Autonomous Robots
© The Author(s) 2012
10.1007/s10514-012-9296-x

Autonomous construction using scarce resources in unknown environments

Ingredients for an intelligent robotic interaction with the physical world

Stéphane Magnenat · Roland Philippsen · Francesco Mondada

Received: 18 February 2011 / Accepted: 26 April 2012 / Published online: 15 May 2012

Abstract The goal of creating machines that autonomously perform useful work in a safe, robust and intelligent manner continues to motivate robotics research. Achieving this autonomy requires capabilities for understanding the environment, physically interacting with it, predicting the outcomes of actions and reasoning with this knowledge. Such intelligent physical interaction was at the centre of early robotic investigations and remains an open topic.

In this paper, we build on the fruit of decades of research to explore further this question in the context of autonomous construction in unknown environments with scarce resources. Our scenario involves a miniature mobile robot that autonomously maps an environment and uses cubes to bridge ditches and build vertical structures according to high-level goals given by a human.

Based on a "real but contrived" experimental design, our results encompass practical insights for future applications that also need to integrate complex behaviours under hardware constraints, and shed light on the broader question of the capabilities required for intelligent physical interaction with the real world.

This work was supported by the Swarmanoid project, which is funded by the Future and Emerging Technologies program (IST-FET) of the European Community under grant IST-022888.

Stéphane Magnenat
Autonomous System Lab, ETH Zentrum CLA E31, Tannenstrasse 3, 8092 Zurich, Switzerland
E-mail: stephane at magnenat dot net

Roland Philippsen
Intelligent Systems Laboratory, Halmstad University, Sweden
E-mail: roland.philippsen@hh.se

Francesco Mondada
EPFL-LSRO, Station 9, CH-1015 Lausanne, Switzerland
E-mail: francesco.mondada@epfl.ch

Keywords intelligent physical interaction · reasoning · HTN planning · symbol grounding · autonomous construction · miniature mobile robots

1 Introduction

Since its beginnings, robotics research has been inspired by the vision of machines that autonomously perform useful but tedious work in a safe and robust manner. Therefore, a fundamental requirement is the capability of influencing the physical world, as amply demonstrated by the prevalence of robots in factories. However, translating this requirement into mobile robots operating in uncontrolled environments demands a greater understanding of the world and its rules, which requires intelligence[1]. For many roboticists, the vision remains unfulfilled, as the degree of autonomy or intelligence exhibited by commercially-available mobile robots is usually still deemed very limited. This means that robots are currently deployed only in a small subset of possible applications, and thus we can still greatly increase their contribution to society. In the past decades, many of the problems limiting a broader deployment of robots have been discovered and addressed by roboticists. Nowadays, the accumulated wealth of robotic knowledge increasingly allows researchers to draw from existing components and methods in order to revisit broader questions about intelligence and its application to real tasks.

We situate our work in the empirical investigation of robots with comprehensive capabilities for real-world interaction. We presume that a rich interplay with the physical environment is an essential ingredient for intelligence. We employ novel combinations of hardware and software to build miniature embodied systems for experimental studies of autonomous goal-directed environment modification. Inspired by early pioneering work on the robot Shakey [37], and now able to exploit significant advances in the state of the art, we investigate which components and methods are needed and how

[1] Although there is no commonly accepted definition of intelligence, there are consensual approximations [12,26]. The Encyclopedia Britannica (2006), as quoted by [26], lists many of the properties that we deem relevant for deciding whether a robot can be considered intelligent:

> [...] ability to adapt effectively to the environment, either by making a change in oneself or by changing the environment [...] intelligence is [...] a combination of many mental processes [...]

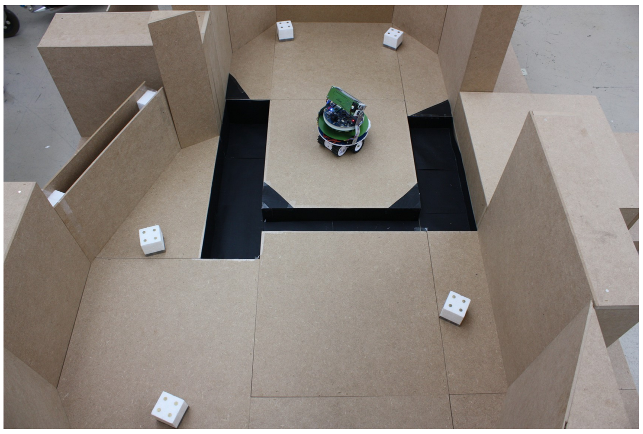


Figure 1 The experimental setup: the black area is a ditch separating two traversable regions. The robot can manipulate the resources (white cubes) to gain access to the other side of the ditch and build a structure.

to coordinate them. Experimental design and system integration are central aspects of our work.

In this paper, we use the example of building a tower out of scarce resources scattered through a partially accessible and unknown environment. Our experimental setup is inspired by the challenges of autonomous construction, which has several potential applications: in the short term, the building industry has recognised the importance of robotics[2] for its potential to reduce casualties and improve efficiency and acknowledges that much work remains to be done [63]. In the long term, robots could replace or reinforce infrastructure after natural disasters, or they might construct settlements for space exploration. We think that autonomous robots capable of reliable construction work must show at least some level of intelligence, and we attribute the uncommon appearance of autonomous construction in mobile-robotics research to the complex subsystem integration necessary to implement sufficiently intelligent behaviours. Our tower-building scenario (see Figure 1) emphasizes three related challenges: it requires robust real-world action execution, careful planning for the use of scarce resources, and real-time reasoning grounded in on-board perception.

2 Related work

Robotics work that is related to construction is diverse and can be found in a variety of fields. The reason is that construction provides a rich context for the investigation of many research questions. This circumstance has also been noted by others, such as [59] who sketch a classification of related work according to the emphasis of research, the test bed and approach, the nature of the robots and the size of the robot team. A crisp categorization of works related to autonomous construction is non-trivial and beyond the scope of this paper, so here we sketch it in broad strokes. For our discussion, the *research focus* is especially important, because it predominates the nature of the experiments and the interpretation of their outcomes. As construction implies close and complex interactions with the environment, it is difficult to simulate this task realistically. Building a simulator of a construction scenario is a problem in itself, and requires substantial experimental trials, which in turn require a physical robot. For this reason, the rest of this section mainly surveys the work employing real robots.

Research on *bio-inspired swarm robotics* provides a considerable amount of prior work, which focuses on emergent behaviours according to local interaction rules in (homogeneous) groups. Robots provide an excellent real-world test ground for validating biological behaviour models, for instance investigating the relationship between model parameters and resulting emergent structures, such as clusters of pucks [32], cleared areas [39] or walls [33, 52]. This line of research emphasizes simplicity and minimalism, which is reflected in the experimental setups. For instance, instead of aiming at engineering a system that builds a perfect wall, the objective would be to show that a small set of simple rules can produce wall-like structures. A recurring ingredient in many of the construction-related works in swarm robotics is stigmergy, the storing of information through the arrangement of entities in the world. Stigmergy is also found in works with other research foci, but stems from studies of social insects [8] and has repeatedly been shown to provide scalable coordination between robots.

Another line of research focuses on the *design of multi-robot algorithms to construct specific structures* based on principles such as Markov processes [20] or local rules and interactions [60, 55]. This problem has also been described as an inverse of that investigated by bio-inspired swarm robotics [61], and shares the heavy use of stigmergy [14] to simplify control. Moreover, researchers frequently point to similarities with work on modular and reconfigurable robots [49]. Real-world experiments in this area tend to be done on simplified versions of the algorithms and setups, with heavy use of simulation to study variations and affirm broader claims, although recent work with real robots presents promising results [40].

The implementation and control of the robot and the parts to be assembled are emphasized by *engineering-centric works*. Some of these target construction systems

[2] e.g. with a dedicated journal for automation in construction: `http://www.elsevier.com/locate/autcon`

for specific tasks, such as structure assembly in extraterrestrial settings [54,19], in orbit [7,58], or for assembling vertical trusses [27]. Some works explore how to integrate a human in the loop to handle contingencies [48]. With the same engineering orientation, architects [13] and the construction industry [10] have shown a long-lasting interest in automation and robotic approaches [50,63]. The foci of these works vary from robots doing specific tasks automatically, such as drilling [34], painting road lanes [62] or constructing structures by continuous deposition of concrete [23], to the automatic deployment and assembly of construction tools [22,50] and the high-level planning and scheduling of construction operations [18, 21]. In these works, construction itself is the research goal, and a strong presence of other robotic aspects is frequently encountered, such as human-robot interaction, coordination of heterogeneous teams, or advanced perception and control techniques.

