in solid rainbow, the approximation in shaded. Right: the approximation in solid rainbow, the true function in shaded.

wall map and the outputs of the proximity sensors and the vision. The ground and proximity sensors provide short range information, but with a high confidence and a quick refresh rate. The vision provides long range information, but at a slower pace with a lower confidence than infrared sensors.

5.2.2 Ground sensors

There are 8 infrared proximity sensors distributed around the marXbot and directed towards the ground. These sensors give information about the presence of valleys at close range. The marXbot uses these sensors to avoid valleys and update the valley map.

5.2.3 Proximity sensors

The marXbot has a ring of 24 outward proximity sensors that perceive obstacles at a height of about 6 cm. In our setup, these obstacles can either be walls or resources. Formally, the probability to see a resource is:

$$p(\text{resource} = 1) = p(\text{obstacle} = 1)(1 - p(\text{wall} = 1)) \tag{5.6}$$

We implement this product using the approximation from 5.5, looking up into the wall map to update the resource map.

5.2.4 Vision

We use vision to locate resources and to detect valleys at long range ($> 40\,\text{cm}$). This is possible because we know that the ground is flat and that the resources are cubes with a height of 6 cm, always located on the ground. Let R be a three-dimensional frame of reference centred on the robot, with x and y in the ground plane, x pointing forward and y pointing left, and z pointing upward out of the ground plane. Let $w = \{w_x, w_y, w_z\}$ be a point in the world given R, and $i = \{i_x, i_y\}$ be a point on the camera image (see Figure 5.5, left). Based on the camera equations [49], given i and the altitude w_z, we compute w_x and w_y as follows:

$$\begin{aligned} w_x &= \mathrm{cam_x} + \mathrm{a_v}\frac{\mathrm{cam_z} - w_z}{i_y - \mathrm{c_v}} \\ w_y &= \frac{(i_x - \mathrm{c_u})(w_x - \mathrm{cam_x})}{\mathrm{a_u}} \end{aligned} \tag{5.7}$$

And, conversely, given w we compute i as follows:

$$\begin{aligned} i_x &= \mathrm{c_u} + \mathrm{a_u}\frac{w_y}{(w_x - \mathrm{cam_x})} \\ i_y &= \mathrm{c_v} + \mathrm{a_v}\frac{(\mathrm{cam_z} - w_z)}{(w_x - \mathrm{cam_x})} \end{aligned} \tag{5.8}$$

With $\mathrm{c_u}, \mathrm{c_v}, \mathrm{a_u}, \mathrm{a_v}, \mathrm{cam_x}, \mathrm{cam_z}$ as constants. Let $\mathbf{S}$ be the space of these constants. We calibrate the constants by minimising the squared error e_{proj} between a training set of hundred couples of points $\{(\mathrm{w}^j, \mathrm{i}^j)\}$ and their projections i_x, i_y over $\mathbf{S}$:

$$\begin{aligned} e_{\text{proj}} &= \sum_j \left((i_x(\mathrm{w}^j) - \mathrm{i}_x^j)^2 + (i_y(\mathrm{w}^j) - \mathrm{i}_y^j)^2 \right) \\ (\mathrm{c_u}, \mathrm{c_v}, \mathrm{a_u}, \mathrm{a_v}, \mathrm{cam_x}, \mathrm{cam_z}) &= \underset{\mathbf{S}}{\operatorname{argmin}}\, e_{\text{proj}} \end{aligned} \tag{5.9}$$

We do so using a simple evolution strategy [8] using 32768 individuals over 1000 generations. This evolution strategy operates on the real-valued space of the constants. We do not perform self-adaptation, and after each generation, we keep the best 25 % individuals. For each of these individuals, we make three mutated copies to replace the discarded 75 % individuals. During evolution, we perform simulated annealing by varying the mutation factor from 200 % to 0 %. The initial value are $(\mathrm{c_u} = 192, \mathrm{c_v} = 256, \mathrm{a_u} = 500, \mathrm{a_v} = 500, \mathrm{cam_x} = 5, \mathrm{cam_z} = 14)$. For mutations, the standard deviation of the normally-distributed added value is 2

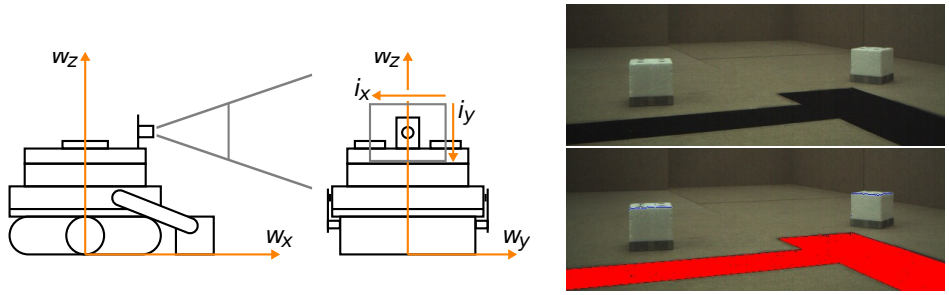


Figure 5.5 The vision on the marXbot. Left: the coordinate system used for computing projections. Right, top: the world as seen by the camera. Right, bottom: the result of the processing: in red, the valleys; in blue, the top lines of the cubes (resources).

for c_u, c_v, a_u, a_v and 0.1 for cam_x, cam_z. Finding these constants allows us to project back and forth a point between its world and screen coordinates, as long as we know the height of the point in world coordinates.

The raw resolution of the camera chip is 2048×1536; the chip has a Bayer filter. We implement the Bayer to RGB transformation by downscaling the image of a factor 2, which produces a high-quality output compared to the usual interpolation-based methods. Thus the maximum usable resolution for our application is 1024×768. As our probabilistic maps have a resolution of 2 cm, and the robot is slightly shaking when moving, we do not need such a high resolution. Thus we only take an image of 128×384, with the advantage of exploiting the hardware average filter of the camera chip. This allows us to use a very short exposition time and still capture noiseless images. We only process the lower half of the image, as there are no objects of interest above the altitude of the camera. We could exploit the full frame if we would tilt the camera, however this would complicate Equations 5.7 and require a hardware re-design.

We process the image from its bottom to its top. The actual detection is different for the valleys and for the resources (see Figure 5.5, right). Let $\{r, g, b\}$ be a pixel and i be its intensity defined as $i = r + g + b$. This pixel might correspond to a valley if $i < t_{\text{intensity valley}}$ and $|r - g| < t_{\text{color valley}}$ where t_x are thresholding constants. Otherwise, the pixel might correspond to the ground. Once we have classified a pixel as ground or not, we back project it from the image to the world, assuming an altitude of 0 cm. Then, we cast a line on the wall and the resource maps from the camera position to the back-projected coordinate. If any of these lines crosses a point with a probability to contain a wall or a resource larger than 0.5, we discard the point. The rationale is that a point seen behind a

wall or a resource is not a genuine piece of information about the presence of a valley.

The detection of the resources works by comparing together pixels of the same column to find the top line of the cubes. Our model is that the top line of the cubes is the most intense part of the image, as it is white and receives light from 75% of its sides. This model is inspired from the ambient occlusion technique used in computer graphics [61]. Given the pixel $\{r, g, b\}$ of intensity i, if $i > \mathrm{t}_{\text{intensity cube}}$ and $b + \mathrm{t}_{\text{colour cube}} > r$ and $i > i_{\text{prev}} + \mathrm{t}_{\text{cube delta}}$, this pixel might belong to the top line of a cube. Note that i_{prev} is the intensity of the last candidate pixel, and that t_x are thresholding constants. Assuming that our lighting model is correct, and knowing that we process pixels from the bottom to the top of the image, we are sure that the last pixel meeting the condition belongs to the top line. Like in the case of valleys, we back project this pixel from the image to the world, this time assuming an altitude of 6 cm. Then we cast again a line on the wall map to ensure that this pixel really corresponds to a visible resource.

Figure 5.5 shows the raw image and the processed image; we use a resolution of 512×384 for these screenshots to provide a normal aspect ratio.

5.3 Segmentation maps

Out of the three probabilistic maps, we create two segmentation maps, as seen in Figure 5.2 (bottom). The first map is the area map. It is based on the fusion of the wall and the valley maps:

$$p(\text{area} = 1) = (1 - p(\text{wall} = 1))\,(1 - p(\text{valley} = 1)) \tag{5.10}$$

We consider a point for segmentation if its probability to be an area, $p(\text{area} = 1)$, is above a certain threshold. We use the approximation from 5.5 to compute this probability. Then, we apply Algorithm 5.1 to create a list of areas. This algorithm also computes the surface of each area and its centre of mass.

The second map is the resource map. We use the same procedure as for the area map. We take directly $p(\text{resource} = 1)$ from the probabilistic resource map as the input of Algorithm 5.1.

```
{Phase 1: segment regions and note equivalent relations}
l ← 0
for y = 0 to height do
  for x = 0 to width do
    if values[x][y] > t then
      l_top ← labels[x][y − 1]
      l_left ← labels[x − 1][y]
      if valid l_top then
        if valid l_left then
          equivalence ← node(l_top,l_left)
        end if
        labels[x][y] ← l_top
      else
        if valid l_left then
          labels[x][y] ← l_left
        else
          labels[x][y] ← l , l ← l + 1
        end if
      end if
    end if
  end for
end for
{Phase 2: fuse equivalent relations}
l ← 0
for c = every clique in equivalence do
  for n = every node in c do
    lookup[n] ← l
  end for
  l ← l + 1
end for
{Phase 3: store final labels }
for y = 0 to height do
  for x = 0 to width do
    labels[x][y] ← lookup[labels[x][y]]
  end for
end for
```

Algorithm 5.1 The map segmentation algorithm. A first phase scans the map and assigns a label to a position if its value exceeds a threshold t. This phase tries to assign the same label to adjacent regions; however when two existing regions join, they have different labels. In this cases, the algorithm stores the two labels into an equivalence graph. The second phase iterates over the cliques of this equivalence graph and builds a lookup table to fuse equivalent labels. The last phase applies the final labels on the map using the lookup table.

5.4 Symbolic-geometric representation

Out of the two segmentation maps, we create a dual symbolic-geometric representation of the environment. At the symbolic level, this representation is based on first-order logic, which is suitable to define a state space for automated planning (see Chapter 6). We refer to real-world objects by symbolic constants, such as `a3` for an area or `r7` for a resource. We type these constants by unary relations, such as `area(a3)` or `resource(r7)`. We relate real-world objects together using n-ary relations over their symbolic constants, such as `isIn(r7, a3)`. At the geometric level, we create maps from symbolic constants to geometric data, for instance we store the centre and the surface of areas and resources.

For each area in the area segmentation map, we check whether its surface is larger than a certain threshold, in our case 200 cm^2, and if so we assign a symbolic constant to the area. We build a table mapping the symbolic constant to a tuple containing the label identifier, the centre of mass and the surface of the corresponding area. Then, we search for pairs of areas that have a small valley between them. To do so, we search for the closest points between the two areas using Algorithm 5.2. This algorithm is not trivial because the areas might not be convex. However, it is fast for relatively convex areas, for which most points drawn following a distribution $\mathcal{N}(\text{centre}, \sqrt{\text{surface}}/2)$ lie inside the area. If the areas would be highly non-convex, this algorithm would need a huge number of iterations to have a good chance of finding the closest points. In that case, it would be better to employ a grid-based algorithm such as A*. The output of the algorithm, the resulting two closest points, are the extremities of a potential bridge. If the distance between these two points is lower than a threshold, in our case 12 cm, we consider the areas to be connectable. We create the logical relation `isConnectable` and store the extreme points of the bridge in a table, indexed by the symbolic constants of the two areas. After processing the areas, we lookup the area in which the robot lies and add an `isIn` relation between the robot and its area.

Then we process the resources. For each resource in the area segmentation map, we check whether its surface is larger than a certain threshold, in our case 32 cm^2, and if so we assign a symbol to the resource. Then we lookup the position of the resource into the area segmentation map and the area table to find in which area this resource lies. If the resource is indeed located within an area, we add an `isIn` relation between the resource and the area. We build a table mapping the symbol constant to a tuple containing the label identifier, the centre of mass and the surface of

$d_{\text{min}} \leftarrow \infty$
for $i = 1$ to N **do**
  draw $p_0 \sim \mathcal{N}(c_0, \sqrt{s_0}/2)$
  draw $p_1 \sim \mathcal{N}(c_1, \sqrt{s_1}/2)$
  $p_0 \leftarrow$ last point in a_0 on the line (p_0, p_1)
  $p_1 \leftarrow$ last point in a_1 on the line (p_1, p_0)
  $d \leftarrow \|p_1 - p_0\|$
  if $d < d_{\text{min}}$ **then**
    $d_{\text{min}} \leftarrow d$
    $p_{\text{closest}} \leftarrow (p_0, p_1)$
  end if
end for
return p_{closest}

Algorithm 5.2 The algorithm to find the closest points between two areas, based on a Monte Carlo method of N iterations. Two areas a_0 and a_1 are given in input, with centres c_0, c_1 and surfaces of s_0, s_1.

the corresponding resource.

5.5 Implementation

The implementation of the perception process is distributed into several programmes (see Figure 5.6). The obstacle avoidance and the odometry are implemented using ASEBA and run on the different microcontrollers. We also use ASEBA to stream data from the rotating distance scanner, the ground sensors and the proximity sensors. A stand-alone programme acquires images from the camera and implements the image segmentation. It builds a bitmap of the lower part of the image to indicate whether a pixel appears to be a valley. For the resources, this programme finds for each pixel column the position of the line of the cube, if one exists. This programme always runs on the robot, as it uses the video for Linux interface. Each time the programme shoots an image, it sends an event to ASEBA to ensure synchronisation with the data from the other sensors. A third programme implements the perception process itself. This programme connects to ASEBA and to the image segmentation programme through TCP/IP. It can thus either run on the robot itself or on a remote computer. In the latter case, this programme provides a GUI. This GUI displays the probabilistic or the segmentation maps, and overlays the symbolic and geometric information. In addition, it can log statistics on regular intervals

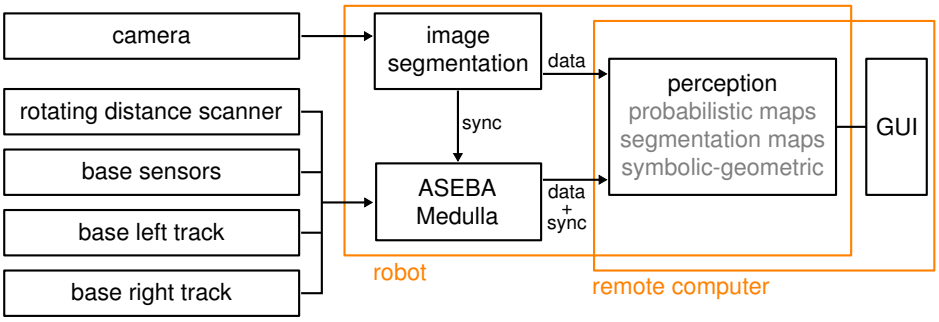


Figure 5.6 The software implementation of the perception process. The high-level perception software can run either on the robot itself or on a remote computer. We set memory and computational requirements to allow this software to run in real-time on the robot. To avoid implementing remote visualisation, when using a GUI we run this software on the remote computer showing the GUI.

to enable analyses.

The exploration behaviour of the robot aims at providing an efficient coverage at a low computational cost. To do so, the robot has an ordered list of relative points. The exploration algorithm runs through this list, and for each point checks whether it is explorable. If so, the algorithm sets a speed command to steer the robot to this point. If no point of the list is explorable, the robot simply goes straight. A point is explorable if it has not been explored yet, that is, if the valley map contains a low certainty for this point. Moreover, the point should not be on the other side of a wall or of a resources, considering the current position of the robot. If the position of the robot has not changed for 5 seconds, the robot enters an emergency unblocking behaviour. It rotates in a random direction for a random time, and then moves straight on for about 3 seconds. The exploration algorithm is executed at about 1 Hz. At the low level, the microcontrollers running ASEBA implement obstacle and hole avoidance autonomously. This behaviour consists of two vector-field avoidance algorithms [10]. If the robot perceives a hole or an obstacle, it avoids it, otherwise it performs exploration. Holes have priority over obstacles, and as avoidance consists in turning on the spot, the hole avoidance behaviour cannot bring the robot into an obstacle. The exploration algorithm must only provide a good exploration, as it delegates safety considerations to the microcontrollers.

5.6 Results

To analyse the performance of our representation system, we have conducted 5 experimental runs of 25 minutes each in the environment shown in Figure 5.2. At the start of every run, we place the robot in the centre of the small area (the one with two resources). The robot faces the opposite area. We keep at least 10 cm between the robot and the walls and the valleys, as the sensors self-calibrate at the beginning of a run.

To understand how the representation of the world evolves over time, we analyse the entropy of the probabilistic maps and the number of grounded objects over time. We take one measure every second. For the probabilistic maps, we are interested in the quantity of information in these maps. Following the definition of mutual information, we define this quantity as the entropy of the map prior to observation minus the entropy of the built map. Let us call m_t the map at time t and m_0 the prior of the map. Remember that we represent a map m as a set of independent grid cells $m = \{m^i\}$. Thus the difference of $I(m)$ for the map m at time t is:

$$\begin{aligned} I(m,t) &= H(m_0) - H(m_t) \\ &= H(\{m_0^i\}) - H(\{m_t^i\}) = |\{m^i\}|H(0) - \left(\sum_i H(m_t^i)\right) \end{aligned} \tag{5.11}$$

where $H(0)$ is the prior entropy of the map corresponding to the value with which the cells of newly created maps are filled. This difference is positive, because the knowledge about the world grows with time while the robot explores. For the grounded objects, we log the number of areas, of resources and of bridges.

Figure 5.7 overlays the evolution of the representation over time for the 5 runs. The top plot shows the probabilistic maps. Be warned that this plot shows how much information the robot think it holds about the world, but does not say anything about the correctness of the information. In this plot, we see that the SLAM map has the fastest growing quantity of information. This is reasonable as the rotating distance scanner can see both close and far, and scans around the robot. On the long run, the SLAM map holds slightly more information than the resource map. One reason for this is that the SLAM algorithm converges while the resource perception is always subject to the imprecisions of the temporal synchronisation between the positioning, the camera and the infrared sensors. Another reason is that the SLAM algorithm sets a prior on wall depth, which allows the wall map to cover a larger area than the resource map. We also see that the resource

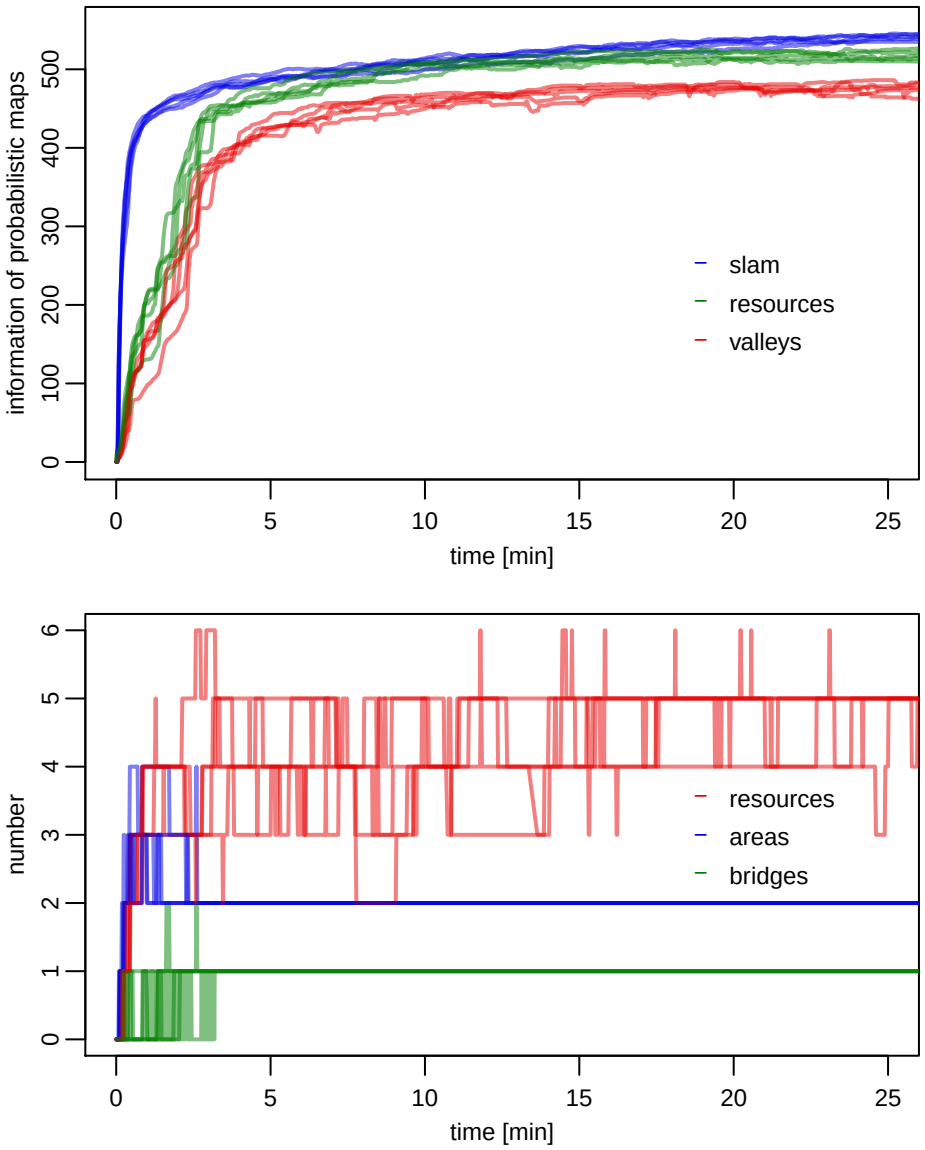


Figure 5.7 The evolution of the environment representation over time. These plots overlay the curves from 5 runs. Top: the information contained in the different probabilistic maps. Bottom: the number of grounded objects per type. In reality, there are 5 resources and 2 areas close enough to build a bridge between them.

map holds more information than the valley map. One reason is probably that thanks to the camera, the robot perceives a large patch of the ground continuously. As the tracks cause the robot to shake when it is moving, the sides of the valleys hold a low certainty. Moreover, the SLAM is not perfect. In particular, it might slightly stretch its map compared to the reality. Thus, when using its camera compared to its ground sensors, the robot sees the valleys at slightly different positions. On the sides of the valleys, these two sensors return contradictory information, which increases the entropy of the valley map. The reason for this stretch might be a constant bias due to the slipping of the tracks. It would be interesting to test this hypothesis by adding a correction factor in the parameters of the SLAM algorithm (see Table 4.1, p. 57), redo the optimisation and redo the runs of Figure 5.7 with the new parameters. Unfortunately, time constraints prevent us from conducting such an investigation.