Following on most of the work in this last category, we also emphasize robotic algorithms and representations. However, the underlying research questions differ: we are interested in determining the set of capabilities and their interplay required, in principle, for a system to exhibit physical environment interaction that may be considered intelligent. In contrast to bio-inspired works, we aim at building precise structures. In contrast to systems of multiple specific robots, we use a single robot with generic manipulation capabilities. Rather than attempting a detailed comparison with the relatively recent contributions previously cited, we trace the roots of our endeavour more directly to the pioneering work on Shakey [37], which reasoned and acted in its environment through on-board sensing and control. The fundamental questions have not really changed, but now the fruit of decades of research from the robotics community is available to us. This is reflected in our mix of methodologies, representations, algorithms, and the modern miniaturized technologies for sensing, actuation, and computation. Note that we conduct our work directly on a physical robot instead of first using a simulator: we are interested in the whole complexity of real-world interactions, and setting up a simulation with enough accuracy to produce results comparable with reality is a daunting task. Moreover, validating such a simulator requires substantial experimental trials to ensure that it behaves like the real world in all cases. These trials in turn require a physical robot. For this reason, and although many theoretical works have explored autonomous construction in simulation, we have decided to conduct our experiments directly in reality.

3 Experimental Setup

Our research interest in empirical studies of intelligent physical interaction has lead to the “real but contrived” experimental design shown in Figure 1. A robot follows high-level orders from a human, such as building a structure at a specific pose. The environment contains various traversable regions separated by ditches. At the beginning of the experiment, the robot does not know the shape of the environment. Moreover, the resources (building material) are scarce, and so the robot must use them parsimoniously. In particular, the robot might have to employ some resources to build bridges to gain access to more resources. As briefly mentioned in the introduction, we consider that such an adverse setup highlights some of the challenges linked to intelligence that a construction robot will face in the real world. In particular, it explores these points:

- Construction is a sustained process involving many operations. Each operation, such as the manipulation of physical objects, can fail in different ways. Failures must be handled at the control level, and the overall system must be sufficiently robust to deal with them.
- Resources are scarce and not usage-specific. The robot has to use them appropriately in a near-to-optimal way, thus precluding purely reactive behaviours: careful action planning is required to avoid deadlocks.
- The environment is initially unknown. All behaviours, including planning, must be grounded in on-board sensors and executed with on-board actuators.

This setup does not explore all the elements needed to build full-scale autonomous construction robots. In particular, it trades off real-world aspects (such as on-board sensing and limited computational power) with engineered scaffolding (e.g. a magnetic gripper and uniform resources) to simplify some of the sub-problems that are not central to the investigation.

The use of the marXbot miniature robot provides a safe and flexible experimental tool, which is of great importance when working on physical interaction with the world. Furthermore, a real-world application will have constraints of size, energy and processing power, which are also exhibited by this robot. Thus, such a small platform highlights the challenges of integration.

The following properties of the setup allow us to address the question of intelligent physical interaction with current technology in a laboratory room. Three types of objects must be mapped by on-board sensors: walls, ditches and resources. Proximity sensors can sense walls and resources, and ditch sensors detect the absence of ground. The distance scanner can only see walls (which provides disambiguation between walls and resources, but complicates sensor fusion). There is no depth sensor for the ditches, but there is a known relationship between their width and their depth. Resources are cubes of expanded polystyrene and have a known side length; they are always on the ground, which is known to be flat. They have a ferromagnetic ring around their lower part, which enables the robot to grasp them with a magnetic gripper. Furthermore, their top faces are known to be the brightest objects in the world, which eases visual detection. They also have magnets at their bottom and small metal plates on their tops, which allows them to self-align, self-assemble and stay connected. The ditches divide the ground into regions, all of which are larger than a known lower bound. The small size of the environment allows two further simplifications: First, the distance scanner has a range large enough to keep a wall in view, so loop closure is currently not an issue for our simultaneous localisation and mapping (SLAM) implementation. Second, any pair of regions shares a ditch, so topological path planning does not require graph search. As our focus is not on the performance details of SLAM or path planning, we just briefly discuss these (and other points) in Section 9.

4 Robotic platform

4.1 Hardware

We employ the marXbot robot, a miniature (170 mm in diameter) mobile robot, shown in Figure 2. The marXbot is a modular robot; our configuration has 4 modules:

1. The base [3] with 2 degrees of freedom (DOF) moves the robot over rough terrain and embeds a 38 Wh lithium polymer battery. This battery provides up to 7 hours of continuous operation. In addition, the base senses objects (walls and resources) within 5 cm using a ring of 24 proximity sensors. The base also detects ditches with a ring of 8 ground sensors.
2. The 3 DOF magnetic manipulator [45] grasps ferromagnetic objects. This module contains 6 proximity sensors for fine alignment and ditch detection when building bridges.
3. The 1 DOF rotating distance scanner [28] perceives walls at distances of up to 1 m.
4. The upper module [3] embeds an ARM computer based on a 533 MHz Freescale i.MX31 and 128 MB of RAM. This module also holds a colour camera, which provides vision.

One or more microcontrollers drive each module. They communicate using ASEBA, an event-based control architecture for microcontrollers [29]. The use of ASEBA allows a compact integration of many different features, with a clean architecture and a great flexibility of usage.

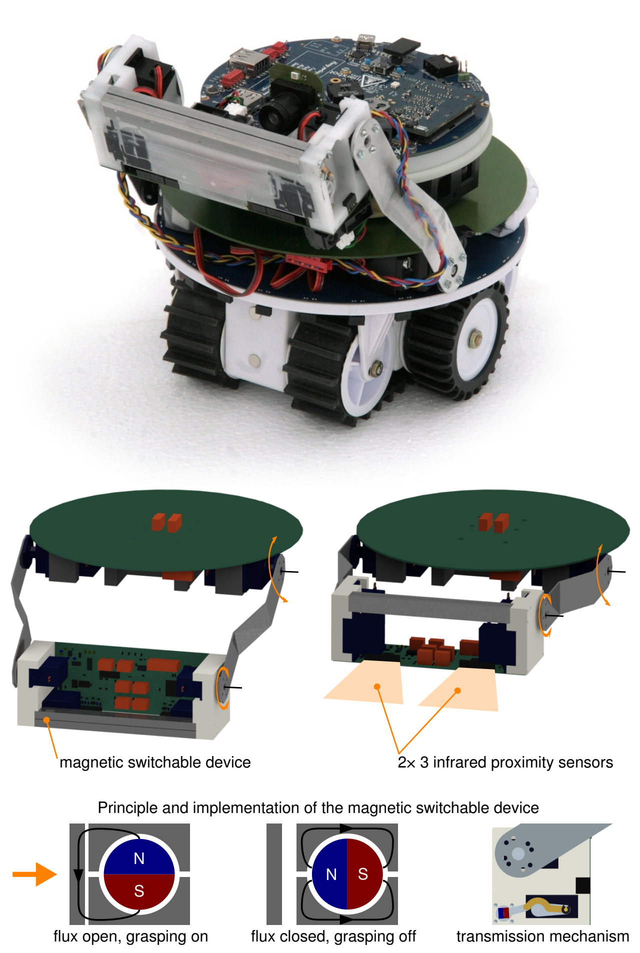

magnetic switchable device
2× 3 infrared proximity sensors
Principle and implementation of the magnetic switchable device
N
S
N
S
flux open, grasping on
flux closed, grasping off
transmission mechanism

Figure 2 The marXbot robot (top) and the details of the magnetic manipulator (bottom).

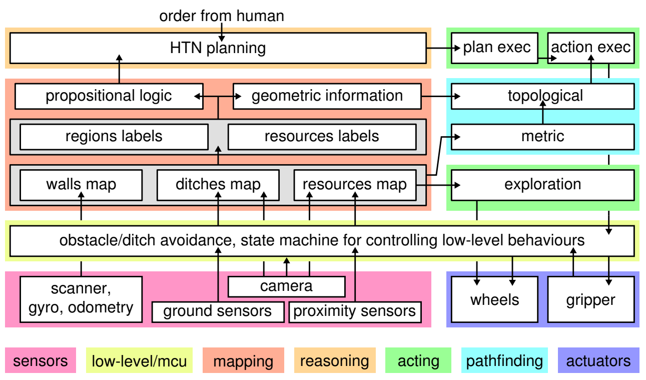


Figure 3 the system architecture

4.2 Software

Figure 3 shows the software architecture of the robot controller. At the low-level, ASEBA implements the reactive behaviours directly inside the microcontrollers.

These behaviours include the avoidance of obstacles and ditches, as well as the grasping and deposing of objects. This provides a short-term closing of the control loop and ensures a fast reactivity of the robot. The high levels implement two classical sense/plan/act architectures. The first manages path planning, and the second provides reasoning based on hierarchical task network (HTN) planning [35, Chapter 11]. We can interpret this scheme as an instance of a hierarchical control system [1].

For monitoring reasons, we have run most of the software on a remote desktop computer using a Wi-Fi link. However, we designed the software considering the available processing power of the robot itself, and thus the software could run in real time on the class of processors that the robot embeds.

5 Global perception

The rotating distance scanner of the robot provides ambiguous distance readings that we fuse together using a SLAM algorithm inspired by FastSLAM [28]. This algorithm builds an occupancy-grid map whose cells hold whether or not there are obstacles at the height of the scanner (9 cm). However, there is a richer variety of elements in the world than the scanner is able to capture. In particular, to build structures autonomously, the marXbot must also use other sensors. In this section, we show how to combine the outputs from the SLAM algorithm and the different sensors to build a representation of the environment.

As Figure 1 shows, the terrain consists of multiple regions separated by ditches of different widths and depths. All ditches are wide enough to fit a block into. As the robot does not have a distance sensor that is able to measure the depth of the ditches, it assumes that locations where ditch width is just over the edge length of a cube have a depth equal to the edge length of a cube, and therefore are suitable to build a bridge. This is a strong prior on the environment, and a depth sensor should be used in future research. However, although it requires work, adding such a sensor does not present extreme challenges, and thus we think that this strong prior does not impair the significance of this work. The robot uses this depth prior to plan bridges in locations where the distance between two regions is the closest. The environment also contains resources, which are 6 cm-cubes of expanded polystyrene.

The aim of the global perception subsystem is to create a propositional-logic representation of the world that is grounded in reality (see Figure 4). Indeed, in order to reason about the world, the robot needs a logical representation [46, Section 7.4]. However, to execute

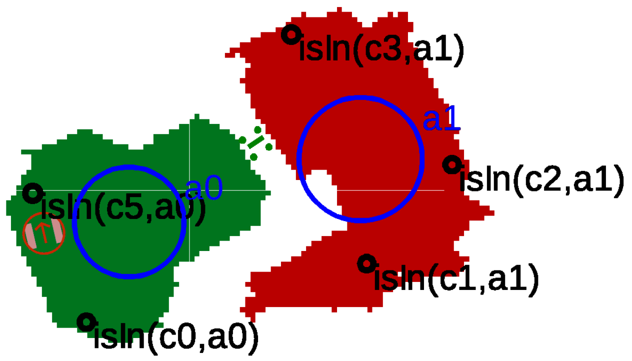


Figure 4 A screenshot of the software showing the representation of the environment. The solid shapes represent the two traversable regions, separated by the ditch. The small black circles represent the resources. The blue circles have their origin on the centres of mass of the regions and their radius are proportional to the areas of the regions. The green line with four points between the regions shows the pose of a potential bridge.

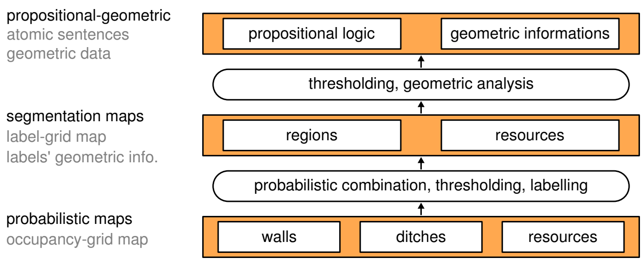


Figure 5 the three layers of the global perception process

decisions, the representation must be linked with geometric information. This leads to a dual propositional-geometric representation of the world in which logical atomic sentences are grounded in geometric data. We implement this representation process through three layers of increasing abstraction (see Figure 5). The first layer consists of three probabilistic maps of the walls, the ditches and the resources. For the second layer, we combine the maximum-likelihood estimations of the probabilistic maps to create two segmentation maps, one of the traversable regions and one of the resources. The third layer provides the dual propositional-geometric representation. We build the latter by analysing and filtering the segmentation maps to extract atomic sentences representing the regions, their connectivity, the resources, their associated regions and the robot itself.

This model shares similarities with related works on semantic maps [24, 64, 38]. However, probably because in these works robots do not manipulate the world, they do not acquire as much geometric information as our robot. Moreover, the software presented in these works is designed to run on laptop-level processors, so the authors are relatively free to use heavy computations, whereas in our case we take into account the limitations of our embedded processor, as described in the following subsections.

5.1 Probabilistic maps

The lowest layer of global perception consists of three probabilistic maps. These maps are two-dimensional occupancy-grid maps [6] of walls, ditches and resources. They have a spatial resolution of 2 cm, which we found sufficiently accurate, while fitting within our processing-power and memory constraints. Each cell holds the log-odds ratio $l(X) = \log(\frac{p(X)}{1-p(X)})$ where $p(X)$ is the probability of containing an element [56, p. 94, p. 286]. We store $l(X)$ with a resolution of 16 bits.

The probability of observing something with a sensor often depends on what has been sensed by another sensor. For instance, when an infrared proximity sensor of the base 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 of detecting a resource is the product of the probability of detecting an obstacle and the probability that this obstacle is not a wall: $p(\text{resource}) = p(\text{obstacle})\,(1 - p(\text{wall}))$. Despite its simplicity, this formula does not take a linear form when expressed as a log-odds ratio, given three binary random variables X, Y, Z such that $p(Z) = p(X)p(Y)$:

$$l(Z) = l(X) + l(Y) - \log(1 + e^{l(X)} + e^{l(Y)}) \tag{1}$$

This is unfortunate as computing logs and exponentials is slow. However, we observe that in practice, there are often cases in which probabilities for obstacles and walls are close to 0 or 1. Thus, we make the following approximation:

$$p(Z = 1) = p(X = 1)p(Y = 1) \\ \Rightarrow l(Z) \approx l(X) + l(Y) + l_{\mathrm{m}} \tag{2}$$

where l_{m} is the smallest possible log-odds–ratio value. This approximation is a smooth function that produces values close to the real formula around the extreme cases. Because it only performs two additions, it is very fast and thus well-suited for real-time processing on a miniature robot. In our implementation of sensor data fusion, we use the approximate function instead of the true function.

5.1.1 Sensing and data fusion

We create the three maps by fusing the data from different sensors. The ground and proximity sensors of the base provide short-range information, but with a high confidence and a quick refresh rate. The vision provides long-range information, but at a slower frequency and a lower confidence than proximity sensors. The wall map is the output of the distance scanner through the SLAM algorithm. The ditch 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 wall map and the outputs of the proximity sensors and the vision.

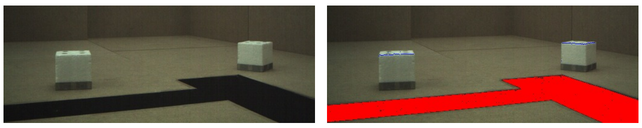


Figure 6 The vision on the marXbot. Left: the world as seen by the camera. Right: the result of the processing: in red, the ditches; in blue, the top lines of the resources. These show the lower half of the image. In the robot the horizontal dimension is further down-sampled by a factor of 4.

5.1.2 Vision

As Figure 6 shows, we use vision to locate resources and to detect ditches at long range (> 40 cm). This is possible because we know that the ground is flat and that the resources are cubes with a height of 6 cm and are always located on the ground.

The camera chip has a resolution of 2048×1536 pixels. As our probabilistic maps have a resolution of 2 cm, and the robot shakes slightly when it moves, we do not need such a high resolution. Thus we down-sample the image to 128×384 pixels by exploiting the hardware average filter of the camera chip. This permits a short exposure time while still capturing noiseless images. We process only the lower half of the image, as there are no objects of interest above the altitude of the camera.

We process the image from its bottom to its top. The detection of ditches is based on thresholding pixels[3]. Note that a point seen behind a wall or a resource is not a genuine piece of information about the presence of a ditch. Therefore, we back-project the classified pixels from the image to the world, assuming an altitude of 0 cm. Then, on the wall and the resource maps, we cast a line from the camera position to the back-projected coordinate. If any of these lines crosses a point with a probability of containing a wall or a resource larger than 0.5, we discard the point.