In the bottom plot of Figure 5.7, we see that the number of areas and bridges quickly reaches the correct value, in all runs. The number of resources is also approximately correct, however it is less stable. In average over all runs, between 20 and 25 minutes the robot was detecting the right number of resources 72 % of the time (1008 correct detections for 1394 measures). The perception of the resources is imperfect because, as seen in Figure 5.2, three resources are in the remote area and are thus far from the robot. This long distance affects the perception in two ways. First, when the robot moves, the tracks create a slight vertical shaking, which in turn creates small displacements in the image's pixels. This results in large changes in the w_x coordinate of the resource, because of the division in Equation 5.7. Second, as the robot turns, the angular position of the object with respect to the robot frame changes very fast. As temporally the image and the odometry are not perfectly synchronised, this results in errors to the w_y coordinate of the resource. These problems also affect the perception of the valleys, but as the valleys are larger than the resources, this does not disturb the perceived topology of the world, even if this increases the entropy of the valley map.

5.7 Discussion

While the robot moves around, it continuously receives information from its different sensors. Thus the robot acquires different sensor data from different positions in the world. All data originating from ASEBA arrive with a first-in first-out (FIFO) policy, which ensures their temporal synchronisation. However, as we can see in Figure 5.6, because of the

size of the images the acquisition programme does not transmit its data through aseba. Thus to synchronise images with the other sensor data, this programme sends an aseba message each time it has acquired an image. The main perception programme stores the estimated position of the robot when it receives this message, and then uses this position when it processes the image. As we do not use a real-time Linux kernel, this synchronisation schema does not provide any temporal guarantee. This results in positioning errors for the camera data when the robot rotates quickly. Currently, we alleviate this problem by limiting the robot's displacement and rotation speeds during exploration. To solve these problems in further developments and to allow the robot to move at faster speeds, we should timestamp both the aseba messages and the images shot. The video for Linux interface allows the latter, while we could modify Medulla to timestamp incoming messages. We could then use these timestamps to fuse the sensor data together, considering the robot position at the time of data acquisition.

The camera provides perception of valleys and resources beyond 30 cm and the infrared sensors give close-up information, so we do not have any sensor covering the range of 5 to 30 cm. This is sub-optimal, as given the size of the arena, this forces the robot to explore its surroundings in details. This operation demands a lot of time and does not take a good advantage of the range-sensing capabilities of the camera. However, time and production constraints prevent us from improving the sensor placement. In the context of our study of how integration affects the application, this negative example still contributes interesting elements. It shows us that the sensors should provide the right piece of information that the high-level control requires. This emphasises the importance of a co-design of hardware and software, or, in the case of this example, the negative effects of the lack of co-design.

The current system implements slam solely using the rotating distance scanner and the odometry/gyroscope. However, we could use the other sensors such as the camera or even the ground infrared sensors to contribute information to the slam algorithm. Yet, the number of combination possibilities is enormous and a slam implementation fusing data from various heterogeneous sensors is an open problem. We think that the most promising direction is to perform symbolic topological slam [6]. This method consists in abstracting local observations into symbols and relations between symbols, and then to perform slam at the symbolic level. By using the symbolic level instead of the metrical one, the probabilistic space is much smaller. Moreover, this method allows to use various

features and spatial relations between them. Recent works have explore the use of vision [20] and range data [6], and some have shown how to take the odometry readings into account [91]. Finally, symbolic topological slam is close to what we currently think to be the implementation of path integration and mapping in mamals [68].

Currently, we discard the existing symbols when we create the symbolic representation. However, it would be better to keep them and update them with the information from the new segmentation maps. To do so, we could match the existing regions with the new ones according to the distance between their centres and their respective surfaces. However, this simple procedure would not always be well suited, for instance when two areas get merged as the result of the exploration by the robot. We could explicitly handle this case by directly comparing the old and the new label information from the old and the new segmentation maps, and fuse the symbolic data accordingly.

Good visualisation and monitoring tools are extremely important for developing a system as complex as this one. In particular, the ability to see the different maps and scans in real time is critical. Indeed, the human eye is excellent at detecting small repeated errors. Moreover, the monitoring tool must be flexible to allow to visually inspect the intermediate steps of the different algorithms. Finally, it is important to overlay the dual symbolic-geometric representation over the probabilistic and the segmentation maps. This allows to easily validate the correctness of this representation.

Aseba provides logging and replay tools, that allow to re-create the stream of events the robot received during an experiment. This feature proved extremely useful in implementing the first draft of the perception subsystem. Indeed, on a fast computer it allows to re-run the experiment at several time the speed of reality. This multiplies the efficiency of finding bugs and implementing features. The camera programme does not support replays, but by taking snapshots we managed to make it work fairly easily. However, we are convinced that for complex application, a logging and replay architecture is of paramount importance.

Our symbol grounding stack uses several magical constants, such as the minimal size of an area or the maximal width of a valley allowing the robot to build a bridge. We chose these constants given our knowledge of the robot hardware and some preliminary experiments. It would be interesting to allow the robot to learn some of them. However, this is difficult as this requires low-level safety procedures, for instance using the accelerometre to back off when the robot begins to fall. Another solution

would be to learn these constants out of a simulation, but again setting up the simulation is a major work in itself. For these reasons, we think that hand-crafted constants are acceptables.

On the technical side, our probabilistic maps have values coded as 16-bit signed integers. Yet we use increments of about the order of 100 or larger, so 8-bit signed char would have sufficed. Our good grounding results also show that computing the log odds ratio of the product of probabilities using the approximation from 5.5, albeit crude, is sufficient.

5.8 Conclusion

Our representation system fuses data from different sensors and is organised in three layers of progressive abstraction. This system is able to perceive a topologically rich environment and to build a dual symbolic-geometric representation. This representation is a solid basis upon which to develop an autonomous construction application. As we will see in Chapter 7, this representation enables the robot to construct three-dimensional structures, which validates Hypothesis 5.1 (p. 64).

However, our experiments have shown the limits of the rotating distance scanner as the sole source of extrinsic information for SLAM. The limited range and precision of this scanner bounds the quality of the data fusion from other sensors, in particular from the camera. If we generalise this observation, we can state that if a first sensor has an error ε, and if the value of a second sensor depends on this error following a function $f(\varepsilon)$, then it is useless for the error of the second sensor to be lower than $f(\varepsilon)$. This leads to a Pareto-optimal [31] sensor allocation policy, where no sensor can increase its error without decreasing the performance of the whole system. Finding this optimal is difficult, as the errors combine in a non-linear way in general. There are several solutions to find this optimal, for instance using a simulator or by repetitive combinations of physical building blocks. In all cases some form of global optimisation is required. The important point is that if we take this search as a guiding principle for the design of perception systems, we can develop systems that are optimal for a given desired performance. This corroborates Hypothesis 1.1 (p. 2), and as the application of this principle allows better perceptions for a given money/sensing/computing budget, this also strengthens Hypothesis 1.2 (p. 2).

Chapter 6

Planner 9, a distributed HTN planner

Automated planning is closely linked to intelligence in robotics. The first automated planner was developed to drive Shakey, an early autonomous mobile robot [78]. Despite this common origin and the frequent use of planning in robotics, the two fields have largely diverged since. Most of the research on planning is currently evaluated on benchmark problems that are far from the robotic reality [62]. We have tested on a robotic scenario several planners from the literature, and found that none of them was satisfactory. Some just crashed, other failed to produce a plan or ran forever. The few which produced plans generated sub-optimal ones.

We thus have developed Planner 9, our own hierarchical task network (HTN) planner [40, ch. 11] based on our robotic needs. Planner 9 is a distributed HTN planner that runs concurrently on multiple mobile robots or on multi-core processors. We choose HTN planning because it is the mostly used planning technique for real-world applications [40, p. 229]. Our robots do not have a large processing power, being built around a smartphone-level computer which provides one tenth of the power of a laptop computer (see Chapter 2). However, these robots have a good Wi-Fi connectivity to their peers. Planner 9 takes advantage of this large bandwidth to processing-power ratio to distribute the planning over different robots. Based on our experience with existing planners, we have formulated the following hypothesis:

Hypothesis 6.1. *The requirements for a distributed robotic implementation of an HTN planner affect the execution properties of its algorithm. In particular, the scenario space for which the algorithm finds a solution and the solution plans are completely different.*

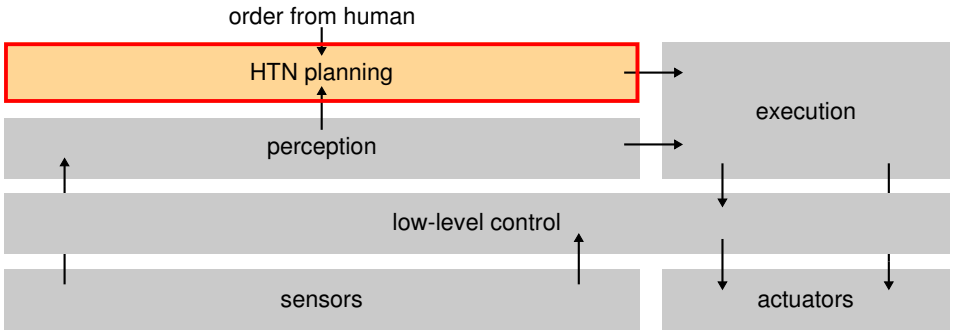


Figure 6.1 The HTN planning part within the bloc scheme of the autonomous construction application.

This chapter presents Planner 9 and gives experimental results that validate this hypothesis. To do so, we compare Planner 9 to JSHOP2, a Free and an often referenced HTN planner. We show that Planner 9 scales well with the number of robots and the complexity of the environment. This chapter also discusses the lessons learnt, the future works and situates Planner 9 in the broader context of this thesis. As Figure 6.1 shows, in the context of the autonomous construction application, Planner 9 provides the reasoning engine.

6.1 Related work

A vast literature exists on planning with multiple agents. However, the choice of a particular algorithm is a trade-off between several factors such as planning expressiveness, distributivity, bandwidth consumption and speed of execution [30]. In this section we focus on the approaches that are implementable on real robots with limited resources.

To distribute planning among different agents, one can run an independent planner on each agent. Works in simulated robotic football have used HTN planning that way [80]. However, in general this solution cannot improve the time nor reduce the memory required to solve a specific problem. In the direction of distributing the planning process, Dix et al. [25] have integrated the SHOP HTN planner within a distributed agent framework. The resulting A-SHOP planner uses the provided infrastructure to query the state of the world, evaluate preconditions, apply effects and estimate potential states from remote agents. However, in this work the planning itself is still centralised. Theoretical works in multi-agent systems have shown that it is possible to integrate the distributed aspect in the

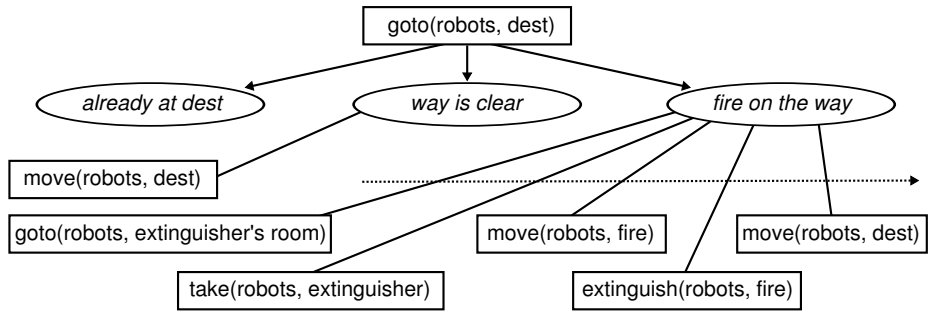


Figure 6.2 Methods for going to a room by recursively putting out fires on the way. The ellipses represent the three alternatives, while the rectangles represent tasks.

planning algorithm [22,27]. For instance Dias et al. [23] have proposed a market-based approach where *and/or* trees of tasks are exchanged between agents. These authors acknowledge the interest of more expressive planning but address the issues of distributivity and scalability first ("Further research will also investigate further generalisation of the tree structures and task constraints." [121, p. 25]). Thus their approach is less centralised but also less expressive than common HTN planning. Recent works in HTN have proposed stratified planning where remote agents plan sub-tasks and report them to a master. In Pellier et al. [82], finding the final plan is the result of the exchange of proposals and counter proposals between agents. In Hayashi et al. [51], the child agents are also responsible for execution, and interleave planning, execution and re-planning. These methods require a large number of synchronisation messages. On the contrary, Planner 9 considers the different robots as a computer cluster and distributes the planning of any task to any robot and thus takes advantage of all the available computational power using simple synchronisation.

6.2 Model and implementation

Planner 9 is a HTN planner. A HTN planner decomposes a goal task into sub-tasks until it finds a sequence of actions that the robots can perform. A task has a name and zero or more parameters. It can be implemented either by an action or by one or more methods. An action holds a precondition which must be true to allow its application. It also holds a list of effects that alter the state of the world; so the planner records these alterations when it applies the action. An action might have local variables that appear in its precondition and its list of effects.

Contrary to an action, a method does not affect the state of the world, but rather decomposes itself into other tasks. A method holds a precondition, a graph of sub-tasks, and possibly local variables. If several methods implement a task, we call these *alternatives*. The planner knows the available actions, methods and their possible alternative decompositions, like in Figure 6.2. When given the goal task and the initial state of the world, the HTN planner seeks an admissible sequence of actions by recursively decomposing tasks into sub-tasks until all its tasks are actions.

Planner 9 plans partially-ordered graphs of tasks using forward decomposition, as in [40, p. 243]. It keeps track of each possible decomposition in a different search node. Planner 9 starts planning with a single node containing the goal task and the initial state of the world. When visiting a node, Planner 9 iterates through all the tasks that have no predecessor. If the task is an action, it applies this action to the current state of the world and stores the action as part of the plan. Otherwise, Planner 9 instantiates the different possible decompositions of the task. This process goes on until there are no more nodes left or until Planner 9 has found a node with no more tasks to decompose.

The state of the world consists of a set of n-ary relations over a set of values. The application of an action affects these relations. The values represent things from the real world, like rooms or robots in Figure 6.2. In the latter, the `move` action will update the `isIn` relation between the robots and the rooms. The planner creates variables when decomposing tasks. For example in Figure 6.2, the `goto` task can be decomposed using the *fire on the way* alternative. This decomposition introduces new variables: the extinguisher to use and the room where the extinguisher is located. The decomposition has preconditions over these variables that the state of the world must satisfy. For instance, the extinguisher must be located in an accessible room. When Planner 9 decomposes a task, it performs *lifting*: it accumulates its preconditions for delayed check. Planner 9 assigns a value to a variable only when an action changes a relation this variable appears in. For every variable assignations allowed by the accumulated preconditions, Planner 9 updates the state and creates a new search node. To do so in an efficient way, it assigns values using DPLL [21]. The use of lifting is one of the improvements of Planner 9 over the basic HTN algorithm. It results in fewer search nodes and more processing per node, which is desirable for parallelisation.

Planner 9 chooses the node to visit by selecting the least expensive one using A* [48]. In its basic configuration, as cost Planner 9 adds the total cost of the decomposition so far (path cost in A*) and, in general, the

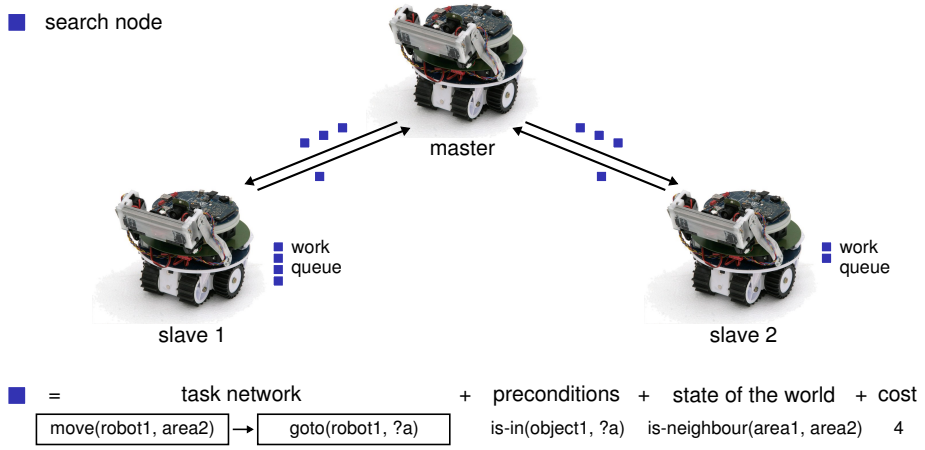


Figure 6.3 Planner 9 balances the load of the different robots by exchanging search nodes over the network.

number of remaining tasks to be decomposed (heuristic cost in A*), see Proof of Theorem 6.1. One advantage of A* with respect to a depth-first search is to allow free recursions in the definition of the planning domain. This is useful in robotics because real-world problems are often expressed in a recursive way, like in Figure 6.2. Moreover, A* finds optimal plans:

Theorem 6.1. *In its basic configuration, Planner 9 always finds the shortest sequence of actions to implement a task.*

Proof. A* is optimal if the heuristic function is admissible, that is, if this function never overestimates the distance to the goal. Let us consider that the cost of an action is ≥ 1. With no loss of generality, with the exception of forbidding free actions, we can rescale the cost of all actions to meet this condition. Thus, the total cost of the decomposition so far, the path cost, is $\geq n$ for a current partial plan containing n actions. If the HTN domain is such that every task will be decomposed into one or more actions, then the number of remaining tasks to be decomposed is an admissible heuristic. If the HTN domain contains tasks that can be decomposed into no action, then the only admissible heuristic is a function always returning 0. In that case, A* reduces to Dijkstra's algorithm [24], which is optimal as well. □

As each node is independent, Planner 9 can distribute the A* search over the network using a master/slave architecture (see Figure 6.3). The slaves run the algorithm described in the previous paragraphs and report periodically the cost of their cheapest nodes to the master. When the cost differences between slaves are significant, the master requests nodes

from the slave with the lowest cost. It then transfers them to the slave with the highest cost. The same robot can be both master and slave, as the master does not require a lot of computational power. As our results show, this load-balancing algorithm is simple yet efficient. Slave robots can appear on and disappear from the network dynamically, for instance when they boot or because their batteries are discharged. The master discovers available slaves dynamically using the Zeroconf protocol [1], which is based on broadcast announcing. Thanks to it, Planner 9 does not require a central registry and can use any robot available on the local network.

Planner 9 is implemented in C++, is open source [2], and runs on embedded Linux. It only depends on the C++ standard library, the boost library [3], and Qt Embedded [4], which are all open source. Planner 9 is thus easy to embed in any robot running a modern Linux board.

6.3 Materials and methods

To measure the performances of Planner 9, we have developed a small search and rescue scenario (Figure 6.4). In this fictional scenario, groups of robots are dropped into a damaged building and must bring a medical kit to humans trapped in a room. Multiple fires block the ways; robots can use fire extinguishers to put them out, but an extinguisher can put out only one fire. This is a typical situation where reactive controllers would fail because there are not enough extinguishers directly available to put out all fires. The robots must use the right extinguishers on the right fire in the right order and thus need a plan (Figure 6.5). This scenario is representative of complex tasks that will require a planner. Here we suppose that the robots know the locations of fires, extinguishers, objects and people. To discover them in a real-world situation, we can imagine to deploy exploration robots that would map the environment.

We perform two types of experiments with different aims:

– First, we measure the scalability of Planner 9 on real marXbot robots by inputing our fictional scenario to the planner. The robots, beside powering up their main computer, rest idle during this experiment.