The detection of the resources is obtained by comparing pixels of the same column to find the top edges

[3] The ditch thresholding is the following: Let $\{r, g, b\}$ be a pixel and i be its intensity defined as $i = r + g + b$. This pixel might correspond to a ditch if $i < \mathrm{t}_{\text{intensity ditch}}$ and $|r - g| < \mathrm{t}_{\text{colour ditch}}$ where t_x are thresholding constants. Otherwise, the pixel might correspond to the ground.

of the polystyrene cubes in the image. In our model, these edges are the most intense parts of the image, as they are white and receive light from 75% of their sides. This model is inspired from the ambient-occlusion technique used in computer graphics [25]. 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 a top edge[4]. To detect the distance to the edge, we back-project this pixel from the image to the world, assuming an altitude of 6 cm. We then cast a line on the wall map to ensure that this pixel really corresponds to a visible resource.

5.2 Segmentation maps

From the three probabilistic maps, we create two segmentation maps, as seen in Figure 4. The first map is the region map. It is based on the fusion of the wall and the ditch maps:

$$p(\text{region} = 1) = (1 - p(\text{wall} = 1))\,(1 - p(\text{ditch} = 1)) \tag{3}$$

We consider a point for segmentation if its probability of being a traversable region, $p(\text{region} = 1)$, is close to one. We use the approximation from Equation 2 to compute this probability. We then apply a segmentation algorithm to create a list of regions. This algorithm also computes the area of each region and its centre of mass.

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

5.3 Propositional-geometric representation

Out of the two segmentation maps, we create a dual propositional-geometric representation of the environment. The propositional-logic part is well-suited for defining a state space for automated planning (see Section 6.2). We refer to real-world objects (regions, resources, robot) by unique constants, such as `a3` for a region or `r7` for a resource. We type these constants by unary predicates, such as `region(a3)` or `resource(r7)`. We relate real-world objects to each other using binary predicates, such as `isIn(r7, a3)`. These predicates describe the region connectivity: `isConnectable(`*region0*`,` *region1*`)` for potential bridges and `isConnected(`*region0*`,` *region1*`)` for actual bridges. They also indicate in which region a resource or the robot lies: `isIn(`*object*`,` *region*`)`. At the geometric level, we map these atomic sentences to geometric data, such as the centres and the areas of regions and resources or the pose of a potential bridge.

We only consider regions with a sufficiently large area ($> 200\,\text{cm}^2$), as our prior knowledge of the environment tells us that no tiny regions exist. This allows filtering out spurious regions in areas of uncertainty. To build the potential-bridge map (connectivity graph), we search for pairs of regions that have a small ditch between them. Because the regions might not be convex, we search for the closest points using a Monte-Carlo algorithm. This algorithm consists of drawing pairs of points following a distribution $\mathcal{N}(\text{centre}, \sqrt{\text{surface}}/2)$. If both points lie within their regions, the algorithm considers a line between the points and finds the intersections of this line with the region boundaries. The algorithm repeats this process a certain number of times and keeps the line with the smallest distance between the two regions as a potential bridge[5]. If this distance is small enough ($< 12\,\text{cm}$), we relate these regions with the predicate `isConnectable` and store the boundary points in the geometric part of the representation.

As with regions, we only consider resources with a sufficiently large area ($> 32\,\text{cm}^2$). We find the position of the resource in the region-segmentation map. If the resource is indeed located within a region, we add an `isIn` predicate between the resource and the region. We store the position of the resource in the geometric part of the representation. Finally, we find the region in which the robot lies and add a corresponding `isIn` predicate.

6 Reasoning

6.1 Pathfinder

The pathfinder is composed of two layers (light blue in Figure 3). The high-level layer allows the robot to go from one region to another. To find this path, the robot looks up the source and the destination regions in the potential-bridge table. Because it suffices for

[4] The resource thresholding algorithm is the following: Given a pixel $\{r, g, b\}$ of intensity i, if $i > \text{t}_{\text{intensity res.}}$ and $b + \text{t}_{\text{colour res.}} > r$ and $i > i_{\text{prev}} + \text{t}_{\text{res. delta}}$, this pixel might belong to the edge of a resource. Note that i_{prev} is the intensity of the last candidate pixel, and that t_x are thresholding constants.

[5] This algorithm is fast for relatively even and convex regions, for which most points drawn following a distribution $\mathcal{N}(\text{centre}, \sqrt{\text{surface}}/2)$ lie inside the region. Otherwise, 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 variant of A* [17].

our application, we have currently only implemented direct lookup when a potential bridge exists between two regions. Extending this with a search in the potential-bridge graph to find the path for two regions that are not directly connected is easy. The result of the high-level path-finding layer is a sequence of points. Note that the HTN planning domain enforces that the robot builds a bridge prior to crossing a ditch.

The low-level layer finds a path between two of these points. This may correspond to a path within a region or to the crossing of a bridge. This path avoids walls and unexplored regions. It also avoids resources if possible. The algorithm performs a wave-front propagation with variable propagation speed and a smooth front [41]. 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 (that is, one with a uniform distribution for wall probability) is not accessible. To prevent the robot from hitting walls when moving, we perform a morphological erosion [16] of this map (4 cells using 8-connectivity) to shrink the free space by 8 cm, which corresponds to the radius of the robot. If the robot does not try to cross a bridge, we also perform dilatation of the ditch map (2 cells using 4-connectivity) and resource map (2 cells using 8-connectivity). After erosion and dilatation, the maps are combined in a manner that ensures that a robot located half-way over a ditch will find a way out, and that the robot will avoid resources when possible. We set a very low propagation speed in the ditches, an intermediate one in the resources and a high one in the free space. Specifically, we first use the eroded free-space map to set a fast propagation in free space and a zero speed in the walls. Then, for each cell, if the dilated resource map holds a positive value, we set the speed to the minimum of the free-space map and an intermediate value. Finally, if the dilated ditch map holds a positive value, we reset the speed to the minimum of its previous value and a low value. Note that the propagation speed within the walls is zero, because the robot is not supposed to be inside them. Should this happen anyway, it is likely due to a bug, and the programme triggers an exception.

This two-layer pathfinder does not in general produce optimal global paths. For example, if a path traverses three regions, there might be more than one potential bridge connecting the first two regions of the path, with one closer to the third region than the others. In that case, choosing the closest one would lead to a shorter global path than choosing the remotest one. To implement such a global optimisation, we would need a unified pathfinder, such as one based on our current low-level layer. However, this is not trivial because we want to limit the number of bridge traversals, and we want the robot to traverse bridges perpendicularly to the ditches. In the current system, we ignore ditches and resources when we explicitly search for paths over bridges, and we position the robot perpendicularly to the ditch before every traversal. Because of these difficulties, we employ the two-layer approach.

6.2 Task planning

We want the robot to find a plan to fulfil the human's order autonomously. The order can be a move or a build order. For example, the human can order the robot to build a vertical structure, which requires 3 resources. Imagine that only 2 resources are readily available but 3 resources are available at a remote location. In this case, the robot could use these 2 resources to fill a ditch to access the remote location to fetch the 3 resources. 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 [35, Chapter 11], in our case Planner 9 [30]. The space of possible plans to solve a given construction task is constrained by the HTN *planning domain* for our construction scenario (Figure 7). We designed this domain around `connectRegions`, an HTN task that connects regions together. The idea is to connect a source region s to another region t, that might or might not be the destination d, and then to recurse from this newly-connected region t towards d. The `region-already-connected` alternative stops the recursion in case the two regions s and d are already connected or are the same. The two top-level tasks `moveRobot` and `buildStructure` call the `connectRegions` task with the destination and the resource regions as parameters. Note that subtasks are executed sequentially. 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 propositional representation of the world. Then we call Planner 9, and if it finds a plan, we execute it. This consists in sequentially executing the actions.