1. `http://www.zeroconf.org/`

2. One can access the source tree using git; to retrieve it, type: `git clone git://gitorious.org/planner9/planner9.git`. We performed the simulation experiments using revision `a501cfd704e4c759a0d83ea27dfcc85be53929ef` and the real-robot ones using revision `34f4ea4e8d1b17efd610b8eab636e4014f7a8d45`.

3. `http://www.boost.org`

4. `http://www.qtsoftware.com/products/`

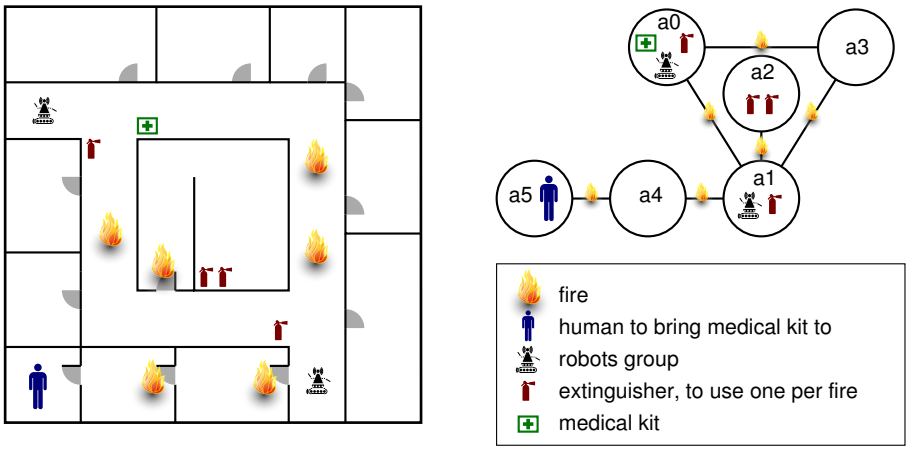


Figure 6.4 Search and rescue scenario (left) with abstract representation (right). The robots are dropped into a damaged building. They must bring a medical kit to the humans trapped in the lower left room. To do so, they must extinguish the fires in the right order; otherwise they would fail the rescue operation as there are not enough extinguishers readily available to put out all fires.

```
take(r0, e0)                    a0, a1, a2, a3, a4, a5 : rooms
extinguish(a1, a0, r0, e0)
take(r1, e1)              r0, r1 : groups of robots initially
extinguish(a2, a1, r1, e1)          in a0 and a1 respectively
move(a2, r0)
take(r0, e2a)               e0, e1, e2a, e2b : extinguishers
move(a1, r0)                 initially in a0, a1, a2, and a2
extinguish(a4, a1, r0, e2a)                      respectively
move(a2, r0)
take(r0, e2b)              o0 : medical kit, initially in a0
move(a4, r0)
extinguish(a5, a4, r0, e2b)
move(a0, r1)
take(r1, o0)
move(a5, r1)
drop(r1, o0)
```

Figure 6.5 Solution plan for the scenario presented in Figure 6.4. `take` let a group of robots pick up an object. `extinguish` puts out a fire between two rooms. `move` displaces a robot group to a room. `drop` puts down the object the robots hold.

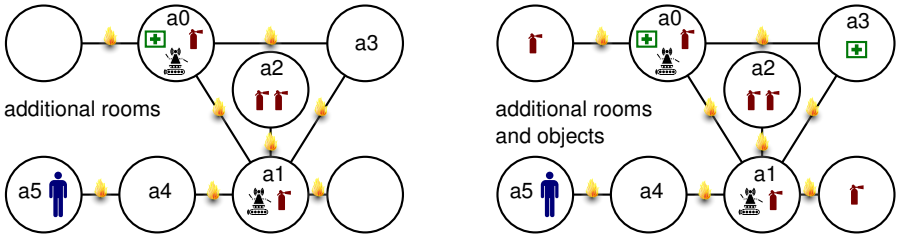


Figure 6.6 Scenarios with additional elements useless to the robots. These elements add more branches to the search tree without affecting the solution.

They use their Wi-Fi adapter to communicate. A desktop computer acts as the master. We vary the number of robots from 1 to 7. We perform 100 runs for each point, and show the average and the standard deviation.

– Second, we compare Planner 9 with JSHOP2 [5] [76], a free and an often referenced HTN planner implemented in Java. As our robots are built around an ARM processor, for which there is no Java runtime environment with an efficient just-in-time compiler, we do this comparison in simulation. We detail the simulation methodology later in this section. To explore the scalability of Planner 9 and JSHOP2, we vary the difficulty of the planning by adding elements to the world that are useless to the robots, as shown in Figure 6.6. We first add two empty rooms, then we add extinguishers in these rooms and a medical kit in a3. These elements add more branches to the search tree without affecting the solution. For each case, we perform 100 runs using 1 and 7 slaves. As JSHOP2 is a depth-first search planner, we add bookkeeping actions that prevent infinite recursions. We do not count these actions when comparing plans' sizes. We subtract the startup time of JSHOP2 (time to decompose an empty task) to put it on an equal basis with Planner 9.

To simulate the robots, we compile, run and benchmark Planner 9 on a real robot and on a desktop computer. We then compute the performance ratio, in term of the number of nodes processed per second, between these two. In the simulation, we use *cpulimit* [6] to only allow as much processing power as available on the robot, which corresponds to 5 % of an Intel Core2 at 2.83 GHz. Thanks to *ulimit* [7], we limit the memory to 100 MB, which is the amount available on the robot. Finally, we divide the

5. `http://sourceforge.net/projects/shop`
6. `http://cpulimit.sourceforge.net/`
7. `http://www.linuxhowtos.org/TipsandTricks/ulimit.htm`

available simulated Wi-Fi bandwidth (19 Mbps using IEEE 802.11g) by the number of slaves and limit the network bandwidth using *trickle*[8]. This way, our measurements are representative of the expected performances on the robots. To prevent the use of fast inter-process communications on the simulation computer, we run multiple slave processes of the planning algorithm on this computer while another computer runs the master control programme. That way, we ensure that data will go through the network. This approach works as long as all slaves plan for a long duration. When we increase the number of slaves, the time to find a solution decreases which is a problem. Indeed, cpulimit and trickle impose limitations by pausing and restarting execution and data transmission. They interfere with our measurements when slaves find solutions in the same order of magnitude of time as these tools operate. We have observed such interferences starting from 8 slaves. Therefore, we limit our measurements to 7 slaves.

6.4 Results

Figure 6.7 shows the measures of the scalability of Planner 9 on the real robots. The first plot shows that the performances scale well with the number of slaves. The speedup, which is the planning time using n slaves divided by the time using 1 slave, is excellent up to 4 slaves. Then it drops a bit but is still very good, as the time to find a plan using 7 robots is 6 times smaller than using only 1 robot. We see that the average number of nodes visited is stable, even if the standard deviation grows. This means that our node distribution algorithm, while being simple, does a good job. While performing these experiments, we noticed that the performances of the wireless network was slow and not very constant. We think that this explains the relatively bad performances with 6 slaves. Even if we conducted 100 runs, these were done in the same period, and thus were not independent, when conditioned over the quality of the network. We think that the major reasons for these inconsistencies are the bad quality of the Linux driver for the Wi-Fi adapter and the transient heavy use of the Wi-Fi by the other users. When operating with a well-supported hardware and in a less bloated Wi-Fi environment, we expect the performances to be slightly better.

Figure 6.8 shows the measures of the scalability of Planner 9 in the simulation. We conducted these runs to compare their results with JSHOP2.

8. `http://monkey.org/~marius/pages/?page=trickle`

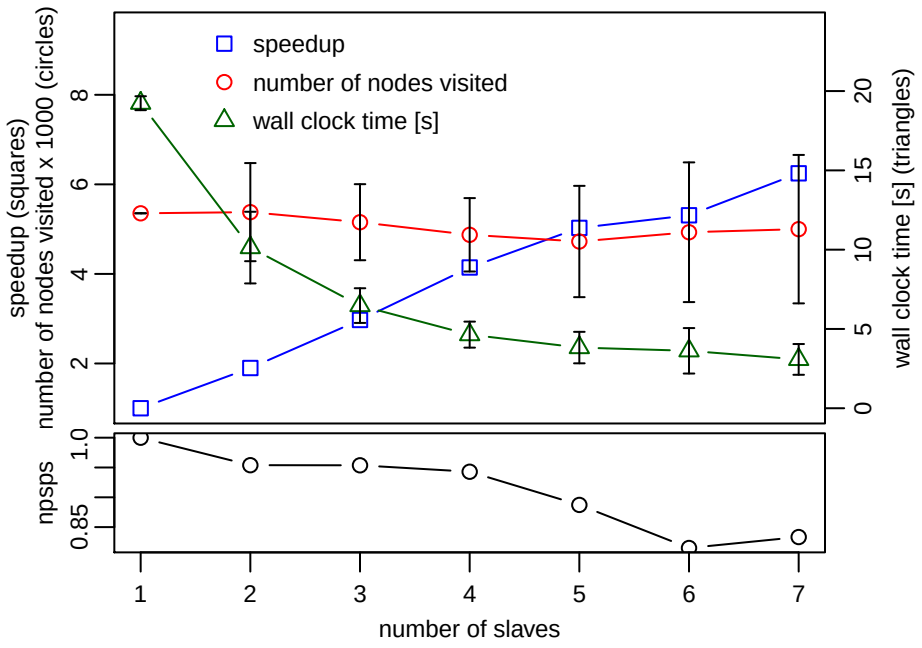


Figure 6.7 The scalability of Planner 9 with respect to the number of slaves, on real robots. The scenario is shown in Figure 6.4. The error bars show the standard deviation over 100 runs. In the second plot, *npsps* represents the number of nodes visited per second per slave, renormalised.

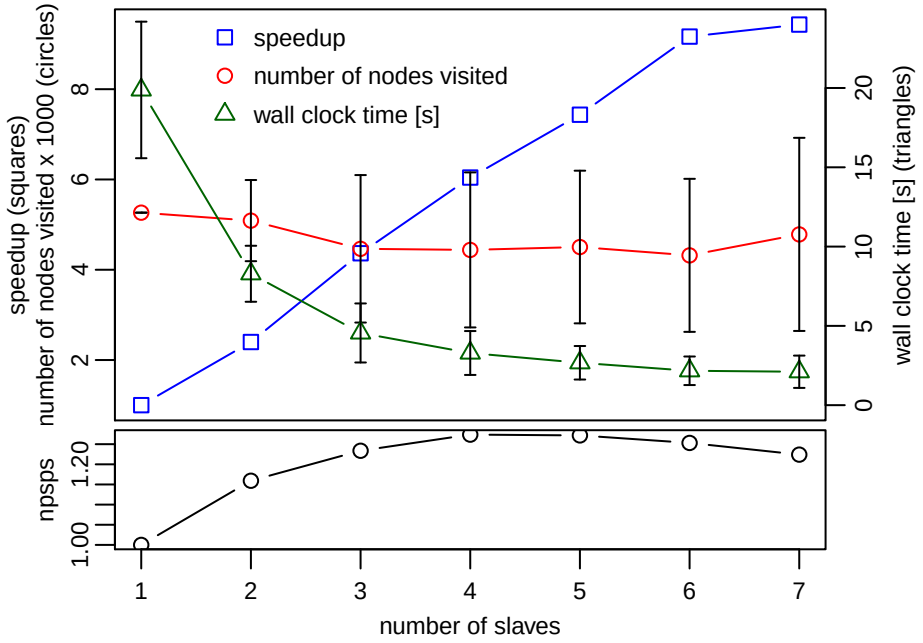


Figure 6.8 The scalability of Planner 9 with respect to the number of slaves, in simulation. The scenario is shown in Figure 6.4. The error bars show the standard deviation over 100 runs. In the second plot, *npsps* represents the number of nodes visited per second per slave, renormalised.

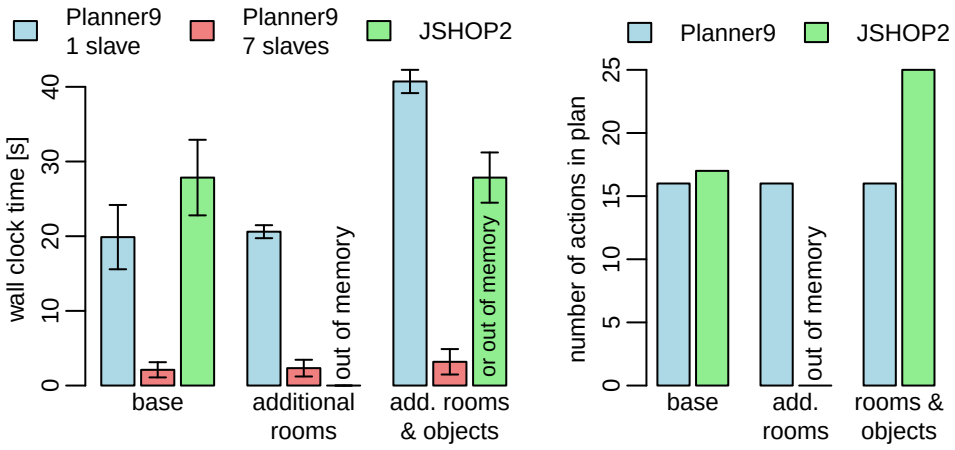


Figure 6.9 The scalability of Planner 9 with respect to the complexity of the environment. The scenario are shown in Figure 6.6. The error bars show the standard deviation over 100 runs. JSHOP2 fails to find a plan when we add rooms, but sometimes succeeds when we further add objects, as we explain in the main text.

The first plot shows that the performances scale nicely with the number of slaves and that the speedup is even super linear. We analyse this using the second plot, which shows the number of nodes visited per second per slave, renormalised. When we increase the number of slaves, each can process more nodes per second, that is, the processor appears to be faster. The cause of the super linearity is a combination of the structure of the problem [89] and a memory cache effect. Indeed, by distributing the planning, each slave holds fewer search nodes in memory. Saving memory reduces the strain on the processor's cache and on the memory allocator, which increases the performance. This super-linearity property means that Planner 9 can exploit optimally multi-core processors. When the number of slaves grows further, the performance deteriorates. We attribute this effect to the load balancing, which increases the average time to find a solution because it moves low-cost nodes around.

Figure 6.9 shows the measures of the ability of Planner 9 and JSHOP2 to cope with environments of increasing difficulty. On the basic environment, Planner 9 with 1 slave is faster than JSHOP2; using 7 slaves, it is one order of magnitude faster. Moreover, Planner 9 uses both groups of robots while JSHOP2 extinguishes all fires with `r0`, which adds one more move action to the plan. Adding two empty rooms does not disturb Planner 9 much, but JSHOP2 cannot cope: it exhausts its memory before finding any solution, even if we give it 2 GB instead of the 100 MB available on the robot. When we add useless objects in the empty rooms, the planning time of Planner 9 increases but does not explode. In this environment, JSHOP2 either finds a solution quickly or exhausts its memory, depending on the order of appearance of the new elements in its initial state of the world. If we declare the new elements (rooms and objects) before the basic environment, JSHOP2 finds a solution. If we declare them afterwards, it fails. Moreover, when it finds a solution it is a long plan with a lot of useless actions, such as using one extinguisher to access a room to get another one. These results show that Planner 9 scales well with the complexity of the environment, compared to JSHOP2. We attribute these excellent performances to the lifting which keeps different value assignments [9] in a single node by abstracting them using a variable. Planner 9 assigns a value later, when it has collected more constraints on this variable. On the contrary, JSHOP2 assigns values early and performs a depth-first search. If by chance the first search branches lead to the solution, JSHOP2 finds it quickly. But otherwise, it must explore an exponentially large number of branches

9. a value assignment corresponds to choosing a constant from the world, such as `r0` or `a1`

before finding a solution.

6.5 Lessons learnt and future works

As we mentioned in the introduction to this chapter, prior to developing Planner 9 we have tested several planners from the literature and found that none of them was working well on a robotic scenario. The underlying reason is that these planners have been developed and tested on a small set of benchmarks, and though the theoretical foundations of the planners are sound, their implementations are highly biased towards these benchmarks. This teaches us that, at the practical level, expecting plug and play with cognitive bricks is not realistic. At the theoretical level, we notice that planning algorithms are most of the time described in the form of non-deterministic algorithms [35]. While this formalism allows a compact description of complex algorithms, it omits a lot of information about how the choices are taken, in which order, etc. Thus the performance profile on a certain domain will vary a lot given different implementations, as shown by the comparison between JSHOP2 and Planner 9. In the same direction, our comparison results between simulation and real robots show that the physical nature of the underlying IT infrastructure does affect the performance profile of the planning algorithm.

Planner 9 is currently a basic HTN planner that does not provide any extension such as probabilistic planning or conditional actions. The rationale is that planning real-world scenarios is computationally intensive and that we want to tackle tasks of some complexity, so we avoid any extension that inflates the search space until such extension is required for the application. Moreover, we can make good use of the availability of multiple robots to reduce the uncertainty of the world perception through concurrent sensing from different locations. Merging these would produce a robust estimation of the state of the world for planning at the symbolic level. With enough robots, we expect this estimation to be good enough not too require probabilities at the level of the planning space. Nevertheless, we can use probabilities in conjunction with learning to guide the search in the plans' space, as demonstrated in Chapter 7. Finally, the parallelisation capabilities of Planner 9 would work with extensions as well so we can implement them should the need arise.

6.6 Conclusion

In this chapter, we have presented Planner 9, a HTN planner that dynamically distributes planning to multiple processing units. This mechanism works with Wi-Fi-enabled mobile robots by considering them as a computer cluster but also exploits at best multi-core workstation processors. To enable this distribution, we have enhanced the HTN algorithm with an A* search and with the lifting of the tasks' preconditions. Previous works have explored how to distribute the decomposition of a particular subtask, but Planner 9 distributes the processing of any task and thus benefits from all available computational power. Albeit simple, this mechanism is efficient as our results show. When compared to JSHOP2, a milestone in HTN research, Planner 9 always finds optimal plans; while JSHOP2 only sometimes finds plans, and they are sub-optimal. This validates Hypothesis 6.1 (p. 83).

Thanks to its excellent performances, Planner 9 brings planning to miniature mobile robots, which improves the state of the art of intelligent applications on these platforms, as shown in Chapter 7. As a contribution to the fundamental hypothesis of this thesis, Hypothesis 1.1 (p. 2), Planner 9 shows that the constraints stemming from integration, in this case the limited available computational power and the requirement to distribute the processing, modify the planning algorithm itself, and in this case lead to an improvement over the state of the art.

Chapter 7

Autonomous construction

In this chapter, we present an application that combines the results of the previous chapters (Figure 7.1). In this application, a marXbot explores its environment and builds structures in it, following orders from a human. We already presented the environment at the beginning of Chapter 5. To build a structure, a tower in this experiment, the robot must pile up 3 resources. However, 3 resources might not be readily available, and the robot might have to harvest them from remote areas. Yet to access a remote area, the robot must fill the valleys to build bridges between its current location and the area. Building a bridge requires 2 resources, thus the robot might have to consume its existing resources to build the bridge. To do so the robot must devise a careful course of action not to squander its resources by accessing useless areas. At the start of the experiment, the robot does not know the environment and ignores the number and the location of the resources. This application is challenging for robotics

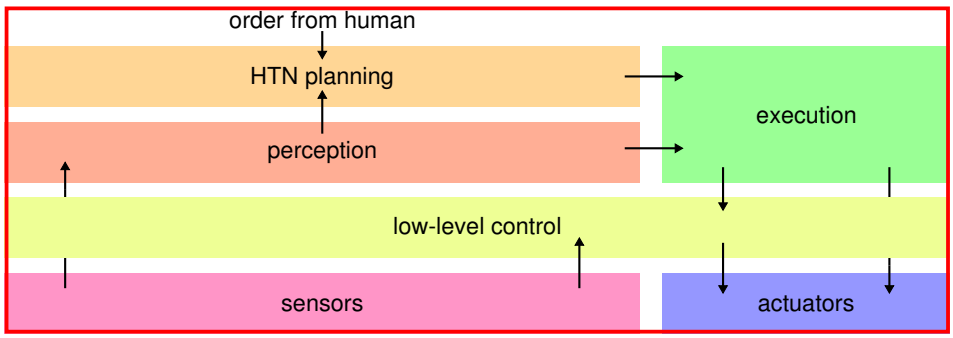


Figure 7.1 The bloc scheme of the autonomous construction application. We combine the results of the previous chapters and add an execution engine.

because it mixes complex low-level actions, such as precisely piling up cubes, with a high-level cognitive behaviour, reasoning on a course of action to avoid situations that prevent the completion of the structure. Yet we will show that by doing the integration properly, we can meet these challenges:

Hypothesis 7.1. *Autonomous construction of three-dimensional structures in an unknown environment with scarce resources is feasible with a miniature mobile robot with imperfect sensors and actuators. Integration plays a key role and is critical for success.*

7.1 Related work

Construction has attracted the interest of roboticists since long, both because it is challenging for robots and because it is dangerous for humans. As construction implies close and complex interactions with the environment, in this survey we will only consider works performed with real robots. Indeed, as stated by [5], "... the complexity of interactions available for exploitation in the real world cannot be matched by any practical simulation environment". While the performances and the accuracy of physics-based simulators have progressed much since this paper was written (1994), the modelling of the interactions that occur in a construction process requires a lot of work, and thus conducting experiments directly in reality is more efficient.

In the early 1990s, researchers in the building industry proposed fully automated workflows to build complete buildings [98]. More pragmatically, roboticists at that time achieved results with real robots employing simple building blocks on flat surfaces. These works are based on the *stigmergic* property of construction; that is, a robot can gain information by looking at what is already present in the environment [5]. This property is the base of collective construction in social insects, like termites and ants [36]. By using stigmergy, researchers managed to build heaps [5], clear areas [81] and build rough walls [69,103]. The construction processes presented in these works require specially crafted building blocks, and only result in simple and approximate structures. Recent works improved the precision and the quality of the constructions by embedding information into the building blocks themselves, by colouring them [114] or even by integrating RFID tags inside them [116,117]. While this indeed improves the extent of what the robots can build, this requires even more complex and handcrafted building blocks. Moreover, all the works surveyed so

far employ multiple simple robots to build simple structures with limited coordination. While this is interesting, in particular in the light of understanding the behaviour of social insects, these works do not allow the construction of complex structures at precise locations. We can retain from this line of research that because it stores information in the world itself, stigmergy provides scalable coordination between the robots and reduces the need for complex positionning. We will exploit this property by doing visual servoing when grasping resources and building structures.

Recently, a second line of research has taken a more engineering perspective and aims at more complex structures. An early work in this direction [37] proposes a robot that aids a human operator to build structures, but is not autonomous in itself. A more recent work from JPL [105] demonstrates two robots assembling beams to build structures. Each robot has a stereoscopic camera, a force sensor to feel the actions and to coordinate with the other robot, and a Wi-Fi connection. Robots detect objects using explicit visual markers, and they employ a multi-layer control strategy. These layers range from low-level reactive behaviours to high-level task planning [55]. NASA hopes to deploy such complex construction systems in outer space [104]. The capabilities of these robots—in particular their ability to build complex structures and to coordinate each other—are impressive. Yet they still rely on precisely engineered beams and well-visible markers, and no reasoning algorithm has been demonstrated so far. Moreover, resources were always readily available.

Our work tackles the problem of autonomous construction of vertical structures with a single robot, in an unknown environment where resources are scarce. We solve the construction problem in this context by integrating into a single system both low-level reactive behaviours and high-level cognition, in this case HTN planning. By doing so, we demonstrate a system that is beyond the state of the art in robotic autonomous construction.

7.2 Experimental setup

The experimental setup is presented in Figure 5.2 (p. 65), at the beginning of Chapter 5. The environment consists of flat areas separated by walls and valleys, and resources that the robot can drop into valleys to build bridges or can pile up to build towers.

The resources are cubes of expanded polystyrene of 6 cm. The cubes have tiny ferromagnetic plates at the bottom of their sides to allow the marXbot to grasp them with its magnetic manipulator. Moreover, the

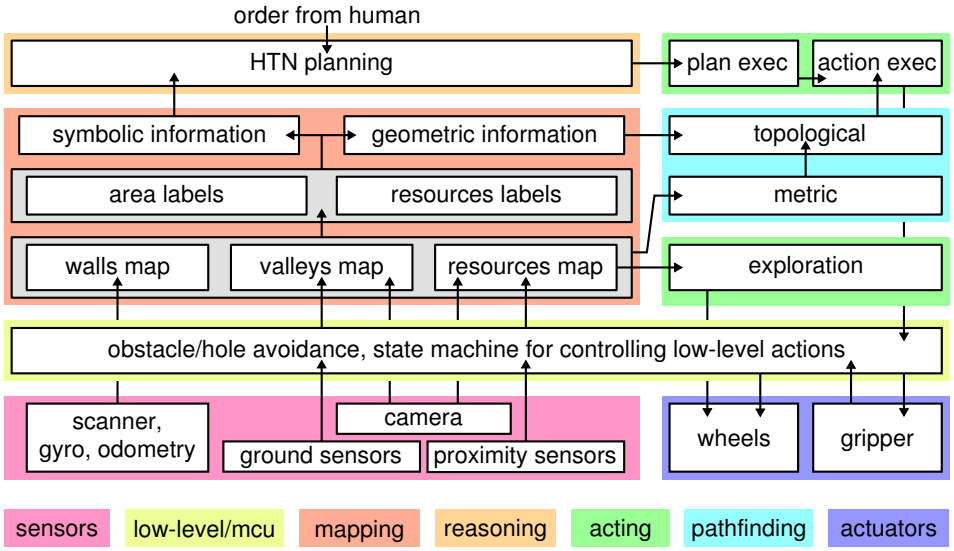


Figure 7.2 The system architecture for the autonomous construction application.

cubes have small magnets at their bottoms and little ferromagnetic plates on their tops. This allows the cubes to align with one other when stacked. The marXbot can employ the cubes either to fill valleys to create passages towards remote areas, or it can pile them up to build structures.

7.3 Overview of the control programme

We use the marXbot in the configuration presented in Figure 2.4 (p. 12). This configuration includes the base of the robot, the magnetic manipulator, the rotating distance scanner and the main embedded computer with the front camera. Figure 7.2 shows the system architecture and the block scheme of the control programme.

In this autonomous construction scenario, the robot performs two main activities: exploration and plan execution. When the experiment starts, the robot begins exploring. Exploration allows the robot to build a dual symbolic-geometric representation of the environment, as described in Chapter 5. The exploration behaviour is the same as the one described in Section 5.5 (p. 74). The resource and the hole maps are only updated when the robot is exploring. When a human gives an order, the robot uses its internal representation to build the initial state of the planning problem. The robot then performs the planning using Planner 9, which is described in Chapter 6. If the planning succeeds, the robot stops exploring

action	description
`move`(d, r)	Move to a given position in the destination area d.
`take`(res)	Move to the front of the resource *res* and take it.
`fill1`(d, s)	Drop a first resource to start building a bridge between the areas s and d.
`fill2`(d, s)	Drop a second resource to finish building the bridge between the areas s and d.
`build1`(d)	Put a first resource to begin building a tower in area d at a given position.
`build2`(d)	Put a second resource on top of the first to build the tower in area d at a given position.
`build3`(d)	Put a third resource on top of the second to build the tower in area d at a given position.

Table 7.1 HTN actions that the robot can perform in the real world.

and executes the plan; otherwise it continues exploring. After executing a plan, the robot explores again.

The execution of the plan consists in executing each action sequentially. An action in the HTN planning sense corresponds to a low-level behaviour implemented in ASEBA. Table 7.1 lists the available actions in the autonomous construction domain, while Section 7.4 details the corresponding low-level behaviours. Prior to executing an action, the robot moves to a specific position in the relevant area. To do so, it uses a two-layer pathfinder algorithm, as explained in Section 7.5.

7.4 Low-level behaviours

We implement the low-level behaviours corresponding to HTN actions using ASEBA (see Chapter 3). The high-level control starts the execution of a low-level behaviour using a specific event, and the low-level behaviour informs the high-level control of its execution result (success or failure) through an event as well.

7.4.1 `move`

This behaviour consists in moving to a given position in a destination area. The area is available in the symbolic part of the representation of the world, and the position in the geometric part.

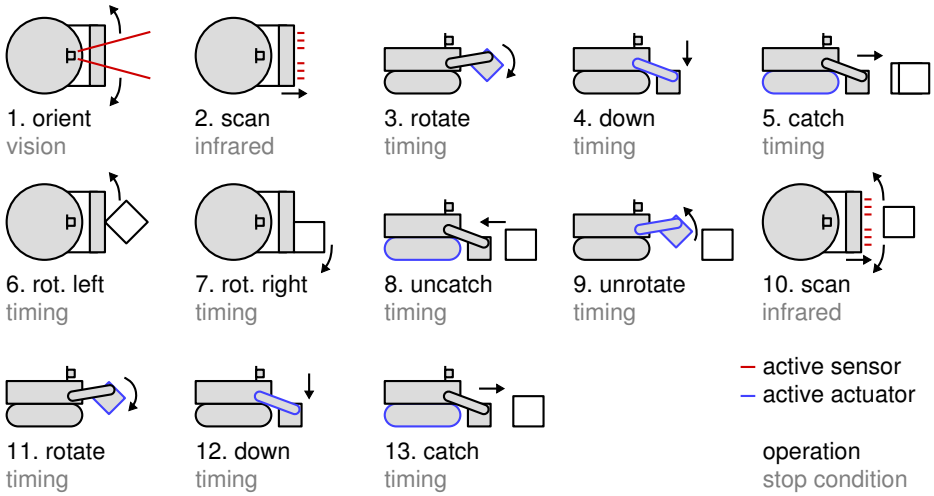


Figure 7.3 The movement sequence to grasp a cube with precise alignment.

7.4.2 `take`

This behaviour consists in accurately grasping a cube; Figure 7.3 shows its movement sequence. The precision is critical because a misaligned cube would add errors to all subsequent operations. Based on preliminary experiments, we have chosen to grasp cubes in two steps to ensure their precise alignment on the manipulator. After an initial orientation using the camera, the robot goes forward and grasps the cube. It then rotates leftward and rightward to ensure a firm grip. At that point the cube is surely grasped, but certainly misaligned. Thus the robot ungrasps the cube and moves back. It then moves forward again while using the infrared sensors of the magnetic manipulator to align the cube at the centre of the manipulator. The robot performs this by rotating if the difference between the median right and left infrared sensors is above a threshold. Otherwise the robot goes straight on. The speed of rotation is proportional to the difference, which results in a P controller with a hysteresis. When the robot is close enough to the cube, it grasps it again, and this time the cube is well aligned. We have experimentally found this control policy to produce a precise grasping. To allow scanning for valleys while holding a cube, the magnetic switchable device lies on one side of the manipulator while the infrared sensors lie on another side. Thus between scanning and grasping, the robot must rotate the manipulator. This explains the complexity of the grasping sequence shown in Figure 7.3.

7.4.3 `fill`

This behaviour consists in dropping a cube into a valley to build a bridge to the opposite area. To build a safe passage for the robot, a bridge requires two cubes side by side. The HTN planning domain ensures that this behaviour is only called when the robot is holding a cube. There are two versions of this behaviour, `fill1` and `fill2`; they differ only by the initial placement of the robot. For `fill1`, the robot is initially placed at the right of the centre of the bridge, for `fill2` at the left.

The robot goes towards the valley and scans for it using the infrared proximity sensors of the magnetic manipulator. The robot computes the difference between the median right and left infrared sensors, and rotates accordingly. We limit the rotation speed. This control law improves the orthogonality between the robot and the local side of the valley. When the two infrared sensors see the valley, the robot goes back for a short distance, then stops and drops the cube. The robot must go back because the infrared proximity sensors are placed far from the magnetic switchable device, so the robot must move to drop the cube close to the border of the valley.

7.4.4 `build`

This behaviour consists in building a tower of up to three cubes. The HTN planning domain ensures that this behaviour is only called when the robot is holding a cube. There are three versions of this behaviour, `build1`, `build2` and `build3`; respectively to place the base cube for the tower, to place the middle cube and to place the top cube. Putting the base cube is easy, the robot simple goes forward for a short distance, then stops and disengages its magnetic switchable device. It then slightly lifts its manipulator to ensure that the cube is well detached, and goes back for a short distance. Putting the middle and the top cubes first requires an orientation using the camera. Then the robot raises its manipulator, goes forward and scans for the previous cube using the infrared proximity sensors of its magnetic manipulator. If the sum of the intensities of the middle sensors is above a threshold, the robot stops and disengages its magnetic switchable device. As the cubes have small magnets at their bottoms and little ferromagnetic plates on their tops, the new cube self-aligns and self-assembles with the existing cube. Finally, the robot goes back for a short distance. The difference between adding the middle and the top cube lies in the height to which the robot raises its manipulator.

7.5 Pathfinder

The pathfinder is composed of two layers. The high-level layer allows the robot to go from an area to another. To find this path, the robot looks up the source and the destination areas into the bridge table. Because it suffices for our application, we have currently only implemented direct lookup, when there exists a bridge between the two areas. Extending this with a search in the bridge graph to find the path for two areas that are not directly connected is easy. The result of the high-level pathfinding layer is a sequence of points. Note that the HTN planning domain enforces that the robot builds a bridge prior to crossing a valley.

The low-level layer finds a path between two of these points. This may correspond to a path within an area or to the crossing of a bridge. This path avoids walls and unexplored areas. It also avoids resources if possible. This pathfinder is based on the work of Philippsen et al. [84]. The algorithm performs a wavefront propagation with variable propagation speed and a smooth front. We use the variable propagation speed to implement soft constraints, such as allowing the robot to avoid the resources if there is enough room. At the implementation level, we first take the maximum likelihood of the probabilistic wall map to extract the free space. We consider that an unknown location (with a uniform probability distribution) is not accessible. Then, we perform a morphological erosion of this map [46], to ensure that the robot will not hit walls when moving. If the robot is not trying to cross a bridge, we perform equivalent operations for the probabilistic hole and resource maps. Finally we combine these three maps together, to set a very low propagation speed in the holes, a middle one in the resources and a high one in the free space. This ensures that a robot located half over a hole will find a way out of it, and that the robot will avoid resources when possible. The propagation speed within walls is zero, as the robot is not supposed to be inside them. If this happens, the programme triggers an exception, as this is probably the manifestation of a bug.

This two-layer pathfinder does not in general produce optimal global paths. For example, if a path traverses three areas, there might be more than one potential bridge connecting the first two areas of the path, with one closer to the third area than the others. In that case, choosing the close one would lead to a shorter global path than choosing the remote one. To implement such a global optimisation, it would be better to transform the two-layer pathfinder into a single-layer pathfinder based on A* and operating on a grid. However, doing so while retaining the smoothness

property of our low-level pathfinder is difficult. Because of this, we employ the two-layer approach.

7.6 HTN planning

We want the robot to autonomously find a plan to fulfil the human's order. The order can be a move or a build order. For example, the human can order to build a vertical structure, which requires 3 cubes. Imagine that only 2 cubes are readily available but 3 cubes are available at a remote location. In this case, the robot could use these 2 cubes to fill a valley to access the remote location to fetch the 3 cubes. To use these separate elements of knowledge to choose its course of action, given the environment and the goal, the robot uses an HTN planner. We use Planner 9 that we have presented in Chapter 6. The space of possible plans to solve a given construction task is constrained by the HTN *planning domain* for our construction scenario (Figure 7.4). We designed this domain around a HTN task that connects areas togother. Then, the two top-level tasks `moveRobot` and `buildStructure` call this task with the destination and the resource areas as parameters. The preconditions and the forward decomposition of Planner 9 prevent infinite recursions.

When the human gives an order, we directly construct the initial state space from the symbolic representation of the world. Then we call Planner 9, and if it finds a plan, we execute the plan. This consists in sequentially executing the actions.

7.7 Adaptive HTN planning

As the HTN algorithm decomposes a high-level task into a sequence of low-level actions, in general there are several admissible sequences of actions that can achieve a given task. The planning algorithm presented in Section 7.6 finds the shortest plan if every task is decomposed into one or more action. It is not the case of the autonomous construction domain shown in Figure 7.4, because the `regions-already-connected` alternative for method `connectRegions` results in no action. Yet by moving the check for connected regions to the top-level methods (`moveRobot` and `buildStructure`), and by providing an alternative for the cases where the regions are always connected, we can translate this domain into one for which Planner 9 is optimal. This will also remove the `setConnected` HTN action that does not correspond to any physical doing by the robot.

Relations

```
unary relations = {robot, region, resource}
binary equivalent relations = {isConnectable, isConnected}
binary relations = {isIn}
```

Actions

```
move(d, r)
  locals:  a
  precond: region(a) ∧ isIn(r, a)
  effects: ¬isIn(r, a) ∧ inIn(r, d)

take(res)
  locals:  a
  precond: resource(res) ∧ region(a) ∧ isIn(res, a)
  effects: ¬isIn(res, a) ∧ ¬resource(res)

fill1(d, s)
  precond: region(d) ∧ region(s)
  effects: ∅

fill2(d, s)
  precond: region(d) ∧ region(s)
  effects: isConnected(d, s)

build1(d)
  precond: region(d)
  effects: ∅

build2(d)
  precond: region(d)
  effects: ∅

build3(d)
  precond: region(d)
  effects: ∅

setConnected(d, s)
  precond: ∅
  effects: isConnected(d, s)
```

Figure 7.4 HTN planning domain for autonomous construction

Methods

```
regions-already-connected(d, s)
  task: connectRegions(d, s)
  precond: region(d) ∧ region(s) ∧ isConnected(d, s)
  subtasks: ⟨⟩

fill-valley(d, s)
  task:     connectRegions(d, s)
  locals:   t, rob, roba, res1, res1a, res2, res2a
  precond:  region(d) ∧ region(s) ∧ region(t) ∧
            robot(rob) ∧ region(roba) ∧ isIn(rob, roba) ∧
            resource(res1) ∧ region(res1a) ∧ isIn(res1, res1a) ∧
            resource(res2) ∧ region(res2a) ∧ isIn(res2, res2a) ∧
            ¬equals(res1, res2) ∧ ¬equals(t, s) ∧
            isConnected(s, roba) ∧ isConnected(s, res1a) ∧ isConnected(s, res2a) ∧
            isConnectable(s, t) ∧ ¬isConnected(s, t)
  subtasks: ⟨take(res1), fill1(t, s),
             take(res2), fill2(t, s),
             connectRegions(d, t)⟩

move-robot(d)
  task:     moveRobot(d)
  locals:   s, r
  precond:  region(d) ∧ region(s) ∧ robot(r) ∧ isIn(r, s)
  subtasks: ⟨connectRegions(d, s), move(d, r)⟩

build-structure(d)
  task:     buildStructure(d)
  locals:   rob, roba, res1, res1a, res2, res2a, res3, res3a
  precond:  region(d) ∧
            robot(rob) ∧ region(roba) ∧ isIn(rob, roba) ∧
            resource(res1) ∧ region(res1a) ∧ isIn(res1, res1a) ∧
            resource(res2) ∧ region(res2a) ∧ isIn(res2, res2a) ∧
            resource(res3) ∧ region(res3a) ∧ isIn(res3, res3a) ∧
            ¬equals(res1, res2) ∧ ¬equals(res2, res3)
  subtasks: ⟨connectRegions(d, roba),
             connectRegions(d, res1a), take(res1), build1(d),
             connectRegions(d, res2a), take(res2), build2(d),
             connectRegions(d, res3a), take(res3), build3(d)⟩
```

Figure 7.4 (continued)

Thus Planner 9 will find the plan with the shortest number of actions. We have presented the domain in its non-optimal form because it is more human-readable that way.

However, in the real world, the optimality of a plan does not depend only on the number of actions, but on the actions themselves. With a physical robot actions might fail, and different actions have different failure rates. We are thus not interested in just finding an admissible sequence of actions in the HTN sense, but we want to find the plan that has the most chance of success. However, as the HTN planning domain allows recursions, we cannot simply consider the expected probability of success of the plan, as recursive tasks with actions that have a probability of success of 100 percent would create infinite loops in the HTN algorithm. We could alleviate this problem by limiting the maximum probability of success of any action to be strictly smaller than 1, but then the choice of the bound is arbitrary. We prefer to model this effect by considering that the usefulness of a plan does not only depend on its probability of success, but also on its duration. Several observations of the real world motivate this choice. For instance, the robot consumes energy with time, and thus at a certain moment the robot will have to recharge, which will prevent it from completing its plan. Also, by performing HTN planning we assume a stationary world, in which nothing changes beside the robot. This is not true in reality, and the longer the execution lasts, the more probable a change is. Finally, the robot is a physical device which can break.