7 Actions

The robot performs two main activities: exploration and plan execution. When the experiment starts, the robot begins exploring, which allows it to build a representation of its environment. The resource and the ditch maps are only updated when the robot is exploring, because

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: ∅
```

Methods

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

fill-ditch(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) ∧ ¬same(res1, res2)
            ∧ ¬same(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) ∧
            ¬same(res1, res2) ∧ ¬same(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 HTN planning domain for autonomous construction. Section 7.2 describes the actions (`take`, `fill1`, etc.).

when the robot is transporting a resource, the latter occludes the camera and the proximity sensors of the base directed forward. 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 task planning. If the planning succeeds, the robot stops exploring and executes the plan; otherwise it continues exploring. The execution of the plan consists in performing each action sequentially. An action in the HTN planning sense corresponds to a low-level behaviour implemented in ASEBA. Prior to executing an action, the robot moves to a specific position in the relevant region. To do so, it uses the two-layer pathfinder algorithm presented in Section 6.1. After executing a plan, the robot switches back to exploration.

7.1 Exploration

The exploration algorithm needs only to provide a good exploration, as it delegates safety considerations to the microcontrollers. The exploration behaviour of the robot aims at providing efficient coverage at a low computational cost. To do so, the robot has an ordered list of relative points (Figure 8). We have selected these points

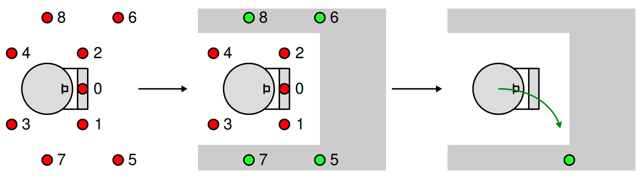


Figure 8 The exploration strategy, grey area is explorable.

based on preliminary experiments. 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 ditch map contains a low certainty for this point. Moreover, the point should not be on the other side of a wall or a resource, considering the current position of the robot. This strategy drives the robot towards unexplored areas and leads to a good ditch map, which is essential for reasoning about the topology of the environment. 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

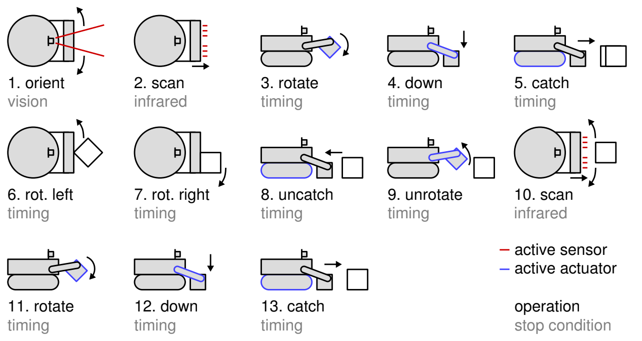


Figure 9 The movement sequence to grasp a resource with precise alignment.

1 Hz. Note that at the low level, the microcontrollers running ASEBA implement obstacle and ditch avoidance autonomously. This behaviour consists of two vector-field avoidance algorithms [4]. If the robot perceives a ditch or an obstacle, it avoids it; otherwise it performs exploration. Ditches have priority over obstacles, and as avoidance consists in turning on the spot, the ditch avoidance behaviour prevents the robot from hitting an obstacle.

7.2 Plan execution

We implement the low-level behaviours corresponding to HTN actions by using ASEBA. The plan executor starts a low-level behaviour using a specific event, and the low-level behaviour also informs the executor of its execution result (success or failure) through an event.

7.2.1 move

This behaviour consists in moving to a given position in a destination region. The region identifier is available in the propositional part of the representation of the world and the position in the geometric part.

7.2.2 take

This behaviour consists in accurately grasping a resource. The precision is critical because a misaligned resource would add errors to all subsequent operations. Based on preliminary experiments, we have chosen to grasp resources in two stages to ensure their precise alignment on the manipulator.

Figure 9 shows the numbered movement sequence, and this text refers to its steps. First, the robot turns to face the resource using the camera (1). We compute the position of the resource in the image by taking the median of the position of pixels whose intensities are above a given threshold on the lowest horizontal line of the image. We consider a pixel if its intensity is higher than 80 % of the maximal intensity, renormalised. The robot turns using an on/off controller. Once the resource is centred in the image, the robot goes forward (2) and grasps the resource (3–5). It then rotates leftward (6) and rightward (7) to ensure a firm grip. Indeed, if the resource was attached by only one of its corners, these rotations would allow the gripper to grasp one of its sides. At that point, the resource is securely grasped, but it is certainly misaligned. Thus, the robot ungrasps the resource and moves back (8). It then moves forward again (9) while using the infrared sensors of the magnetic manipulator to align the resource at the centre of the manipulator (10). 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 hysteresis. When the robot is close enough to the resource, it grasps it again (11–13), and this time the resource is well aligned. We have found this control policy to produce a precise grasping. To allow scanning for ditches while holding a resource, 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, which explains the steps (3–5) and (11–13).

7.2.3 fill

This behaviour consists in dropping a resource into a ditch to build a bridge to the opposite region. To build a safe passage for the robot, a bridge requires two resources side by side. The HTN planning domain ensures that this behaviour is called only when the robot is holding a resource. 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, and at the left for `fill2`.

The robot goes towards the ditch 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 ditch. When the two infrared sensors see the ditch, the robot goes back for a short distance, then stops and drops the resource. 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 resource close to the border of the ditch.

7.2.4 `build`

This behaviour consists in building a tower of up to three resources. The HTN planning domain ensures that this behaviour is called only when the robot is holding a resource. There are three versions of this behaviour, `build1`, `build2` and `build3`, which are used to place the base resource for the tower, the middle resource and the top resource, respectively. Putting the base resource is easy, the robot simply goes forward for a short distance, then stops and disengages its magnetic switchable device. It then lifts its manipulator slightly to ensure that the resource is well detached, and then it goes back for a short distance. Putting the middle and the top resources first requires an orientation using the camera, as described in Section 7.2.2. The robot then raises its manipulator, goes forward and scans for the previous resource using the infrared proximity sensors of its magnetic manipulator. If the sum of the intensities of the two sensors close to the middle is above the threshold, the robot stops and disengages its magnetic switchable device. As the resources have small magnets at their bottoms and little ferromagnetic plates on their tops, the new resource self-aligns and self-assembles with the existing resource. Finally, the robot goes back for a short distance. The difference between adding the middle and the top resource lies in the height to which the robot raises its manipulator.

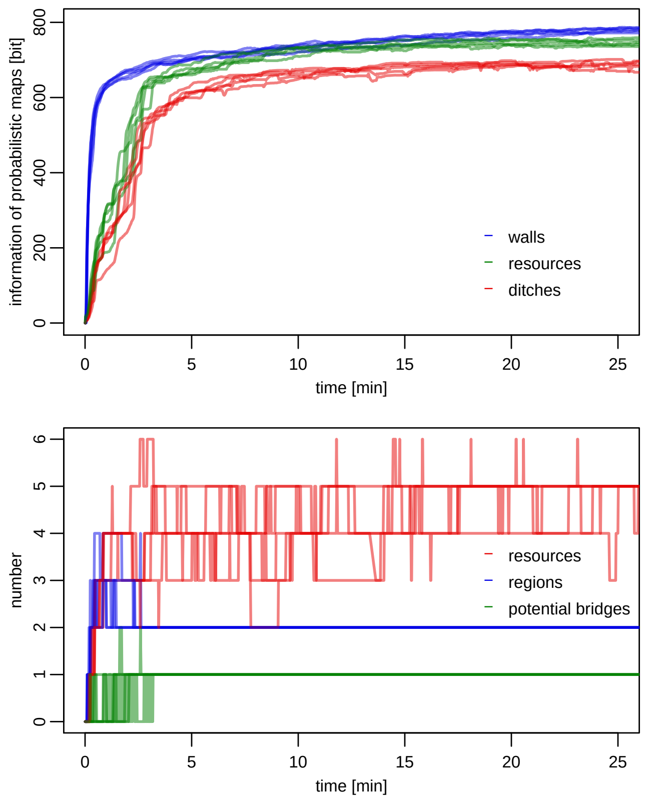


Figure 10 The evolution of the environment representation over time. These plots overlay the curves from 5 runs. Top: the quantity of information contained in the different probabilistic maps. Bottom: the number of grounded objects per type. In reality, there are 5 resources and 2 regions close enough to build a bridge between them.

8 Results

8.1 Representation

To analyse the performance of our representation system, we conducted 5 experimental runs of 25 minutes each in the environment shown in Figure 1. At the start of every run, we placed the robot in the centre of the small region (the one with two resources). The robot faces the opposite region. We keep at least 10 cm between the robot and the walls and the ditches, as the sensors self-calibrate at the beginning of a run.

To understand how the representation of the world evolves over time, we analysed the entropy of the probabilistic maps and the number of grounded objects over time. We took one measurement every second. We are interested in the quantity of information in the probabilistic 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. This difference is positive, because the knowledge about the world grows over time while the robot explores. We use a binary logarithm to compute the entropy; therefore the unit of information is bit. For the grounded objects, we recorded the number of regions, resources and potential bridges, which are found in the propositional-geometric representation.

Figure 10 overlays the evolution of the representation over time for the 5 runs. The top plot shows how much information the robot holds about the world in its probabilistic maps. These values correspond to the subjective view from the perspective of the robot, not to a ground truth. 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 near and far, in addition to scanning around the robot. In 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, because it uses a sensor model to update the probabilistic map. This prior allows the wall map to cover a larger region than the resource map does. We also see that the ditch map holds less information than the resource map. We believe that this is due to the shaking induced by the tracks, combined with the fact that the camera always sees a large patch of the ground, causing the sides of the ditches to move in the image, thus destabilizing their position. Moreover, the SLAM is not perfect. In particular, it might slightly stretch its map compared to reality. Thus, when using its camera compared to its ground sensors, the robot sees the ditches at slightly different positions. On the sides of the ditches, these two sensors return contradictory information, which increases the entropy of the ditch 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 to the parameters of the SLAM algorithm.

In the bottom plot of Figure 10, we see that the number of regions and bridges quickly reaches the correct value in all runs. The number of resources is also approximately correct, although it is less stable. On average over all runs, between 20 and 25 minutes the robot detected the right number of resources 72 % of the time (1008 correct detections for 1394 measurements). The perception of the resources is imperfect because, as shown in Figure 1, three resources are in the remote region 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 perceived distance to the resource, because the coordinate transform divides a constant by the pixel position. Second, as the robot turns, the angular position of the object with respect to the robot frame changes very quickly. The image and the odometry are temporarily unsynchronized, which results in errors to the perceived horizontal position of the resource. These problems also affect the perception of the ditches, but as the ditches are larger than the resources, these errors do not disturb the perceived topology of the world, even if they increase the entropy of the ditch map.

8.2 Autonomous construction

We tested the construction application in the real-world setup presented in Figure 1. A video of this experiment is available online[6].

We ran several iterations of three tasks. The first two validated the two types of construction–a bridge and a tower–and the last one combines both. To validate the bridge, we ordered the robot to move to the opposite region. To validate the tower, we ordered the robot to build one in its current region, in which there were enough resources. To validate the combination, we ordered the robot to build a tower in its region, in which there were not enough resources. Table 1 (top) shows the solution plans for these tasks, with constants corresponding to the example in Figure 4. In all tasks, we initially let the robot explore for a while until we saw that its representation of the environment was stable. This exploration phase lasted 5 to 10 minutes. We have ran each task 10 times. Figure 11 shows the sequence of images of a successful run (third task). Table 1 (bottom) shows the success rate and the average duration of the different tasks. We took into account only successful task executions to compute the average duration.

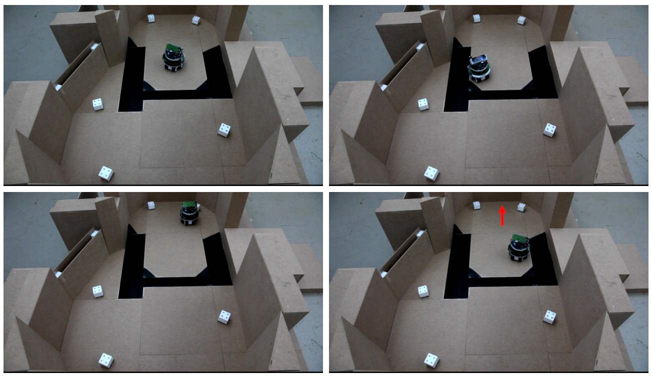
The first task consists in moving from the small region (the one with two resources) to the large one (the one with three resources). To fulfil this task, the robot must build a bridge. At the beginning of this task, we placed the robot in the centre of the small region, facing the opposite region. This task fully succeeded 9 times out of 10 trials, and was partially successful the remaining run when the robot dropped the second resource aside instead of into the ditch. However, when the robot then moved, it pushed the resource into the ditch and managed to cross successfully.

The second task consists in building a tower in the large region, using the three resources available in this region. At the beginning of this task, we placed the robot in the centre of the large region, turning its back on the ditch. This task fully succeeded 9 times out of 10 trials, and was partially successful the remaining run when the robot put the third and last resource of the tower half onto the second resource. It thus built a tower but not a straight one.

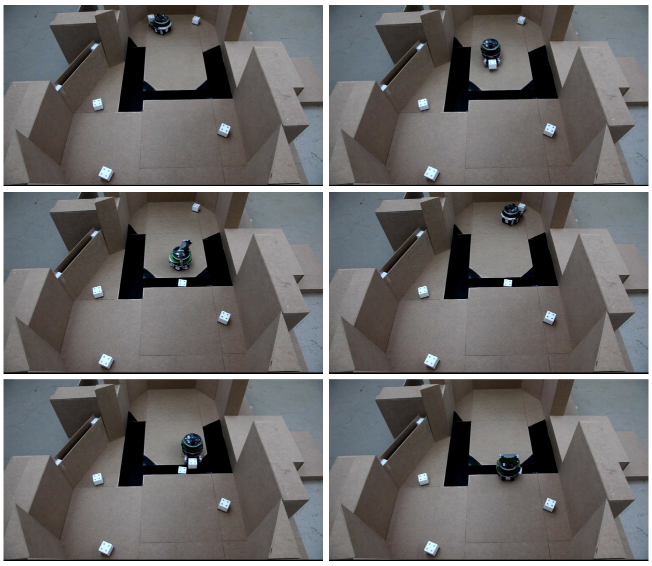
The third task consists in building a tower in the small region, starting from that region. Because that region holds only two resources, the robot must first employ them to build a bridge towards the large region, and use the three resources there to build the tower in the small region. At the beginning of this task, we placed the robot in the centre of the small region, facing the large region. This task fully succeeded 8 times out of 10 trials. In one trial, the robot fell into a ditch just before executing `fill2`, because of the imperfection of the world perception. In another trial, 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 able to grasp one of its side.

[6] http://www.youtube.com/watch?v=h865RHbT9Ms

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:



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

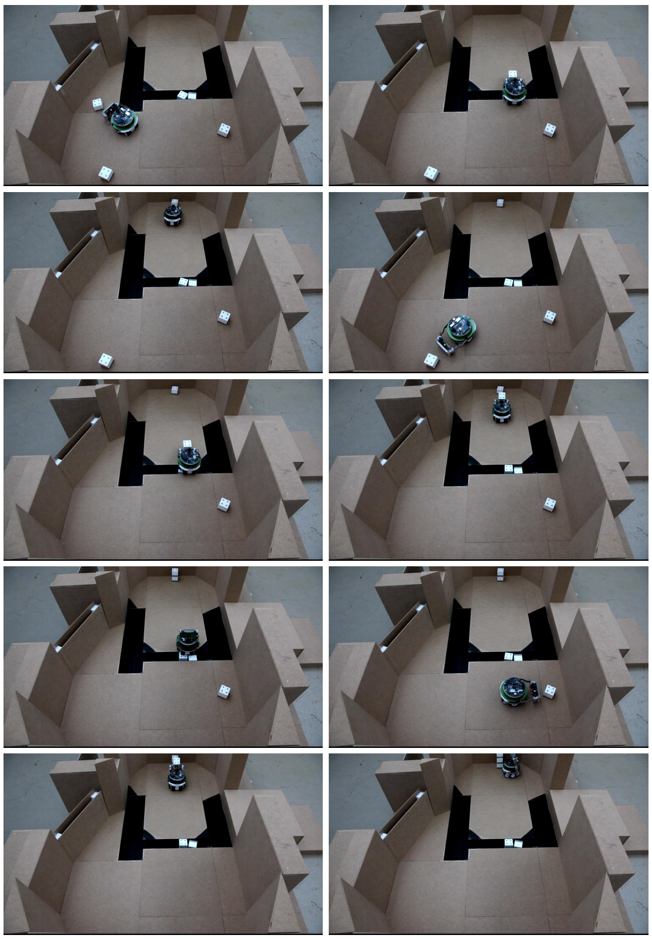


Figure 11 Image sequence of a successful construction.

bridge (local res.)	tower (local res.)	bridge + tower (remote res.)
`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)`

task	mean dur.	success rate
bridge	100 s	9.5/10 (1 partial success)
tower	286 s	9.5/10 (1 partial success)
bridge+tower	726 s	8/10

Table 1 Experimental tasks. Top: solution plans. Bottom: mean duration and success rate over 10 runs.

9 Lessons learnt and future work

In this section, we share the lessons we learnt by implementing and validating the autonomous-construction application, and based on these we propose future work.

9.1 On physical constraints

The integration of many features in a miniature robot leads to the optimisation of competitive constraints.

There is a hardware conflict between sensing and acting, as both demand access to the front of the robot. Indeed, vision requires empty space, but one must also fit a manipulator within this space. This is a commonly encountered design challenge in complex robots where multiple sensors and actuators must share the same physical space, but it is exacerbated in tasks like construction.

Hardware constraints are due to the sparse sampling of the space of possible hardware devices by existing products. For example, in our robot, the camera provides perception of ditches 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 because the size of the arena forces the robot to explore its surroundings in detail. This operation demands a lot of time and does not take the best advantage of the range-sensing capabilities of the camera. However, production constraints prevent us from improving the sensor placement, and we did not find any commercially-available wider-angle lens suited to our camera hardware. As there is a gap between the perception range of the camera and that of the proximity sensors, the robot must compensate by using a conservative exploration strategy. This strategy aims at ensuring dense ditch and resource maps, despite the limitations of the hardware. This example shows that sensor properties strongly affect the algorithms to use for perception and control. This might be one of the limiting factors in the development of advanced real-world applications, where the availability of sensors and the requirements of high-level control do not always coincide.

9.2 On the software architecture

The robot controller is a complicated piece of software because many modules interact with each other to form a complex network. This is a problem, because when programming a module, one must consider the other modules at the level of both the data flow and the temporal constraints. For instance, the state machines for action execution can become very complex if one has to consider potential failures at any level. Frameworks such as ROS [43] help in coping with the structural complexity. We have not employed ROS in this project because at the time we started, it was not ready for production use, especially on ARM platforms. Moreover, on resource-limited platforms, such component-based frameworks have significant overheads compared to single programs. In future work, ROS or a similar framework may be a suitable choice, as nowadays platforms are more powerful.