To model the dependency on the plan length, we associate to each action type a a corresponding utility $u(a)$, which is a random variable depending on the outcome of the action a. We only consider the type of the action and not the parameters of every instances, because the parameters represent real-world objects such as resources or areas that change from experiment to experiment. We want Planner 9 to find the plan π that has the maximum expected utility $\mathbb{E}[u(\pi)]$. In the general case of HTN decomposition, a plan π is a directed acyclic graph (DAG) of partially ordered actions. However, in this application to autonomous construction, we use a single robot and consider the plan as a sequence of k actions $\pi = \langle a_1, \ldots, a_k \rangle = \boldsymbol{a}_1^k$. We define the utility of a plan π to be the product of the utilities of its actions:

$$u(\pi) = \prod_{i=1}^{k} u(a_i) \tag{7.1}$$

The utility of a plan is a random variable which depends on the outcomes of its actions. Let $\mathrm{R} = \{\mathrm{S}, \mathrm{F}\}$ be the set of possible outcomes (success or

failure). For a plan containing k actions, let R^k be k-ary Cartesian power of the set R, representing the space of possible outcomes. Let $\boldsymbol{r} = \langle r_1, \ldots, r_k \rangle$ iterate over R^k. Thus, by the definition of the mathematical expectation:

$$\mathbb{E}\left[u(\pi)\right] = \sum_{\boldsymbol{r} \subset \mathrm{R}^k} p(\boldsymbol{r}|\pi) u(\pi|\boldsymbol{r}) = \sum_{\boldsymbol{r}} p(\boldsymbol{r}|\pi) \prod_{i=1}^{k} u(a_i|r_i) \tag{7.2}$$

If an action fails, the robot stops the execution of the plan. We thus consider the utility of a failure to be 0. Thus, the product of the utility in 7.2 is non zero if and only if all action executions result in a success. We can thus rewrite 7.2 as:

$$\begin{aligned} \mathbb{E}\left[u(\pi)\right] &= p(\boldsymbol{r} = S|\pi) \prod_{i=1}^{k} u(a_i|r_i = S) \\ &= \prod_{i=1}^{k} \underbrace{p(r_i = S|\boldsymbol{r}_1^{i-1} = S, \pi)}_{\theta(a_i)} \prod_{i=1}^{k} \underbrace{u(a_i|r_i = S)}_{\gamma(a_i)} \end{aligned} \tag{7.3}$$

This equation shows that the expectation of the utility of a plan is the product of the probabilities of success of each successive action multiplied by the product of the utility of each action. The probability of successfully executing an action depends on some of the previous actions performed by the robot. We call these its dependency list. In our domain there is only a single dependency list for a given action. For instance, the action `fill1` depends on its preceding action, which is always `take`. Thus, the probability of success of `fill1` is:

$$\begin{aligned} & p(r_i = \mathrm{S}|\boldsymbol{r}_1^{i-1} = \mathrm{S}, a_i = \texttt{fill1}, \boldsymbol{a}_1^{i-1}) \\ =& p(r_i = \mathrm{S}|\boldsymbol{r}_1^{i-1} = \mathrm{S}, a_i = \texttt{fill1}, a_{i-1} = \texttt{take}, \boldsymbol{a}_1^{i-2}) \\ =& p(r_i = \mathrm{S}|\boldsymbol{r}_1^{i-1} = \mathrm{S}, a_i = \texttt{fill1}, a_{i-1} = \texttt{take}) \end{aligned} \tag{7.4}$$

Knowing that the HTN task decomposition algorithm enforces that `fill1` will always be preceded by `take`, we can define the probability in 7.4 by a parameter $\theta(\texttt{fill1})$. Thus for each action a, the probability that its execution is a success is defined by the parameter $\theta(a)$. In 7.3, we see that the utility $\gamma(a)$ is also a function of the action a. We can define $\hat{\gamma} = \max \gamma$. With no loss of generality, we can enforce $0 < \hat{\gamma} < 1$ by a simple renormalisation. Thus the expectation of the utility of an infinite plan is zero, because both products of 7.3 are < 1. In our domain, we have 7 possible actions and thus 7 values for θ and γ.

As we saw in the introduction to this section, we can transform our HTN domain such that every task is decomposed into one or more actions.

action type	description	dependency list
`move`	move the robot	
`take`	take a resource	
`fill1`	fill a valley, part 1	`take`
`fill2`	fill a valley, part 2	`take,fill1,take`
`build1`	build a structure, part 1	`take`
`build2`	build a structure, part 2	`take,build1,take`
`build3`	build a structure, part 3	`take,build1,take,build2,take`

Table 7.2 The different actions and the preceding actions they depend on, called the dependency list. For every action, the HTN task decomposition algorithm enforces that actions in the dependency list are executed before it. For each action a, the probability of successfully executing it is defined by the parameter $\theta(a)$.

In this domain, we can use the expected utility of a partial plan, $\mathbb{E}\left[u(\boldsymbol{a}_1^i)\right]$, to guide the A* search [48] of Planner 9 such that it always finds the plan that has the largest expected utility:

Theorem 7.1. *Using $-\log \mathbb{E}\left[u(\boldsymbol{a}_1^i)\right]$ as the path cost for a partial plan containing i actions and by using $-n \log \hat{\gamma}$ as the heuristic cost, with n being the number of remaining tasks to be decomposed, Planner 9 finds the plan that has the largest expected utility.*

Proof. A* is optimal if the heuristic function, in this case $-n \log \hat{\gamma}$, is admissible, that is, if this function never overestimates the distance to the goal. Let us consider a node with a partial plan $\boldsymbol{a}_1^i$ and n remaining tasks to be decomposed. If Planner 9 can decompose this node into a plan π, the latter will have at least $i + n$ actions, as every task is decomposed into one or more actions. Let us consider that this plan π has k actions, with $k \geq i + n$ and $m = k - i - n$; its expected utility is:

$$\begin{aligned} \mathbb{E}\left[u(\pi)\right] &= \mathbb{E}\left[u(\boldsymbol{a}_1^i)\right] \underbrace{\mathbb{E}\left[u(\boldsymbol{a}_{i+1}^n)\right]}_{\leq \hat{\gamma}^n} \underbrace{\mathbb{E}\left[u(\boldsymbol{a}_{i+n+1}^k)\right]}_{\leq 1} \\ &\leq \mathbb{E}\left[u(\boldsymbol{a}_1^i)\right] \hat{\gamma}^n \end{aligned} \tag{7.5}$$

By plugging inequality 7.5 into the path cost of the final plan, and noting that log is a monotonic function, we see that:

$$-\log \mathbb{E}\left[u(\pi)\right] \geq -\log \mathbb{E}\left[u(\boldsymbol{a}_1^i)\right] + (-n \log \hat{\gamma}) \tag{7.6}$$

move	build, local res. training	build, remote res. validation
`take(c0)`	`take(c1)`	`take(c0)`
`fill1(a1, a0)`	`build1(a1)`	`fill1(a1, a0)`
`take(c5)`	`take(c2)`	`take(c5)`
`fill2(a1, a0)`	`build2(a1)`	`fill2(a1, a0)`
`move(a1, r0)`	`take(c3)`	`take(c1)`
	`build3(a1)`	`build1(a0)`
		`take(c2)`
		`build2(a0)`
		`take(c3)`
		`build3(a0)`

Table 7.3 Solution plans for the two training tasks and for the validation task

This inequality shows that the path cost of the completed plan is always larger than the path cost of a node plus the heuristic cost. The heuristic function is thus admissible, and A* always finds the plan of minimum cost, that is, the plan of maximum expected utility. □

7.8 Experimental methodology

We test the autonomous construction application in the real-world setup presented in Figure 5.2 (p. 65). To validate the adaptive planning model, we must measure the utility $\gamma(a)$ and the success rate $\theta(a)$ of the different actions. We cannot measure the success rate directly because the actions have dependency lists (see Table 7.2). We thus make indirect measures, by letting the robot perform high-level tasks and by monitoring their outcomes. We define two training tasks and one validation task. Table 7.3 shows the solution plans for these tasks, with constants corresponding to Figure 5.2 (bottom). In all tasks, we initially let the robot explore for a while until it has a stable representation of the environment. This exploration phase lasts 5 to 10 minutes.

The first training task consists of letting the robot move from the small area (the one with two resources) to the large one (the one with three resources). To fulfil this task, the robot must build a bridge. This task provides measurements for the following actions: `take`, `fill1`, `fill2` and `move`. At the beginning of this task, we place the robot in the centre of the small area, facing the opposite area. The second training task consists

task	average duration	success rate
move	100 s	95 %
build, local resources	286 s	95 %
build, remote resources	726 s	80 %

Table 7.4 Average duration and success rate over 10 runs for three different tasks

of building a tower in the large area, using the three resources available in this area. This task provides measurements for the following actions: `take`, `build1`, `build2` and `build3`. At the beginning of this task, we place the robot in the centre of the large area, turning its back on the valley. The validation task consists of building a tower in the small area, starting from this area. Because this area holds only two resources, the robot must first employ them to build a bridge toward the large area, and use the three resources there to build the tower in the small area. This task employs all types of actions, excepted `move`. At the beginning of this task, we place the robot in the centre of the small area, facing the large area.

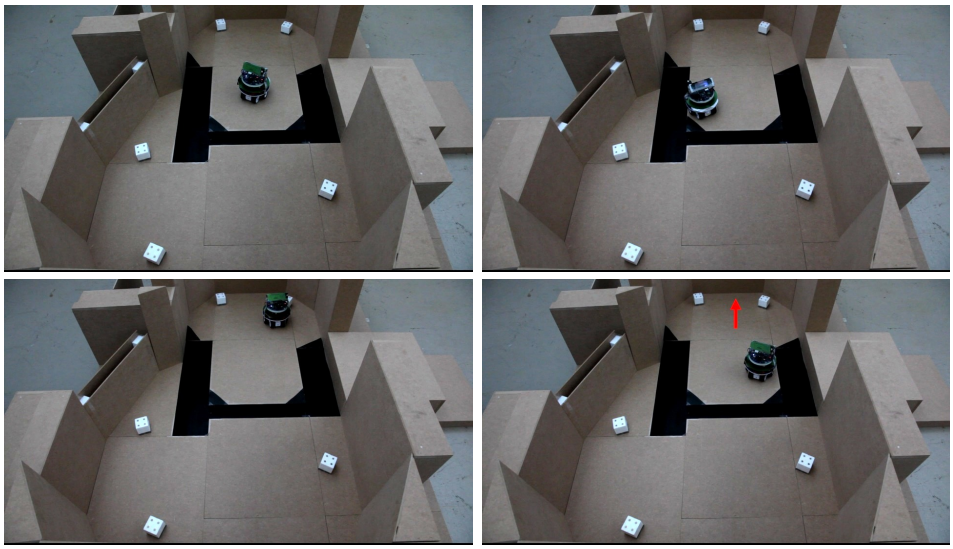
We consider that all actions have the same utility $\gamma(a) = \gamma, \forall a$. This utility is not 1 because events external to the robot controller may prevent the successful completion of the action. In this series of experiments, from times to times the Wi-Fi network fails. We thus estimate the utility factor γ by counting the number of times a network failure interrupts an experimental run. When such an event occurs, we restart the experimental run.

7.9 Results

We have run each task 10 times. Figure 7.5 shows a sequence of images of a successful validation-task run. Table 7.4 shows the success rate and the average duration of the different tasks. We only take successful task executions into account to compute the average duration.

The move task (training) fully succeeded 9 times out of 10 trials. One time the robot dropped the second cube aside instead of into the valley. However, when the robot then moved, it pushed the cube into the valley. As we could interpret this either as a success (finally, the task was a success) or as a failure (the action itself did not result in the expected situation), we decided to consider this as a semi-success, hence the 95 % success rate for the move task. The building task with local resources

The robot explores its environment for a while, then a human orders the construction of a tower:



The robot needs 3 resources to build the tower, but only 2 are readily available; thus the robot must build a bridge to harvest remote resources:

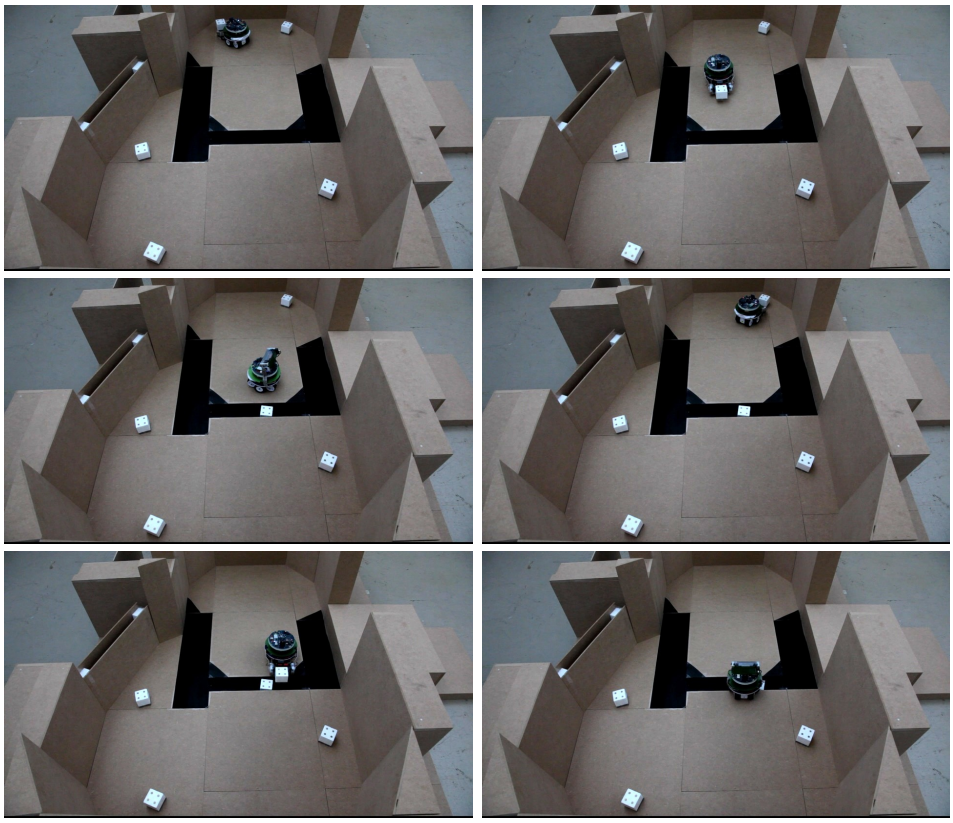


Figure 7.5 Image sequence of a successful construction.

Once the bridge is completed, the robot can harvest the remote resources to build the tower:

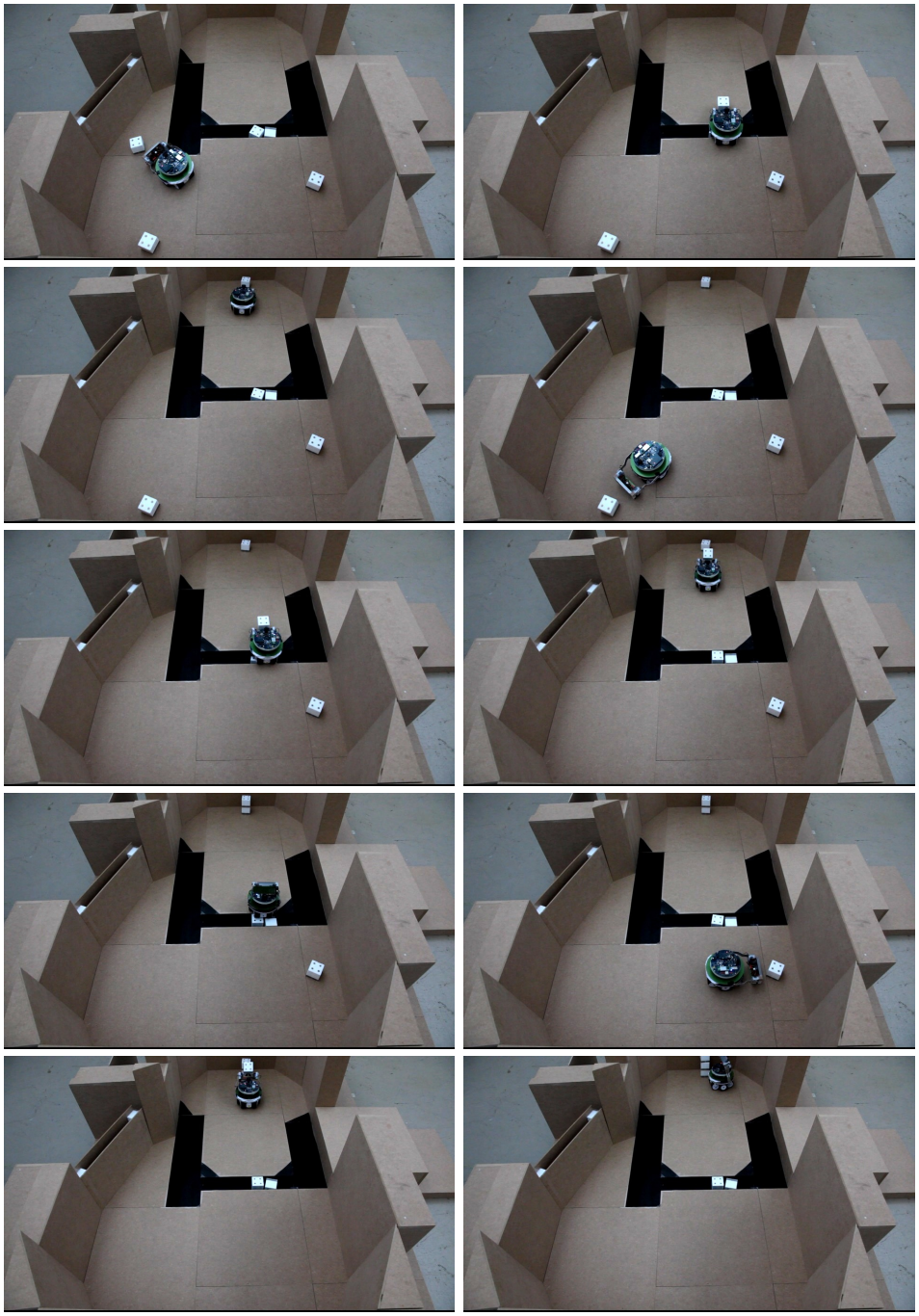


Figure 7.5 (continued)

	action	number of executions	number of successes	rate
training	`move`	10	10	100 %
	`take`	50	50	100 %
	`fill1`	10	10	100 %
	`fill2`	10	9.5	95 %
	`build1`	10	10	100 %
	`build2`	10	10	100 %
	`build3`	10	9.5	95 %
validation	`move`	0	0	undef.
	`take`	45	44	98 %
	`fill1`	10	10	100 %
	`fill2`	10	9	90 %
	`build1`	8	8	100 %
	`build2`	8	8	100 %
	`build3`	8	8	100 %

Table 7.5 Success rate for different actions over 10 runs of two training tasks and one validation task

(training) fully succeeded 9 times out of 10 trials. One time the robot put the third and last cube of the tower half onto the second cube. It thus built a tower but not a straight one. Here we could also interpret this either as a success (the robot built a tower) or as a failure (the tower was imperfect), so we decided to consider this as a semi-success, hence the 95 % success rate for the building task with local resources. The building task with remote resources (validation) fully succeeded 8 times out of 10 trials. One time the robot fell into a valley just before executing `fill2`, because of the imperfection of the world perception. One time the robot failed to take its third resource, because it tried to grasp the resource from its corner, and the gripper alignment procedure was not extensive enough. This results in a 80 % success rate for this validation task. This rate is not as high as in the training tasks because this experiment lasts longer and combines more actions; but also because the perception of the remote area is less precise than the perception of the local area.

While running the training tasks, we have executed 110 actions and experienced 3 network problems. We can thus define the utility of an action by one minus the probability that it is interrupted by such a problem. In our case, this leads to $\gamma = 1 - \frac{3}{110} = 0.973$.

Based on these experimental runs, we extract the average success rate

of each action (Table 7.5). This allows us to compute the probability of success of a plan and its related utility using Equation 7.3. The probability of success of the plan corresponding to building a tower using remote resources is:

$$p(\pi_{\text{build tower using remote resources}}) = \theta^5_{\texttt{take}}\theta_{\texttt{fill1}}\theta_{\texttt{fill2}}\theta_{\texttt{build1}}\theta_{\texttt{build2}}\theta_{\texttt{build3}} \tag{7.7}$$

Using the estimation of the actions' success rate from the training tasks (Figure 7.5, top), this value is 0.90. However, out of 10 validation runs, only 8 were successful, which corresponds to a probability of 0.80. We have a difference because during both training tasks, the robot always interacts with resources close to the places it has physically explored. But in the validation task, the robot has only seen the remote resources with its camera. As we already discussed in Section 5.6 (p. 76), the estimation of the position of these resources is inaccurate. Thus the `take` action is not 100 % successful anymore, which explains the worse performance of the validation task compared to the estimation resulting from the training tasks. Regardless of this discrepancy, we consider these results close enough to prove that our adaptive planning model is successful in guiding the planning along the most best course of action.

Because in this experiment we want to quantify our action model and not our perception model, we always issue task orders only when we consider that the perception of the world is correct. Even in this case we notice that failures, in particular for the `take` action, tend to occur when the perception quality is worse than usual. This confirms that perception quality is critical for action success. Moreover, automating the decision of when to stop exploring and when to start acting is not trivial. A simple solution could be to wait for a certain duration, and then to start acting as soon as Planner 9 finds a successful plan. However, there is a risk that a transient reading, seen as a resource, produces an erroneous grounding and thus an erroneous plan. A more robust solution would be to wait for the symbol grounding output to stabilise; and only to start planning and execution when this output has not changed for a certain duration.

7.10 Discussion

To implement this experiment, we performed a complete vertical integration of the robot control software from the low-level sensor processing and actuator control to the high-level HTN planning. We are fully satisfied with Planner 9, as it is fast enough to produce plans in less than one second.

We are also pleased with aseba, as it allows us to quickly implement complex low-level behaviours such as the one presented in Figure 7.3. Aseba decouples the low-level behaviours from the high-level control, and thus improves the cleanness of the latter. Low-level behaviours employ many state machines, and it might be interesting to add explicit state-machine support to the aseba language. We have observed that the most difficult part of the software stack is the perception subsystem, in particular the symbol grounding part, which is consistent with the literature [47]. The rest of this section discusses some of the lessons we learnt by developing and running this autonomous construction experiment.

The robot control programme fuses data from the infrared proximity sensors and the camera. As the proximity sensors have only a short range and the camera has a long range but a narrow view angle, these two sensors cover different regions of space. As we saw in Section 5.7 (p. 78), this difference leads to a suboptimal perception quality. Another problem lies in the limited range of the robot's sensing. The rotating distance sensor barely sees over 1 m. The camera can see farther, but as we compute the distances by dividing by the pixel's ordinate, a slight difference on that axis introduces major errors. As the tracks of the robot make it shake while moving, this limits the effective range of the camera. Despite a short sight range, the robot needs room when grasping resources and moving around while carrying them. Thus the space of possible experimental scenarios is constrained by these opposite requirements, a small world for a good perception and enough room for action. This fact is a weakness of this robot as a tool to study autonomous construction, but it also shows the importance of the integration and of a global analysis. A good solution must consider the constraints of both the robot and the application.

As we have already mentioned in Section 5.7, timestamping the different incoming data would improve the performance of the perception subsystem. We also think that if the application becomes more complex, if the high-level processes such as symbol grounding, finding paths or planning take a lot of time, these should run in worker threads with low priorities. Building an application with many threads of different priorities is not trivial, in particular because dead-locks can easily appear if the programmer does not consider the information flows with care.

In this application, the robot stops sensing the valleys and the resources once it starts the execution of the plan. The robot does not update its maps following its actions; for example if the robot takes a resource and drops it into a valley, this resource is not removed from the resource map. This behaviour is the simplest and is fine as long as we ask the

robot to perform a single task. It would be a problem if the robot is interrupted in the middle of its task, or if we want the robot to perform further tasks. Indeed, in the first case the robot would need time to update its resource map through new exploration, and in the second case the robot could mistake cubes in valleys for ghost resources. We could alleviate the first problem by updating the probabilistic maps following the robot's actions, and alleviate the second problem by using the knowledge of the presence of cubes in valleys to interpret the camera image. This would complicate the code and throws doubts on the scalability of such quirks. However, implementing them correctly would enable continual planning [11]. Instead of requiring plans that completely solve the problem, we could add to the HTN domain decompositions that perform some actions and then trigger a re-planning. This feature would be necessary for real-world applications, and its dependency on special updates to the probabilistic maps clearly shows the level of intrication between the different subsystems for realistic robotic applications. This corroborates Hypothesis 1.1 (p. 2).

If we analyse the perception process in the light of the needs of the robot's actions, we see that we invest a lot of computational power to build a metric map, yet the robot only uses this map to go to specific positions and to find paths. Because the localisation is not precise, the robot must still align using its camera before taking resources and filling valleys. Moreover, we build probabilistic maps of the resources and the valleys following the assumptions that all cells are independent and that all observations contribute equally to the maps. These assumptions are crude approximations and result in a slow and imprecise perception. All these considerations together lead us to question whether metric maps are the good representation choice for this application. Semantic topological maps [59] might be a better representation model. We can imagine to use a wide-angle camera to detect resources and holes. With a good image-processing algorithm we can extract a lot of information from a single frame, for instance the distances between the visible resources, whether they are in the same area or not, etc. This solution could allow the robot to perceive its environment faster and more robustly.

One could criticise us for using simple building blocks that embed magnets and metal plates to self-align. However, we could object that currently a lot of real-world constructions are made of prefabricated materials, shaped to ease the assembling. Adding magnets to our cubes is not qualitatively different. Moreover, our cubes do not embed fiducials, markers or RFID tags contrary to works such as [105] or [116]. In the

future, the exploration of robotic construction with raw materials is clearly a difficult but promising direction, that presents many scientific and technical challenges.

As we presented in Section 7.8, to find the parameters of the adaptive planning model we estimate the probability of success of each action during a training phase, by running several training tasks. We have seen that this leads to discrepancies with a more complex validation task. Moreover, this supposes a static world, in which the properties of the elements do not change with time. If the world is dynamic in this respect, or if we want to estimate the probability of success online, we can update the parameters during the operations of the robot using an exponential forgetting model. Let us suppose that the robot has executed N times an action a, at times $\boldsymbol{t} = \langle t_1, \ldots, t_N \rangle$. Moreover, let us suppose that these executions resulted in N outcomes $\boldsymbol{r} = \langle r_1, \ldots, r_N \rangle$. The i-th outcome can either be a success, and $r_i = 1$ or a failure, and $r_i = 0$. We can implement our exponential forgetting model to estimate the probability $p(r = \mathrm{S})$ that the action a executed after time t_N results in a success by the following equation in which λ is a time constant, corresponding to the probability that the world model changes:

$$p(r = \mathrm{S}) = \frac{\sum_{i=1}^{N} e^{-\lambda(t_N - t_i)} r_i}{\sum_{i=1}^{N} e^{-\lambda(t_N - t_i)}} \tag{7.8}$$

In this equation we iterate over the whole history to compute the final probability. We can, however, transform this equation into an iterative version that only requires two parameters a and b with $p(r = \mathrm{S}) = \frac{a}{b}$. Initially, $a = r_1$ and $b = 1$. Then, knowing a and b at time i, their values at time $i + 1$ are:

$$\begin{aligned} f &= e^{-\lambda(t_{i+1} - t_i)} \\ a_{i+1} &= f \cdot a_i + r_{i+1} \\ b_{i+1} &= f \cdot b_i + 1 \end{aligned} \tag{7.9}$$

We can interpret this algorithm as a form of reinforcement learning applied to our particular case of HTN planning. Its form is simple as long as we can model the success rate of an action by a single parameter. Figure 7.6 shows a simulation of this learning algorithm on synthetic data. Applying this algorithm to a construction application would improve the adaptivity of the robot. However, the robot must first be able to perfectly sense the result of its actions, and must be able to recover from execution failure at any time. These constraints present major practical difficulties, which

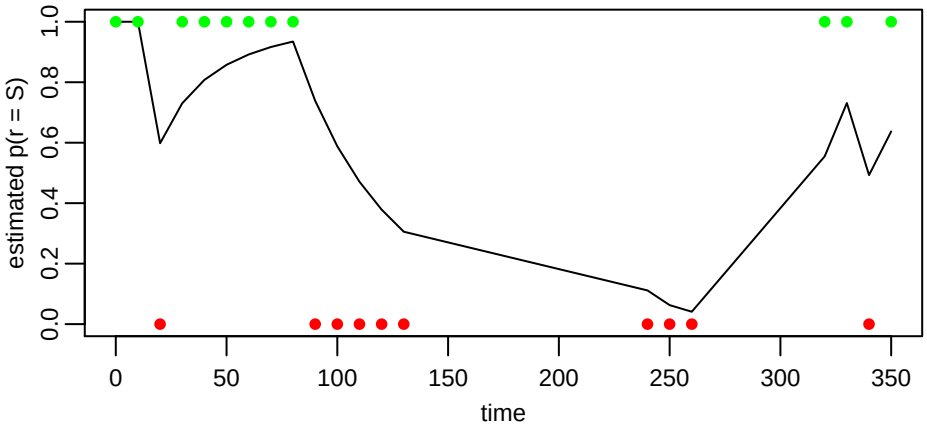


Figure 7.6 Simulation of the learning algorithm for the task's probability of success, on synthetic data. The top green dots indicate a successful action, and the bottom red dots indicate a failed action.

are much harder to solve than implementing a learning scheme. Yet we think that this learning algorithm, albeit simple, is interesting because it springs from an integration process. This shows that simple mechanisms can efficiently increase robot intelligence if they are well integrated.

In general, the choice of the amount of knowledge to hand-code in a robot and what to leave to learning is not easy. If too much is hand-coded, the robot lacks adaptivity; but if too much is left to learning, the robot will learn too slow or not learn at all. In the extreme case when everything is left to learning, the planning process becomes a partially observable Markov decision process. In this case, solving a problem even as modest as our autonomous construction scenario is intractable. We think that our approach of defining the HTN domain by hand and letting the robot learn online the success rate of alternatives is a good compromise for current hardware. However, future developments might shift this balance, probably in the direction of learning, especially if robots can share their acquired knowledge with their peers.

7.11 Conclusion

In this chapter, we have demonstrated an autonomous construction application with scarce resources. We have shown a robot that autonomously reasons about the available resources and that employs them intelligently to build three-dimensional structures. This robot is composed of mechanic and electronic parts that are not at the state of the art, in the sense that

most of these parts are consumer-grade components. In particular, the robot does not embed a laser scanner nor a laptop-level processor. However, by building upon our previous bricks and by integrating them carefully, we managed to create an application that is beyond the state of the art in autonomous construction. This validates Hypothesis 1.2 (p. 2), as our robot demonstrates a more intelligent behaviour than related works. We have also proposed a learning mechanism that allows to take profit of a priori human knowledge while still providing adaptation. This mechanism is computationally lightweight and allows the robot to learn quickly. These features make this mechanism well suited for providing adaptation in realistic scenarios, in which the robot has to perform different tasks and thus cannot redo a single tasks a hundred time to wait for the learning to converge.

Chapter 8

Lessons learnt and future work

Beside the various contributions presented in the previous chapters, the meta contribution of this work lies in the epistemology of mobile robotics. In this chapter, we share some of the thinkings that result of the experience we gained performing this work, and we propose future research directions.

8.0.1 On where to run the controller

A design goal of this work was to build a system whose software can run onboard, as autonomy is a critical feature for mobile robots. However, because we wanted to analyse the behaviour of the robot in real time, during most of the experiments we ran the high-level controller on a remote computer and only ran a simple vision back end on the robot, as shown in Figure 5.6 (p. 75). The problem of allowing autonomous operations by the robot while enabling the developers to analyse its behaviour is always present in robotic research. Some researchers try to alleviate it by running experiments in simulator, but as we already discussed in Section 7.1 (p. 98), this is not realistic enough to develop complex interactions with the world. So we are left with two approaches to solve this problem.

The first is to run everything on the robot, and to use a remote control software such as VNC to access the debug output from a remote computer. We tried this with the VNC server provided with Qt Embedded, but over a Wi-Fi 802.11g link it was unbearably slow. The reason probably lies in the VNC implementation of Qt Embedded, but in general this solution needs a powerful computer on the robot and a fast connection with the remote computer. For instance, Willow Garage proposes to use this development scheme for their PR2 robot.

The other solution is to do minimal processing on the robot, to transmit relevant data and to do processing on a fast remote computer. This is the solution that we have finally chosen, as we process the camera image and send only a bitmap of the relevant part to the remote computer, along events from aseba. We think that this solution is very interesting and should be explored further. For instance, if the vision algorithm implies some feature-extraction step, as most visual slam algorithm do, it would be wise to detect the features on the robot, and then to transmit their descriptors to the remote computer. Moreover, as features tend to stay visible for several frames, it would be interesting to only send their translation vector between images, in a compressed form, and their appearance/disappearance events. Globally, we can consider this method as an extension of the event-based approach of aseba. We could also consider it as an instance of the hcs architecture [2]. We think that exploring this approach in collective robotics would be particularly interesting, because it allows the robots to easily share knowledge about the world. We think that a comparison with other approaches such as swarm robotics (fully distributed) or complete remote control (fully centralised) would also provide a useful contribution to the state of the art.

Autonomy is critical for mobile robots, but as most of these robots work in human environments, they can take advantage of the existing infrastructure. In particular, in modern urban environments, wireless networks allow to deport part of the computation to a remote computer, while maintaining full physical autonomy. We think that this possibility should be taken into consideration when designing robotic applications.

8.0.2 On the design methodology

In most engineering fields, there are methodologies to build complex systems by assembling simple parts. However, robotics lacks successful methodologies. This question was discussed in a recent workshop [13], focusing on what makes robotics different. A common trait of existing engineering methodologies, for instance middleware-based approaches, is that they propose to assemble small black boxes into bigger black boxes. The black boxes cover distinctive functionalities. These methodologies thus consider the parts to be independent and define abstractions to interface them. In robotics, as we saw in Chapter 5, the basic bricks—the sensors, the actuators and the different levels of algorithms—are not independent as they interact through the world in addition to their connections. They are also not orthogonal, which makes the system more

difficult to devise. Moreover, standard machines (such as a production chain) or simple autonomous artefacts (such as an aeroplane autopilot) have only a small information input space and a small repertoire of behaviours. On the contrary, mobile robots have a very large and loosely structured information input space and a large repertoire of behaviours. As a mobile robot must perform different tasks, the information flows in its physical and control structures are not fixed. The flows change constantly as the inputs vary. These variations complicate the design and the analysis of the robot controller.

Furthermore, as some researchers have highlighted [100], specifying the requirements for robotic applications is difficult. Because robots and their interactions with the world are so complex, it is mostly impossible to define fixed requirements. This implies that even if we had a methodology, the design process would still require experimentation and redesign phases. Thus the reality of mobile-robot development is closer to applying a genetic algorithm in a huge non-linear space than to an engineering process: We have basic bricks but combining them correctly is not trivial and we might have to modify them. In this respect, the automated global optimisation that we presented in Chapter 4 makes a lot of sense, as a global optimisation allows to find a Pareto-optimal design given a set of constraints.

Even if there exist no successful system-level methodologies to build robots, there is still room for guiding principles: The design space is large and non-linear, but it is structured and not completely chaotic. Indeed, sub-fields such as motor control or image processing are well understood and there exist templates of successful combinations. Knowing a solution to a problem and facing a similar problem, there is a high probability that a neighbouring solution will apply. We think that this property allows the creation of *design patterns* for robotics [38, 56]. The global optimisation methodology (Chapter 4) is such pattern, as is the distribution of processing through events (Chapter 3). The search for design patterns in robotic development is part of the science of integration. We propose to search them by targeting applications of increasing complexity and versatility, for which the robot hardware and software will get more and more complex. The analysis of the successful approaches will uncover the reliable patterns. This meta-methodology, while being basic, is close to the reality of evolution in which successful patterns are reproduced.

8.0.3 On the generality of published results

Recent works in robotics have provided solutions to various problems that were considered difficult from a theoretical point of view, such as integrating reasoning with motor control in a probabilistic framework [109] or performing real-time slam and world reconstruction with a single camera [77]. These results are impressive and clearly advance the state of the art; however their scope of application is not clearly defined. It is very difficult, starting from a research paper, to know whether the presented solution will be effective in another context or under slightly different conditions. This makes the integration task difficult as one cannot judge the general quality of an approach based on the research papers alone. Morevore, this hinders the technological transfer from research to applications, because one wanting to develop a real-world application is left with many choices one must test separately. To improve this state of affairs, researchers should provide versatile demonstrators that function under different conditions. Moreover, they should thoroughly report the conditions of application of their algorithms, along their limitations and weaknesses.

8.0.4 On human-level robotic intelligence

A common question from the lay public is whether robots will one day be better than humans, and if so, when. Scientifically, this question is ill formulated; but given its philosophical importance, it is still worthy of attention. So let us discuss this question from the side of intelligence and autonomy; as at the level of physical strength, some industrial robots are already far beyond human abilities.

In this work, we managed to implement a complex reasoning ability into a physical robot. Indeed, being able to choose an optimal course of action given limited resources implies intelligence (see Section **??**, p. **??**). However, the task diversity that our robot can achieve is low, and the robot does not cope with failures nicely. On the contrary, humans are versatile and in case of failure, their behaviour degrades gracefully in most cases. This difference shows the enormous work still required to endow robots with human-level abilities. We must improve the precision, the bandwidth and the generality of the hardware devices, but also improve the reusability of the software and the behaviours. We must find methods to cope with failure, and these methods should be adjustable to different scenarios and tasks. Moreover, we should find perception and reasoning algorithms that adapt to various conditions. Learning is certainly part of the solution, and

for instance calibration is a primitive form of learning. However, as we already discussed in Section 7.10 (p. 116), learning requires much help from low-level layers, and their development represents a significant time investment. Finding a way to build these primitives semi-autonomously is certainly one of the challenges ahead of us. The research field of autonomous mental development [115] tackles this problem, however for a real robot all existing approaches still require a huge amount of hand-crafted work. Moreover, these approaches do not propose systems that scale. A better understanding of scalable structural-discovery algorithms is clearly a promising research direction.

We must address all these questions and solve most of them in many different contexts to imagine robotic applications in which robots show human-level intelligence. We think that the best way to proceed is to implement tasks of increasing complexity and versatility. The needs of these tasks will demand intelligence in the robots, and naturally researchers will develop algorithms to provide this intelligence. Versatility is important, because the reality is not a perfectly calibrated room. Intelligence implies adaptation and learning, but the type and the amount of learning is an open question. Modern robots have the ability to share thoughts through Wi-Fi. We think that taking advantage of this to reuse knowledge between robots and experiments is also a good direction to increase robot intelligence. The existing knowledge on the functioning of the brain is rapidly increasing, and we can use this knowledge to aid us in developing algorithms. However, we should not naively copy existing brain structures, as robots have fundamentally different bodies and constraints than animals.

If we look back at the recent history of robotics, during which we experienced the rapid maturation of high-performance SLAM algorithms, the appearance of dense energy storage solutions or the generalisation of inexpensive wireless networks, we see that robotics is improving at a rapid pace. We thus expect further progresses in the near future, but human-level intelligence is still far-off and its reachability still a matter of conjecture.

Chapter 9

Conclusion

9.1 Summary of personal contributions

The mechanics and the electronics of the marXbot and the handbot were developed by Michael Bonani, Philippe Rétornaz, Valentin Longchamp, Daniel Burnier, Florian Vaussard and Tarek Baboura. During the development process, I contributed ideas and feedback, but these were minor.

All our electronic modules are based on the dsPIC family of microcontrollers. I have laid-out the foundations, the design philosophy and started the implementation of Molole, a software library to wrap the hardware devices of these microcontrollers. This library was subsequently developed by Philippe Rétornaz

Based on the idea that we should not erect walls between conceptual and physical layers, I have developed ASEBA. The original idea, the architecture, the language, the compiler, the VM, the IDE and the communication layer are my own work. The integration into the dsPICs, which comprises the low-level control C code and the assembly-optimised native functions are the contributions of Philippe Rétornaz.

I made the implementation of the SLAM algorithm in collaboration with Valentin Longchamp. The idea of globally optimising the error parameters of the motion model and the ray budget allocation by using an evolution strategy is a personal contribution. To measure the ground truth positions of the robot, I have implemented a tracker using an overhead camera and topological fiducials.

Planner 9 is the result of a collaboration with Martin Voelkle. The choice of the algorithm and its adaptation to the collective-robotic context are mostly my contributions, while Martin contributed efficient data structures.

We performed the implementation together.

The magnetic manipulator was first developed by Thierry Barras given my specifications, then after testing it was redesigned by Frédéric Rochat, Patrick Schoeneich and Pierre Noirat. The magnetic switchable device is a development of Frédéric Rochat and Patrick Schoeneich while Philippe Rétornaz programmed the deep low-level control of the manipulator.

The semantic maps and the autonomous construction experiments are my own work. This encompasses the original idea, the construction of the modular environment and the cubes, and all the programming, experimentation and analysis work.

As further contributions to the community, we have developed four open-source software as part of our research work. Molole is available at `http://gna.org/projects/molole/` under the LGPL. Dashel is a communication media abstraction library and is available at `http://gna.org/projects/dashel` under a BSD license. Aseba is available at `http://gna.org/projects/aseba/` under the GPL. Finally, Planner 9 is available at `http://gitorious.org/planner9` under the GPL.

9.2 Outlook

In the beginning of this report, we have formulated Hypothesis 1.1 (p. 2). This hypothesis states that integrating capabilities together in a mobile robot raises new questions that are not present in the parts but that reflect the inner structure of the application and the physics of the world. This hypothesis also states that the integration process modifies the parts themselves, and infers that integration is a science. In Chapter 3, we have seen that the electronic architecture of a robot influences the performance of its control algorithms. We have demonstrated that building a software control architecture knowing the electronic constraints of the robot greatly improves the capabilities of this robot. In Chapter 4, we have shown a global optimisation of a robot's intrinsic parameters and of the allocation of processing resources. Given the constraints of an imprecise but inexpensive sensor coupled with a low-speed processor, the optimisation chose the parameters that allowed to perform SLAM in real time. These parameters reflect the shape of the robot and the structure of the world. In Chapter 5, we have seen that the amount of information one can extract from a sensor might depend on the precision of another sensor. This implies that the sensors cannot be considered separately, but rather that the robot developer should look for a Pareto-optimal sensor allocation policy. In Chapter 6 we have shown that the constraints stemming from

integration can modify an algorithm, because algorithms are often described in general terms. We have shown that this modification leads to an improvement over the state of the art. All these chapters contribute evidences that together validate Hypothesis 1.1, that is, integration is a science.

In Chapter 1, we have also formulated Hypothesis 1.2 (p. 2). This second hypothesis states that the science of integration allows to scale up mobile-robot intelligence in real-world applications. In Chapter 7 we have demonstrated an autonomous construction application that is beyond the state of the art of this field, while using only commonly available hardware parts. We achieved this by performing a complete vertical integration from the mechanic, electronic and the low-level control software up to the reasoning algorithms. This successful application, and the lessons we learnt doing it, validate Hypothesis 1.2.