Current frameworks do not tackle the temporal complexity arising from many interacting components. However, there are promising ways to address this problem [44, 15]. In addition, to implement complex actions, an interesting research direction could be to define actions sequentially, execute each of them in its own thread and use exceptions to handle failures. This could be implemented using the semantics of continuations [53]. Recently, scripting languages such as Python[7] [57] have provided this feature. We think that this aspect of action programming will become important when versatile action capabilities are integrated in a single mobile robot.

9.3 On robustness

A complex application such as autonomous construction has multiple points of failure. These failures are not only binary and local, but can propagate from one step to another. For instance, the robot might incorrectly grasp a resource, such that the resource would jam itself in the ditch when crossing a bridge, which could make the robot fall. Thus, it is important to detect and correct faults early.

Faults can happen not only in actions but also in perceptions. One way to detect a fault in action is to compare the sensor values resulting from the action with those expected after a success. Experimental evidence has shown that humans employ this strategy to detect failure of predicted actions [9]. Another way is to build compliant behaviours, as we did for grasping resources (see Section 7.2.2). Faults in perception can be corrected by using various principles. One principle is that the world state tends to be constant over time and to use this information to correct the perception. For instance, the loop closing in SLAM [51] is based on this principle. In our application, the environment is small enough such that we do not need to perform this operation. One can also correct faults using prior world knowledge. We perform this correction when we put a threshold on the area of a resource to create a symbolic constant.

9.4 On artificial-intelligence architecture

In this experiment, we implemented reasoning using Planner 9, a generic HTN planner that we developed in a previous work [30]. Using an HTN planner brings flexibility to the reasoning and allows the combining of elements of knowledge that are described independently. In this experiment, the topology of the world is simple, and thus the execution time of the planner is negligible compared to the time taken by the perception subsystem. As the paper on Planner 9 shows [30], the latter can solve more complex problems than the experimental scenario of this study can. Clearly, in this application, the difficulty of perceiving the world and implementing the behaviours on a physical robot shadows the interest of such a powerful reasoning engine. In a more general

[7] Continuations are available in the *stackless* version of Python, see `http://www.stackless.com` or through the `yield` keyword in normal Python.

context, one can imagine a rich world with many entities, leading to large maps with many symbolic constants. Under such conditions, the planner's computational load might again become significant. Nevertheless, we think that in many real-world applications, there are only a limited number of entities relevant to the planner at a given moment. Therefore, there are ways to prune most of the entities to limit the search space. For instance, typing removes all entities of the wrong type. Moreover, one might use a salience mechanism, such as considering only physically close entities, to further reduce the search space. For these reasons, we think that the factors limiting the intelligence of a robot are not the AI algorithms but the robustness of the perception (symbol grounding in particular), the precision of the actions, and the detection of their outputs.

9.5 On the representation

The dual propositional-geometric representation successfully allowed our robot to reason in an abstract model of the world, and to link the resulting plan to physical objects with geometric properties. This highlights the importance of defining the representation, which in our case is atomic sentences referring to two-dimensional geometric shapes.

In general, choosing a propositional-geometric mapping is not trivial, as there are many ways to express the same reality. For instance, reasoning about the number of resources in a region could be done using a numeric fluent, `numberOf(resource, region)`. In this application, we decided to use one constant per physical resource because it eased the implementation of the planning domain. In general, creating a domain for a real-world problem is a hard task. This has prompted the implementation of algorithms that learn such domains from traces of plan executions [65,36]. However, the application of such methods to robotic problems remains to be demonstrated (preliminary results can be found in [42]).

Currently, we discard the existing constants when we create the propositional 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, such as when two regions merge as a result of the exploration by the robot. We could handle this case explicitly by directly comparing the old and the new label information on the old and the new segmentation maps, and fuse the atomic sentences accordingly.

Our symbol-grounding process uses several thresholding constants, such as the minimal size of a region or the maximal width of a ditch allowing the robot to build a bridge. We chose these constants in function of our knowledge of the robot hardware and some preliminary experiments. It would be interesting to allow the robot to learn some of these constants. However, this would be difficult because it would require low-level safety procedures, such as using the accelerometer to trigger a back move when the robot begins to fall. Another solution would be to learn these constants from a simulation, but setting up the simulation would be a major work in itself.

9.6 On localisation

The current system implements SLAM using only the rotating distance scanner and the odometry/gyroscope. However, we could use other sensors, such as the camera or even the ground infrared sensors, to contribute information to the SLAM algorithm. However, 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 [2]. 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 the use of various features and the spatial relations between them. Recent studies have explored the use of vision [5] and range data [2], and some have shown how to take the odometry readings into account [47]. Finally, symbolic topological SLAM is close to what we currently think is the implementation of path integration and mapping in mammals [31]. On the other hand, metric maps provides the user with a nice representation with which they can interact. It would be interesting to explore novel types of interactions from the user, such as pointing using a laser or a colour patch, and to see whether this allows for topological localisation. If so, it would be interesting to compare this type of localisation with the metric one, in the context of a demanding application, such as autonomous construction.

9.7 On the development process

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 visual inspection of the intermediate steps of the different algorithms. Finally, it is important to overlay the dual propositional-geometric representation over the probabilistic and the segmentation maps, which allows easy validation of the correctness of the representation.

Aseba provides logging and replay tools, which allow the re-creation of 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 is possible to rerun the experiment at several times the speed of real time, thus increasing the efficiency of finding bugs and implementing features. The camera programme cannot log in sync with aseba, but by taking snapshots we managed to make it work fairly easily. However, we are convinced that for complex applications, a logging and replay architecture is of paramount importance. Modern software frameworks for robotics provide and emphasise such capabilities.

Finally, the development of such a complex application requires knowledge in many fields, such as mapping, exploration, symbol grounding, task planning and action execution. These are not easy to combine, albeit integration projects such as ros facilitate this by providing readily usable bricks.

9.8 On exploration vs execution

In this experiment, we issued task orders only when we considered that the perception of the world was correct. Even so, we noticed that failures, in particular for the `take` action, tended to occur when the perception quality was 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, would produce an erroneous grounding and thus an erroneous plan. A more robust solution would be to wait for the symbol-grounding output to stabilise and to start planning and execution only when this output has not changed for a certain duration. We could also make the planning more robust for environmental changes, such as using continual planning [11]. However, in this scenario it would be a hack to work around the limitations of the sensors, as the robot and its constructions are the only dynamic elements in the environment. Moreover, it would be more difficult to design the planning domain, because planning and action execution are more interwoven than in a simple htn planner. This interweaving does not increase the expressivity of planning, but it does make the planning process more reactive.