9.3 Contributions to the state of the art

This work provides the following contributions to the state of the art:

- With aseba, we have demonstrated a novel architecture for distributed low-level control of miniature mobile robots. Aseba is the first architecture for such robots that allows both event-based communication and dynamic reprogramming of the microcontrollers.
- For the first time, we have demonstrated the use of a global optimisation to find the parameters of a slam algorithm. This optimisation takes the reconstruction quality of the trajectory as the evaluation function and optimises over the parameter space.
- We have implemented the first multi-computer parallel htn planner. We have shown that this feature dictates the implementation choices for the htn algorithm and leads to improved performances over existing planners.
- We have demonstrated an autonomous construction application using a miniature mobile robot. This application is beyond the state of the art in this field.
- We have performed the first complete vertical integration of a complex robotic application involving high-level reasoning and structure construction on a miniature mobile robot. Based on this work, we have contributed insights to the question of integration in mobile robotics.
- We have contributed a theoretical framework of the integration of optimal htn planning and learning under limited computational

resources.

9.4 Final conclusion

In conclusion, we want to stress once again the importance of integration for mobile-robotic research. Mobile robotics is different from other fields because both the problem space and the solution space are huge and highly non-linear. It is difficult to specify the requirements for a particular problem; it is hard to find a good solution to this problem; and most of the time this solution will not generalise. Yet, we see no theoretical obstacle to scale up the capabilities of mobile robots, even in theory to human-level intelligence. Improving the intelligence of mobile robots requires a lot of real-world experimentation following a path of increasingly complex scenarios. And this path is filled with many open questions related to integration. We hope that this work sheds some light on how to approach these questions.

Appendix A

Technical details on ASEBA

A.1 EBNF grammar of the language

Formally, the EBNF grammar of the ASEBA language is the following:

```
program =
    [ { variable_definition } ] {statement} EOF ;

statement =
    onevent_definition |
    subroutine_declaration |
    block_statement ;

block_statement =
    event_emission |
    if_statement |
    when_statement |
    for_statement |
    while_statement |
    function_call |
    subroutine_call |
    assignment ;

variable_definition =
   "var" VAR [ "=" INT16 { "," INT16 } ] ;

assignment =
    var_write "=" shift_expression ;
```

```
if_statement =
   "if" or_expression "then" {block_statement}
   [ "else" {block_statement} ] "end" ;

when_statement =
    "when" or_expression "do" {block_statement} "end" ;

for_statement =
    "for" VAR "in" INT16 ":" INT16 [ "step" INT16 ]
    "do" {block_statement} "end" ;

while_statement =
    "while" or_expression "do" {block_statement} "end" ;

function_call =
    "call" FUNC "(" [ var_array_read { "," var_array_read } ] ")" ;

subroutine_call =
    "callsub" SUB ;

subroutine_declaration =
    "sub" SUB ;

onevent_definition =
    "onevent" EVENT ;

event_emission =
    "emit" EVENT [var_array_read] ;

or_expression =
     and_expresssion "or" and_expresssion ;

and_expresssion =
     not_expression "and" not_expression ;

not_expression =
    { "not" } condition;

condition =
    binary_or_expression condition_literal binary_or_expression ;
```

```
condition_literal =
    ">" | "<" | ">=" | "<=" |"==" | "!=" ;

binary_or_expression =
    binary_xor_expression |
    binary_xor_expression "|" binary_xor_expression ;

binary_xor_expression =
    binary_and_expression |
    binary_and_expression "^" binary_and_expression ;

binary_and_expression =
    shift_expression |
    shift_expression "&" shift_expression ;

shift_expression =
    add_expression |
    add_expression ">>" add_expression |
    add_expression "<<" add_expression ;

add_expression =
    mult_expression |
    mult_expression "+" mult_expression |
    mult_expression "-" mult_expression ;

mult_expression =
    unary_expression |
    unary_expression "*" unary_expression |
    unary_expression "/" unary_expression |
    unary_expression "%" unary_expression ;

unary_expression =
    INT16 |
    var_read |
    "(" or_expression ")" |
    "-" unary_expression |
    "~" unary_expression |
    "abs" unary_expression |
    var_read ;
```

```
var_read =
    VAR |
    VAR "[" binary_or_expression "]" ;

var_array_read =
    VAR  [ "[" INT16 [ ":" INT16 ] "]" ] ;

var_write =
    VAR |
    VAR "[" binary_or_expression "]" ;
```

where:

- INT16 is a 16-bit integer or a constant,
- VAR is a known (or to be defined) variable,
- FUNC is a known native function,
- EVENT is an event name,
- SUB is a know (or to be defined) subroutine,
- EOF is the end of input file.

This grammar does not show the additional elements used in the parser for intelligent error reporting. For information about these, please read the source code of the Aseba compiler. This grammar enforces syntactic consistency. Semantic consistency is enforced by a mix of identifier validation, variable range checking, a type checking pass on the parsed tree and run-time checks. The Aseba language supports comments on single lines or after any valid statement. Comments consist of an "#" and the comment, which continues until the end of the line. The lexer removes comments prior to calling the parser.

A.2 Deployment of an ASEBA VM

The VM itself is straightforward to port, as it is plain C and thus should compile on any platform. In any case, the source code of the VM counts fewer than 1000 lines of C, so adapting it to non-standard C should be feasible in a short time. The interfaces to the external world require slightly more work.

First, the interface to the Aseba network will depend on the type of bus one uses. Currently, Aseba supports the CAN bus and direct links (usually serial). The `transport/` directory in the Aseba source code contains adapters, that on one end provide functions to the VM for communication

and on the other end provide an interface to the communication bus. A CAN adapter is available in `transport/can`, and a generic adapter for stream-based communication links is available in `transport/buffer`.

Second, the interfaces to the sensors, actuators and execution-flow sources (typically interrupts) depend on the feature-set of the microcontroller and on the application. The directory `targets/` in the ASEBA source code contains several examples. The directories `targets/challenge` and `targets/playground` provide examples of VM controlling virtual e-puck robots inside the Enki simulator. The directory `targets/e-puck` contains the implementation of ASEBA on the real e-puck robot. All these examples use the buffer transport interface. In opposite, the directory `targets/dspic33` contains a generic structure for running ASEBA inside a dspic33 using CAN as a communication layer.

The directory `targets/can-translator/` contains the code and the schematics of a CAN to UART translator using a dspic33. This code uses Molole [1] to access the dspic's peripherals.

In summary, to port ASEBA to a new microcontroller, one first has to select the type of communication one is interested in, and then to choose an example close to the target application, and start hacking from there.

1. `http://mobots.epfl.ch/molole.html`

Appendix B

Experimental supplementary material

B.1 Source code for experiment 3.3.3

This section lists the source code that we used to perform experiment 3.3.3 (p. 37).

B.1.1 Constants

name	value
`proximity.Threshold`	38
`proximity.Shift`	14
`ground.ThresholdDetect`	450
`ground.ThresholdLeave`	570
`ground.Shift`	11
`sensors.Period`	15

B.1.2 Events

name	data size
`Stop`	0
`SetSpeed`	2
`HoleDetected`	1
`ObstacleDetected`	1
`FreeOfHole`	0
`FreeOfObstacle`	0

B.1.3 Left motor microcontroller

```
var user_target = 0
var hole_target = 0
var obstacle_target = 0
var val

sub UpdateTargetSpeed
if hole_target != 0 then
  motor.pid.target_speed = hole_target
else
  val = |user_target + 1
  call math.muldiv(val, val, obstacle_target, 60)
  motor.pid.target_speed  = user_target + val
end

onevent Stop
user_target = 0
callsub UpdateTargetSpeed

onevent SetSpeed
user_target = event.args[0]
callsub UpdateTargetSpeed

onevent HoleDetected
hole_target = event.args[0]
callsub UpdateTargetSpeed

onevent FreeOfHole
hole_target = 0
callsub UpdateTargetSpeed

onevent ObstacleDetected
obstacle_target = event.args[0] + event.args[1]
callsub UpdateTargetSpeed

onevent FreeOfObstacle
obstacle_target = 0
callsub UpdateTargetSpeed
```

B.1.4 Right motor microcontroller

```
var user_target = 0
var hole_target = 0
var obstacle_target = 0
var val

sub UpdateTargetSpeed
if hole_target != 0 then
```

```
  motor.pid.target_speed = hole_target
else
  val = |user_target + 1
  call math.muldiv(val, val, obstacle_target, 60)
  motor.pid.target_speed  = user_target + val
end

onevent Stop
user_target = 0
callsub UpdateTargetSpeed

onevent SetSpeed
user_target = event.args[1]
callsub UpdateTargetSpeed

onevent HoleDetected
hole_target = event.args[0]
callsub UpdateTargetSpeed

onevent FreeOfHole
hole_target = 0
callsub UpdateTargetSpeed

onevent ObstacleDetected
obstacle_target = event.args[0] - event.args[1]
callsub UpdateTargetSpeed

onevent FreeOfObstacle
obstacle_target = 0
callsub UpdateTargetSpeed
```

B.1.5 Proximity sensors microcontroller

```
var proximity.vectorX[24] =
  -254, -241, -212, -168, -113, -50, 17, 82, 142, 192, 229, 250,
  254, 241, 212,  168, 113, 50, -17, -82, -142, -192, -229, -250
var proximity.vectorY[24] =
  -17, -82, -142, -192, -229, -250, -254, -241, -212, -168, -113, -50,
  17, 82, 142, 192, 229, 250, 254, 241, 212, 168, 113, 50
# differences are negative
var ground.vectorX[8] = 255, 180, 0, -180, -255, -180, 0, 180
var ground.vectorY[8] = 0, 180, 255, 180, 0, -180, -255, -180
var hole = 0
var targets[2]
var i
var val
var min
var max
var mean
```

```
var eventBuffer[2]
var proximity.activation
var angle1
var angle2
var angleScalarProduct
var inHole

var proximity.skipped = 20
var proximity.old[2]
var tmp[2]

sensors.period = sensors.Period

onevent Stop
sensors.period = 10
targets[0] = 0
targets[1] = 0

onevent SetSpeed
targets[0] = event.args[0] + event.args[1]
targets[1] = event.args[1] - event.args[0]

onevent sensors.updated

# obstacle avoidance
call math.dot(
  eventBuffer[0], proximity.corrected, proximity.vectorX, proximity.Shift)
call math.dot(
  eventBuffer[1], proximity.corrected, proximity.vectorY, proximity.Shift)
call math.dot(proximity.activation, eventBuffer, eventBuffer,4)

if proximity.activation > proximity.Threshold then
  call math.sub(tmp, eventBuffer, proximity.old)
  call math.dot(val, tmp, tmp, 4)
  if val > 15 then
    proximity.skipped = 20
  end

  if proximity.skipped >= 20 then
    if (targets[0] != 0 or targets[1] != 0) and
        |eventBuffer[0] < 6 and |eventBuffer[1] < 6 then
      eventBuffer[1] = 20
    end
    call math.copy(proximity.old, eventBuffer)
    proximity.skipped = 0
    emit ObstacleDetected eventBuffer
  else
    proximity.skipped = proximity.skipped + 1
  end
```

```
end

when proximity.activation <= proximity.Threshold do
  proximity.skipped = 20
  emit FreeOfObstacle
end
```

Bibliography

[1] F. Abrate, B. Bona, and M. Indri. Experimental EKF-based SLAM for mini-rovers with IR sensors only. In *Preceedings of 3rd European Conference on Mobile Robots*. European Conference on Mobile Robots, 2007.

[2] J. Albus, A. Barbera, and R. Nagel. Theory and practice of hierarchical control. In *Proceedings of the 23rd IEEE Computer Society International Conference*, pages 18–39, 1981.

[3] J. Anderson. ACT: A simple theory of complex cognition. *American Psychologist*, 51(4):355–365, 1996.

[4] 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)*, pages 3933–3938. IEEE Press, 2005.

[5] R. Beckers, O. Holland, and J.-L. Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. In R. A. Brooks and P. Maes, editors, *Proceedings of the* 4^{th} *International Workshop on the Synthesis and Simulation of Living Systems (Artificial Life IV)*, pages 181–189. MIT Press, Cambridge, MA, 1994.

[6] P. Beeson, J. Modayil, and B. Kuipers. Factoring the mapping problem: Mobile robot map-building in the Hybrid Spatial Semantic Hierarachy. *The International Journal of Robotics Research*, 2009.

[7] R. Bencina and M. Kaltenbrunner. The Design and Evolution of Fiducials for the reacTIVision System. In *Proceedings of the 3rd International Conference on Generative Systems in the Electronic Arts*, Melbourne, Australia, 2005.

[8] H. Beyer. *The theory of evolution strategies*. Springer Verlag, 2001.

[9] R. Bonasso, R. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2):237–256, 1997.

[10] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobilerobots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

[11] M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. In *Proceedings of the 2006 international symposium on*

Practical cognitive agents and robots, pages 15–26. ACM New York, NY, USA, ACM Press, 2006.

[12] R. A. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23, 1986.

[13] D. Brugali, S. Wrede, and I. Lütkebohle, editors. *EBS-RO09 workshop*, 2009.

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

[15] R. Burton. Connect desktop apps using d-bus. http://www.ibm.com/developerworks/linux/library/l-dbus.html, 2004.

[16] A. Chalmers. *What is this thing called science?* Univ. of Queensland Press, 2006.

[17] T. Collett, B. MacDonald, and B. Gerkey. Player 2.0: Toward a practical robot programming framework. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005)*, 2005.

[18] S. Collins, A. Ruina, R. Tedrake, and M. Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307(5712):1082, 2005.

[19] C. Cote, Y. Brosseau, D. Letourneau, C. Raïevsky, and F. Michaud. Robotic software integration using MARIE. *International Journal of Advanced Robotic Systems*, 3(1):55–60, 2006.

[20] M. Cummins and P. Newman. FAB-MAP: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665, 2008.

[21] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[22] M. DesJardins, E. Durfee, C. Ortiz, and M. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 1999.

[23] M. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, July 2006.

[24] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[25] J. Dix, H. Munoz-Avila, D. Nau, and L. Zhang. IMPACTing SHOP: Putting an AI planner into a multi-agent environment. *Annals of Mathematics and Artificial Intelligence*, 37(4):381–407, 2003.

[26] T. Duckett. A Genetic Algorithm for Simultaneous Localization and Mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 434–439. IEEE Press, 2003.

[27] E. Durfee. Distributed problem solving and planning. *Multiagent systems: a modern approach to distributed artificial intelligence*, pages 121–164, 1999.

[28] 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*, pages 1331–1336. IEEE Press, May 1990.

[29] A. Elfes. Using Occupancy Grids for Mobile Robot Perception and Navigation. *Computer*, 22(6):46–57, Jun 1989.

[30] K. Erol, J. A. Hendler, and D. S. Nau. HTN Planning: Complexity and Expressivity. In *AAAI*, pages 1123–1128. AAAI Press, 1994.

[31] A. Feldman and R. Serrano. *Welfare economics and social choice theory*. Springer Verlag, 2006.

[32] D. J. Feng, S. Wijesoma, and A. Shacklock. Genetic Algorithmic Filter Approach to Mobile Robot Simultaneous Localization and Mapping. In *9th International Conference on Control, Automation, Robotics and Vision*, pages 1–6. IEEE Press, Dec. 2006.

[33] P. Fitzpatrick, G. Metta, and L. Natale. Towards long-lived robot genes. *Robotics and Autonomous Systems*, 56(1):29–45, 2008.

[34] S. Fleury, M. Herrb, and R. Chatila. Genom: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *International conference on intelligent robots and systems*, volume 2, pages 842–848, 1997.

[35] R. Floyd. Nondeterministic algorithms. *Journal of the ACM (JACM)*, 14(4):636–644, 1967.

[36] N. Franks and J. Deneubourg. Self-organizing nest construction in ants: individual worker behaviour and the nest's dynamics. *Animal Behaviour*, 54(4):779–796, 1997.

[37] E. Gambao, C. Balaguer, and F. Gebhart. Robot assembly system for computer-integrated construction. *Automation in Construction*, 9(5-6):479–487, 2000.

[38] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-wesley Reading, MA, 1995.

[39] E. Gat. On three-layer architectures. In D. Kortenkamp, R. P. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*, pages 195–210. MIT/AAAI Press, 1997.

[40] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. Morgan Kaufmann, 2004.

[41] C. Gifford, R. Webb, J. Bley, D. Leung, M. Calnon, J. Makarewicz, B. Banz, and A. Agah. Low-Cost Multi-Robot Exploration and Mapping. In *Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications*, pages 74–79. IEEE Press, 2008.

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

[43] L. Gottfredson. Mainstream science on intelligence: An editorial with 52 signatories, history, and bibliography. *Intelligence*, 24(1):13–23, 1997.

[44] 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*, volume 5, pages 5250–5255, April-1 May 2004.

[45] S. Grzonka, G. Grisetti, and W. Burgard. Towards a Navigation System for Autonomous Indoor Flying. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2878–2883, Kobe, Japan, 2009. IEEE Press.

[46] R. Haralick, S. Sternberg, and X. Zhuang. Image analysis using mathematical morphology. *IEEE PATTERN ANAL. MACH. INTELLIG.*, 9(4):532–550, 1987.

[47] S. Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346, 1990.

[48] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4:100–107, 1968.

[49] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[50] N. Hawes, A. Sloman, J. Wyatt, M. Zillich, H. Jacobsson, G. Kruijff, M. Brenner, G. Berginc, and D. Skocaj. Towards an integrated robot with multiple cognitive functions. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, pages 1548–1553. AAAI Press, 2007.

[51] H. Hayashi, S. Tokura, and F. Ozaki. Towards Real-World HTN Planning Agents. *Knowledge Processing and Decision Making in Agent-based Systems*, 170:13–41, 2009.

[52] M. Henning. A new approach to object-oriented middleware. *Internet Computing, IEEE*, 8(1):66–75, Jan–Feb 2004.

[53] T. Henzinger and C. Kirsch. A typed assembly language for real-time programs. In *Proceedings of the 4th ACM international conference on Embedded software*, pages 104–113. ACM, 2004.

[54] 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*, volume 2638 of *Lecture Notes in Computer Science*, pages 159–187. Springer-Verlag, 2003.

[55] T. Huntsberger, P. Pirjanian, A. Trebi-Ollennu, H. Nayar, H. Aghazarian, A. Ganino, M. Garrett, S. Joshi, and P. Schenker. CAMPOUT: A control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 33(5):550–559, 2003.

[56] P. Kahn, N. Freier, T. Kanda, H. Ishiguro, J. Ruckert, R. Severson, and S. Kane. Design patterns for sociality in human-robot interaction. In *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, pages 97–104. ACM, 2008.

[57] S. Kim, J. Clark, and M. Cutkosky. iSprawl: Design and tuning for high-speed autonomous open-loop running. *The International Journal of Robotics Research*, 25(9):903, 2006.

[58] M. Krishna, J. Bares, and E. Mutschler. Tethering system design for Dante II. In *1997 IEEE International Conference on Robotics and Automation, 1997. Proceedings.*, volume 2, 1997.

[59] B. Kuipers. The Spatial Semantic Hierarchy. *Artificial Intelligence*, 119(1–2):191–233, 2000.

[60] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[61] H. Landis. Production-ready global illumination. In *Siggraph Course Notes*, volume 16, pages 87–101. ACM, 2002.

[62] D. Long and M. Fox. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20(1):1–59, 2003.

[63] R. P. Loui. In praise of scripting: Real programming pragmatism. *Computer*, 41(7):22–26, 2008.

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

[65] S. Magnenat, V. Longchamp, M. Bonani, P. Rétornaz, P. Germano, H. Bleuler, and F. Mondada. Affordable SLAM through the Co-Design of Hardware and Methodology. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*, 2010. To be published.

[66] Q. Mahmoud, editor. *Middleware for Communications*. Wiley, 2004.

[67] T. Marks, A. Howard, M. Bajracharya, G. Cottrell, and L. Matthies. Gamma-SLAM: Visual SLAM in unstructured environments using variance grid maps. *Journal of Field Robotics*, 26(1):26–51, 2008.

[68] B. McNaughton, F. Battaglia, O. Jensen, E. Moser, and M. Moser. Path integration and the neural basis of the'cognitive map'. *Nature Reviews Neuroscience*, 7(8):663–678, 2006.

[69] C. Melhuish, J. Welsby, and C. Edwards. Using templates for defensive wall building with autonomous mobile ant-like robots. In *Proceedings of Towards Intelligent Autonomous Mobile Robots*, volume 99, 1999.

[70] M. Mizukawa, H. Matsuka, T. Koyama, and A. Matsumoto. ORiN: Open Robot Interface for the Network, a proposed standard. *Industrial Robot: An International Journal*, 27(5):344–350, 2000.

[71] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a Robot Designed for Education in Engineering. In P. Gonçalves, P. Torres, and C. Alves, editors, *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pages 59–65. IPCB: Instituto Politécnico de Castelo Branco, 2009.

[72] 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*, 17(2-3):193–221, 2004.

[73] 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*, 17(2–3):193–221, 2004.

[74] M. Montemerlo, N. Roy, and S. Thrun. Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pages 2436–2441. Citeseer, 2003.

[75] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *IJCAI*, pages 1151–1156, 2003.

[76] D. Nau, T. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20(1):379–404, 2003.

[77] R. A. Newcombe and A. J. Davison. Live dense reconstruction with a single moving camera. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Press, 2010.

[78] N. Nilson et al. Shakey the robot. *SRI AI Center Technical Note*, 323, 1984.

[79] A. Nüchter and J. Hertzberg. Towards semantic maps for mobile robots. *Robotics and Autonomous Systems*, 56(11):915–926, 2008.

[80] O. Obst and J. Boedecker. Flexible Coordination of Multiagent Team Behavior Using HTN Planning. *Lecture Notes in Computer Science*, 4020:521–528, 2006.

[81] C. A. C. Parker and H. Zhang. Collective Robotic Site Preparation. *Adaptive Behavior*, 14(1):5–19, 2006.

[82] D. Pellier and H. Fiorino. A Unified Framework Based on HTN and POP Approaches for Multi-Agent Planning. In *Intelligent Agent Technology. IEEE/WIC/ACM International Conference on*, pages 285–288. IEEE Press, Nov. 2007.

[83] 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)*, pages 2361–2368. IEEE Press, 2001.

[84] R. Philippsen. A light formulation of the e* interpolated path replanner. Technical report, Autonomous Systems Lab, Ecole Polytechnique Federale de Lausanne, 2006.

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

[86] P. Pirjanian, N. Karlsson, L. Goncalves, and E. Di Bernardo. Low-cost visual localization and mapping for consumer robotics. *Industrial Robot: An International Journal*, 30:139–144, 2003.

[87] S. Popper. *The logic of scientific discovery*. Hutchinson, 1959.

[88] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[89] V. N. Rao and V. Kumar. Superlinear speedup in parallel state-space search. In *Proceedings of the Eighth Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 161–174, London, UK, 1988. Springer-Verlag.

[90] F. Rochat, P. Schoeneich, M. Bonani, S. Magnenat, and F. Mondada. Design of magnetic switchable device and applications in climbing robots. In *Climbing And Walking Robots: Proceedings of the 13th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, 2010. To appear.

[91] D. Sabatta, D. Scaramuzza, R. Siegwart, and Z. ETH. Improved Appearance-Based Matching in Similar and Dynamic Environments using a Vocabulary Tree. In *International Conference on Robotics and Automation (ICRA)*. IEEE Press, 2010.

[92] C. Schroter, H. Bohme, and H. Gross. Memory-Efficient Gridmaps in Rao-Blackwellized Particle Filters for SLAM using Sonar Range Sensors. In *Proceedings of the European Conference on Mobile Robots 2007*, pages 138–143, 2007.

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

[94] B. Siciliano and O. Khatib. *Springer handbook of robotics*. Springer, 2008.

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

[96] R. Sim, P. Elinas, and J. Little. A study of the Rao-Blackwellised particle filter for efficient and accurate vision-based SLAM. *International Journal of Computer Vision*, 74(3):303–318, 2007.

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

[98] M. J. Skibniewski and S. C. Wooldridge. Robotic materials handling for automated building construction technology. *Automation in Construction*, 1(3):251–266, 1992.

[99] A. Sloman. Beyond shallow models of emotion. *Cognitive Processing*, 2(1):177–198, 2001.

[100] A. Sloman. Requirements for Digital Companions It's harder than you think. In *Close Engagements with Artificial Companions: Key social, psychological, ethical and design issues*, pages 179–200, 2007.

[101] I. Sobel and G. Feldman. A 3x3 isotropic gradient operator for image processing. *Never published but presented at a talk at the Stanford Artificial Project*, 1968.

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

[103] R. L. Stewart and R. A. Russell. A Distributed Feedback Mechanism to Regulate Wall Construction by a Robotic Swarm. *Adaptive Behavior*, 14(1):21–51, 2006.

[104] A. Stroupe, T. Huntsberger, B. Kennedy, H. Aghazarian, E. Baumgartner, A. Ganino, M. Garrett, A. Okon, M. Robinson, and J. Townsend. Heterogeneous robotic systems for assembly and servicing. In B. Battrick, editor, *'i-SAIRAS 2005'-The 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, volume 603, page 82. ESA SP-603. European Space Agency, 2005.

[105] A. Stroupe, A. Okon, M. Robinson, T. Huntsberger, H. Aghazarian, and E. Baumgartner. Sustainable cooperative robotic technologies for human and robotic outpost infrastructure construction and maintenance. *Autonomous Robots*, 20(2):113–123, 2006.

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

[107] W. M. Thorburn. The myth of occam's razor. *Mind*, 27:345–353, 1918.

[108] S. Thrun. *Probabilistic robotics*. ACM, 2002.

[109] M. Toussaint, N. Plath, T. Lang, and N. Jetchev. Integrated motor control, planning, grasping and high-level reasoning in a blocks world using probabilistic inference. In *International Conference on Robotics and Automation (ICRA)*. IEEE Press, 2010.

[110] N. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A. Ijspeert, M. Carrozza, et al. iCub: the design and realization of an open humanoid platform for cognitive and neuroscience research. *Advanced Robotics*, 21(10):1151–1175, 2007.

[111] O. Unver, A. Uneri, A. Aydemir, and M. Sitti. Geckobot: a gecko inspired climbing robot using elastomer adhesives. In *International Conference on Robotics and Automation (ICRA)*, pages 2329–2335. IEEE Press, 2006.

[112] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar. Miro - middleware for mobile robot applications. *Robotics and Automation, IEEE Transactions on*, 18(4):493–497, Aug 2002.

[113] J. Vincent, O. Bogatyreva, N. Bogatyrev, A. Bowyer, and A. Pahl. Biomimetics: its practice and theory. *Journal of the Royal Society Interface*, 3(9):471, 2006.

[114] J. Wawerla, G. Sukhatme, and M. Mataric. Collective construction with multiple robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2696–2701. IEEE Press, 2002.

[115] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen. Artificial intelligence: Autonomous mental development by robots and animals. *Science*, 291(5504):599–600, 2001.

[116] J. Werfel, Y. Bar-Yam, D. Rus, and R. Nagpal. Distributed construction by mobile robots with enhanced building blocks. In *Proceedings of 2006 IEEE International Conference on Robotics and Automation*, pages 2787–2794. IEEE Press, 2006.

[117] J. Werfel and R. Nagpal. Extended stigmergy in collective construction. *IEEE Intelligent Systems*, 21(2):20–28, 2006.

[118] T. Yap and C. Shelton. SLAM in Large Indoor Environments with Low-Cost, Noisy, and Sparse Sonars. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1395–1401. IEEE Press, May 2009.

[119] H. Zender, O. Martínez Mozos, P. Jensfelt, G. Kruijff, and W. Burgard. Conceptual spatial representations for indoor mobile robots. *Robotics and Autonomous Systems*, 56(6):493–502, 2008.

[120] 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*, pages 2355–2360. IEEE Press, 2001.

[121] R. Zlot and A. Stentz. Market-based multirobot coordination for complex tasks. *The International Journal of Robotics Research*, 25(1):73–102, 2006.

Curriculum vitæ

Stéphane Magnenat, born on 14th February 1980, Swiss citizenship

Education

2007–2010	**PhD degree in mobile robotics** (expected, August 2010) École Polytechnique Fédérale de Lausanne, Switzerland.
2007–2008	PhD courses · Tutoring Skills (2008) · Problem Solving in Computer Science (2007) · Machine Learning (2007) École Polytechnique Fédérale de Lausanne, Switzerland.
2005–2009	Personal development courses · Breakthrough Thinking (2009) · Prise de parole en public (2007) · Le stress, comment s'en servir ? (2005) École Polytechnique Fédérale de Lausanne, Switzerland.
1998–2003	**Master in computer science** École Polytechnique Fédérale de Lausanne, Switzerland.
1995–1998	**Swiss federal scientific maturity** Gymnase August Piccard, Lausanne, Switzerland.

Languages

French	native
English	fluent, daily spoken and written
German	intermediate, 8 years at school, weekly spoken

Research and working experience

11.2005–09.2006 **Laboratory of Robotics Systems, EPFL, Switzerland**
PhD student, teaching assistant. Supervision of students. Research on the integration methodology for miniature mobile robots. Development of a low-level event-based architecture for the control of sensors and actuators. Adaptation of several state-of-the-art capabilities (SLAM, HTN planning, symbol grounding, object manipulation) to the constraints of these robots. Research on structure construction by autonomous mobile robots. Implementation of a functional demonstrator; quantitative results. Worked in the *Swarmanoid* European project.

11.2005–09.2006 **Laboratory of Digital Systems, hepia, Geneva, Switzerland**
Research and teaching assistant. Development of signal processing software for the control of an orthesis for myopathic patients. Development of an open-source software oscilloscope. Development of the electronics and the control software for a miniature turbojet.

05.2003–10.2005 **Laboratory of Intelligent Systems, EPFL, Switzerland**
Research and teaching assistant, system administrator. Supervision of students. Research on collective robotics. Research on the emergence of communication in artificial agents, modelling of the evolutionary dynamics of social insects. Development of a fast open-source robot simulator. Development of an open-source artificial evolution framework. Worked in the *Swarmbot* and the *ECAgents* European projects.

2002–2003 **Processor Architecture Laboratory, EPFL, Switzerland**
Design of an open-hardware ARM XScale processor board for use in mobile robotics. Porting of Linux and adaptation of development tools. Design of a multimedia extension board. Semester and Master projects, group of two people.

2001–2002 **Laboratory of Peripheral Systems, EPFL, Switzerland**
Design and implementation of a virtual ecosystem running on a parallel supercomputer. Creation of an OpenGL 3D client. Semester project, group of two people.

2000–2001	**VulcanArts Development S.A., Geneva, Switzerland** Design and implementation of a software architecture for interactive television, leading of a team of five people.
1998–2006	**The Globulation Project, Internet** Initiator and project leader of an open-source multiplayer real-time strategy game. Design and implementation of the software framework and the game engine. Team of six people over Internet.
1999–2005	**DIDEL S.A., Lausanne, Switzerland** Development of various software for embedded development on PIC microcontroller.
1999–2001	**The SnakeMe Project, Internet** Design and implementation of a successful open-source snake game, $>$ 200000 downloads.
1999	**Cyberbotics SARL, Lausanne, Switzerland** Deployment of a 100 Mbps network under GNU/Linux.
1999	**Mediapolis SARL, Lausanne, Switzerland** Installation of GNU/Linux workstations.

Teaching activities

2010	Main teacher for the course of C++ programming, last year bachelor in microengineering, EPFL.
2008–	Initiator and coordinator of programming workshops for children using Aseba; team of 10 assistants, 2 events per year, EPFL.
2006–2010	Teaching assistant for courses of embedded systems and C++ programming, last year bachelor in microengineering, EPFL.
2005–2006	Teaching assistant for the course of embedded systems, microengineering and computer science students, hepia.
2003–2005	Teaching assistant for the master course of bio-inspired computing machines, EPFL.
2001–2003	Student assistant for courses of computer hardware, peripherals and real-time systems, EPFL.

Project supervision

Trainee	Kevin Frugier, Matthieu Bontemps
Master	Christophe Gusthiot, Martin Voelkle
Semester	Max Laager, Yves Stauffer, Guillaume Monnard

Scientific open-source software

2009–2010	Planner9, a HTN planner distributed on groups of miniature mobile robots. Used by EPFL, Switzerland for research.
2007–2010	Aseba, an event-based architecture for distributed control of mobile robots. Used by *Swarmanoid* and *Perplexus* European projects, and by EPFL and University of Fribourg, Switzerland.
1999–2010	Enki, a fast 2D mobile robot simulator. Used by EPFL for education and by Cyberbotics as the 2D mode for the Webots commercial robot simulator.
2005–2006	Osqoop, a software Oscilloscope with support for orthesis control. Used by hepia, Hesso, Geneva for research.
2004–2005	Teem, an artifical evolution framework. Used by EPFL, University of Lausanne, Switzerland and some other laboratories worldwide for research.

Presentations and talks

2010.05.07	Affordable SLAM through the Co-Design of Hardware and Methodology, at the *2010 IEEE International Conference on Robotics and Automation*, Anchorage, USA.
2010.01.27	Autonomous Construction by a Mobile Robot in Unknown Environments with Scarce Resources, at the *4th International Conference on Cognitive Systems*, ETH Zurich, Switzerland.

2010.01.22 Aseba Presentation, at the *Meeting day of the société suisse pour l'informatique dans l'enseignement (SSIE)*, EPFL, Lausanne, Switzerland. **Invited talk**.

2009.12.18 Planner9, a HTN planner distributed on groups of miniature mobile robots, at the *2nd International Conference on Intelligent Robotics and Applications*, Singapore.

2009.10.13 Segregation in swarms of mobile robots based on the Brazil nut effect, at the *2009 IEEE/RSJ International Conference on Intelligent RObots and Systems (IROS)*, St. Louis, USA.

2009.10.15 Aseba Meets D-Bus: From the Depths of a Low-Level Event-Based Architecture into the Middleware Realm, at the *EBS-RO workshop, 2009 IEEE/RSJ International Conference on Intelligent RObots and Systems (IROS)*, St. Louis, USA. **Invited talk**.

2009.05.16 Aseba Workshop, at the *Festival de Robotique 2009*, EPFL, Lausanne, Switzerland. **Invited workshop**.

2009.05.02 Aseba Workshop, at the *IC Faculty open days*, EPFL, Lausanne, Switzerland. **Invited workshop**.

2009.03.19 The Mobots' robots and their vision capabilities, at *HVRL - Hideo Saito Laboratory*, Keio University, Japan. **Invited talk**.

2009.02.20 Aseba Introduction and Tutorial, for the *Microclub*, EPFL, Lausanne, Switzerland. **Invited talk**.

2008.11.03 Scripting the swarm: event-based control of microcontroller-based robots, at the *International Workshop on Standards and Common Platforms for Robotics, SIMPAR 2008*, Venice, Italy.

2008.10.20 Aseba-Challenge: An Open-Source Multiplayer Introduction to Mobile Robots Programming, at the *International conference on Fun and Games*, Eindhoven, The Netherland.

2008.04.19 Aseba Workshop, at the *Festival de Robotique 2008*, EPFL, Lausanne, Switzerland. **Invited workshop**.

2008.02.23 Globulation 2, at the *Free and open source software developers' European meeting*, Brussel, Belgium. **Invited talk**.

2006.04.29 Presentation of hepia/LSN robots, at the *Salon international du Livre et de la Presse*, Geneva, Switzerland.

2004.11.16 S-bots usage in ECAgents, at the *IST 2004 Event*, The Hague, Netherlands.

2003.10.09 from S-bot to Swarmbot, at the *Evolvability & Interaction symposium: Evolutionary Substrates of Communication, Signaling, and Perception in the Dynamics of Social Complexity*, London, England.

Peer-reviewed publications

[1] M. Bonani, V. Longchamp, S. Magnenat, P. Rétornaz, D. Burnier, G. Roulet, H. Bleuler, and F. Mondada. The marxbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*. 2010. To appear.

[2] F. Rochat, P. Schoeneich, M. Bonani, S. Magnenat, and F. Mondada. Design of magnetic switchable device and applications in climbing robots. In *Climbing And Walking Robots: Proceedings of the 13th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*. 2010. To appear.

[3] S. Magnenat, P. Schoeneich, F. Rochat, P. Rétornaz, M. Bonani, V. Longchamp, M. Voelkle, T. Barras, D. Burnier, P. Noirat, T. Baaboura, F. Vaussard, and F. Mondada. Autonomous Construction by a Mobile Robot in Unknown Environments with Scarce Resources. Presented at CogSys 2010, ETH Zurich, Switzerland, 2010.

[4] S. Magnenat, P. Rétornaz, M. Bonani, V. Longchamp, and F. Mondada. ASEBA: A Modular Architecture for Event-Based Control of Complex Robots. *IEEE/ASME Transactions on Mechatronics*, 2010. To appear.

[5] S. Magnenat, V. Longchamp, M. Bonani, P. Rétornaz, P. Germano, Bleuler, Hannes,, and F. Mondada. Affordable SLAM through the Co-Design of Hardware and Methodology. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*. IEEE Press, 2010. To appear.

[6] S. Magnenat, M. Voelkle, and F. Mondada. Planner9, a HTN planner distributed on groups of miniature mobile robots. In M. Xie et al., editor, *Proceedings of the Second International Conference on Intelligent Robotics and Applications*, volume 5928 of *Lecture Notes in Computer Science*, pages 1013–1022. Springer, Berlin / Heidelberg, 2009.

[7] M. Bonani, S. Magnenat, P. Rétornaz, and F. Mondada. The Hand-bot, a Robot Design for Simultaneous Climbing and Manipulation. In M. Xie et al., editor, *Proceedings of the Second International Conference on Intelligent Robotics and Applications*, volume 5928 of *Lecture Notes in Computer Science*, pages 11–22. Springer, Berlin / Heidelberg, 2009.

[8] S. Magnenat and F. Mondada. Aseba Meets D-Bus: From the Depths of a Low-Level Event-Based Architecture into the Middleware Realm. In *IEEE TC-Soft Workshop on Event-based Systems in Robotics (EBS-RO)*. 2009. Invited paper.

[9] R. Groß, S. Magnenat, and F. Mondada. Segregation in swarms of mobile robots based on the Brazil nut effect. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4349–4356. IEEE Press, 2009.

[10] R. Groß, S. Magnenat, L. Küchler, V. Massaras, M. Bonani, and F. Mondada. Towards an Autonomous Evolution of Non-Biological Physical Organisms. In *Proceedings of the 10th European Conference on Artificial Life*, Lecture Notes in Computer Science. Springer, Berlin / Heidelberg, 2009.

[11] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a Robot Designed for Education in Engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, volume 1, pages 59–65. IPCB: Instituto Politécnico de Castelo Branco, 2009.

[12] S. Magnenat, P. Rétornaz, B. Noris, and F. Mondada. Scripting the swarm: event-based control of microcontroller-based robots. In E. Menegatti, editor, *SIMPAR 2008 Workshop Proceedings*. 2008. Workshop Proceedings ISBN: 978-88-95872-01-8.

[13] S. Magnenat, B. Noris, and F. Mondada. Aseba-Challenge: An Open-Source Multiplayer Introduction to Mobile Robots Programming. In P. Markopoulos, B. de Ruyter, W. IJsselsteijn, and D. Rowland, editors, *Fun and Games*, Lecture Notes in Computer Science, pages 65–74. Springer, Berlin / Heidelberg, 2008.

[14] A. Beyeler, S. Magnenat, and A. Habersaat. Ishtar: a flexible and lightweight software for remote data access. In *European Micro Air Vehicle Conference EMAV08*. 2008.

[15] S. Magnenat, V. Longchamp, and F. Mondada. ASEBA, an event-based middleware for distributed robot control. In *Workshops and*

Tutorials CD IEEE/RSJ 2007 International Conference on Intelligent Robots and Systems. IEEE Press, 2007.

[16] D. Floreano, S. Mitri, S. Magnenat, and L. Keller. Evolutionary Conditions for the Emergence of Communication in Robots. *Current Biology*, volume 17, pages 514–519, 2007.

[17] M. Waibel, D. Floreano, S. Magnenat, and L. Keller. Division of labour and colony efficiency in social insects: effects of interactions between genetic architecture, colony kin structure and rate of perturbations. *Proceedings of the Royal Society B*, volume 273, pages 1815–23, 2006.

[18] F. Mondada, M. Bonani, A. Guignard, S. Magnenat, C. Studer, and D. Floreano. Superlinear Physical Performances in a SWARM-BOT. In *Proceedings of the VIIIth European Conference on Artificial Life*, Lecture Notes in Artificial Intelligence, pages 282–291. Springer, Berlin / Heidelberg, 2005.

[19] F. Mondada, M. Bonani, S. Magnenat, A. Guignard, D. Floreano, F. Groen, N. Amato, A. Bonari, E. Yoshida, and B. Kröse. Physical connections and cooperation in swarm robotics. In *8th Conference on Intelligent Autonomous Systems (IAS8)*, pages 53–60. 2004.

Technical Skills

applied mathematics	machine learning, baysian methods, neural networks, numerical optimisation, evolutionary computation
robotic technologies	system integration, sensor calibration, digital electronics, embedded systems, microcontrollers, middleware, virtual machines, compilers
programming languages	C++, C, Java, Scala, Python, Perl, R, MATLAB, PHP, Ada, Pascal, assembly, Smalltalk, Lisp, BASIC, SQL
frameworks	C++ STL, Qt, OpenGL, Open Inventor, SDL, Posix, Win32

Grants and fellowships

1999.10.4–8	Travel and venue grant for participation to the European Student Outreach Program and 50 IAF congress in Amsterdam, Netherland.

Colophon

This document is written in British English and typeset with LaTeX. The text typeface is Palatino in 11 pt, while the graphics and illustrations use either Helvetica or Nimbus Sans L. I made most of the illustrations using Inkscape and most of the graphs using R. In rare cases I have used XFig, Illustrator, Asymptote or my own C++ code.

The extinguisher and the fire pictures used in the schematics in Chapter 6 are from the following URLs:

`http://openclipart.org/people/Anonymous/Anonymous_aiga_fire_extinguisher.svg`

`http://openclipart.org/people/Anonymous/Anonymous_fire.svg`

Both pictures are in the public domain.