10 Conclusion

This paper presents an empirical study that sheds light on the capabilities required for intelligent physical interaction with the real world. Our system leverages modern algorithms and representations as well as the recent increase in miniaturized computational resources. A few application-specific simplifications allow the use of state-of-the-art perception and planning approaches on a miniature robot with tight computational constraints. This provides a small experimental environment for autonomous construction, which avoids the pitfalls of simulation-based studies.

From an application-oriented point of view, this paper is a case study involving integration and system engineering. We presented implementation details for the various components, described their coordinated interplay and experimentally validated system performance. The robot performed its tasks successfully 80% of the time, and we discussed lessons learnt and future work that would improve performance and extend the proposed approach to a wider range of domains. Our practical contribution lies in a novel combination of world modelling, symbolic-geometric reasoning and sensori-motor behaviours for construction.

From a more conceptual point of view, we endeavour to understand the essential ingredients for intelligent behaviour. In this context, we trace our roots back to the advent of mobile robotics and ai, and postulate that modifying one's environment is an essential part of intelligence. Here, our contribution stems from the application of modern methods to a question that continues to accompany robotics research even after four decades. The challenges that we have designed into the test scenario reflect this deeper interest: the experimental setup requires robust real-world action execution, careful planning of the use of scarce resources, and reasoning that is in real time and grounded in on-board perception. We have shown that a mixture of continuous and symbolic representations and methods over several components in a hybrid system architecture can indeed lead to flexible, robust, goal-driven behaviour in novel situations.

11 Acknowledgements

The authors thank Francis Colas for proofreading the manuscript. We thank Michael Bonani, Valentin Longchamp, Philippe Rétornaz, Florian Vaussard, Daniel Burnier and Tarek Baaboura for their contribution to the marXbot robot. We thank Frédéric Rochat, Patrick Schoeneich, Thierry Barras and Pierre Noirat for their work on the magnetic manipulator. We thank Martin Voelkle for his contribution to Planner 9. We thank Basilio Noris and Emmanuel Eckard for their artistic contributions in relation to this work. We are also grateful for the wealth of feedback provided by the anonymous reviewers. They allowed us to improve the presentation of the related work significantly, thus better situating our contribution in a wider context.

References

1. Albus, J., Barbera, A., Nagel, R.: Theory and practice of hierarchical control. In: Proceedings of the 23rd IEEE Computer Society International Conference, pp. 18–39. IEEE Press (1981)
2. Beeson, P., Modayil, J., Kuipers, B.: Factoring the mapping problem: Mobile robot map-building in the Hybrid Spatial Semantic Hierarachy. The International Journal of Robotics Research **29**(4), 428–459 (2010). DOI 10.1177/0278364909100586
3. Bonani, M., Longchamp, V., Magnenat, S., Rétornaz, P., Burnier, D., Roulet, G., Vaussard, F., Bleuler, H., Mondada, F.: The MarXbot, a Miniature Mobile Robot Opening new Perspectives for the Collective-robotic Research. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010). IEEE Press (2010)
4. Borenstein, J., Koren, Y.: The vector field histogram-fast obstacle avoidance for mobilerobots. IEEE Transactions on Robotics and Automation **7**(3), 278–288 (1991)
5. Cummins, M., Newman, P.: FAB-MAP: Probabilistic localization and mapping in the space of appearance. The International Journal of Robotics Research **27**(6), 647–665 (2008)
6. Elfes, A.: Using Occupancy Grids for Mobile Robot Perception and Navigation. Computer **22**(6), 46–57 (1989). DOI 10.1109/2.30720
7. Everist, J., Mogharei, K., Suri, H., Ranasinghe, N., Khoshnevis, B., Will, P., Shen, W.M.: A system for in-space assembly. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2004)
8. Franks, N., Deneubourg, J.: Self-organizing nest construction in ants: individual worker behaviour and the nest's dynamics. Animal Behaviour **54**(4), 779–796 (1997)
9. Förster, K., Biasiucci, A., Chavarriaga, R., Millán, J.d.R., Roggen, D., Tröster, G.: On the use of brain decoded signals for online user adaptive gesture recognition systems. In: Pervasive 2010 The Eighth International Conference on Pervasive Computing (2010)
10. Gambao, E., Balaguer, C., Gebhart, F.: Robot assembly system for computer-integrated construction. Automation in Construction **9**(5-6), 479–487 (2000). DOI 10.1016/S0926-5805(00)00059-5
11. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: theory and practice. Morgan Kaufmann Publishers (2004)
12. Gottfredson, L.: Mainstream science on intelligence: An editorial with 52 signatories, history, and bibliography. Intelligence **24**(1), 13–23 (1997)
13. Gramazio, Kohler: Structural oscillations. http://www.dfab.arch.ethz.ch/web/d/forschung/142.html (2007–2008). Installation at the 11th Venice Architectural Biennale
14. Grushin, A., Reggia, J.: Stigmergic self-assembly of prespecified artificial structures in a constrained and continuous environment. Integrated Computer-Aided Engineering **13**(4), 289–312 (2006)
15. Hager, G., Peterson, J.: Frob: A transformational approach to the design of robot software. In: In Robotics Research: The Ninth International Symposium (1999)
16. Haralick, R., Sternberg, S., Zhuang, X.: Image analysis using mathematical morphology. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) **9**(4), 532–550 (1987)
17. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. Systems Science and Cybernetics, IEEE Transactions on **4**(2), 100–107 (1968). DOI 10.1109/TSSC.1968.300136
18. Hsie, M., Chang, C.J., Yang, I.T., Huang, C.Y.: Resource-constrained scheduling for continuous repetitive projects with time-based production units. Automation in Construction **18**(7), 942–949 (2009). DOI 10.1016/j.autcon.2009.04.006
19. Huntsberger, T., Pirjanian, P., Trebi-Ollennu, A., Nayar, H., Aghazarian, H., Ganino, A., Garrett, M., Joshi, S., Schenker, P.: 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)
20. Jones, C., Mataric, M.J.: Automatic synthesis of communication-based coordinated multi-robot systems. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2004)
21. Jun, D.H., El-Rayes, K.: Optimizing the utilization of multiple labor shifts in construction projects. Automation in Construction **19**(2), 109–119 (2010). DOI 10.1016/j.autcon.2009.12.015
22. Kang, S., Miranda, E.: Planning and visualization for automated robotic crane erection processes in construction. Automation in Construction **15**(4), 398–414 (2006). DOI 10.1016/j.autcon.2005.06.008. The first conference on the Future of the AEC Industry (BFC05)
23. Khoshnevis, B.: Automated construction by contour crafting—related robotics and information technologies. Automation in construction **13**(1), 5–19 (2004)
24. Kuipers, B.: The Spatial Semantic Hierarchy. Artificial Intelligence **119**(1–2), 191–233 (2000)
25. Landis, H.: Production-ready global illumination. In: Siggraph Course Notes, vol. 16, pp. 87–101. ACM (2002)
26. Legg, S., Hutter, M.: A collection of definitions of intelligence. In: Proceeding of the 2007 conference on Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms: Proceedings of the AGI Workshop 2006, pp. 17–24. IOS Press (2007)
27. Lindsey, Q., Mellinger, D., Kumar, V.: Construction of cubic structures with quadrotor teams. In: Proceedings of Robotics: Science and Systems. Los Angeles, CA, USA (2011)
28. Magnenat, S., Longchamp, V., Bonani, M., Rétornaz, P., Germano, P., Bleuler, H., Mondada, F.: Affordable SLAM

through the Co-Design of Hardware and Methodology. In: Proceedings of the 2010 IEEE International Conference on Robotics and Automation, pp. 5395–5401. IEEE Press (2010)
29. Magnenat, S., Rétornaz, P., Bonani, M., Longchamp, V., Mondada, F.: ASEBA: A Modular Architecture for Event-Based Control of Complex Robots. IEEE/ASME Transactions on Mechatronics **PP**(99), 1–9 (2010). DOI 10.1109/TMECH.2010.2042722
30. Magnenat, S., Voelkle, M., Mondada, F.: Planner9, a HTN planner distributed on groups of miniature mobile robots. In: Intelligent Robotics and Applications, Proceedings of the Second International Conference on Intelligent Robotics and Application, *Lecture Notes in Computer Science*, vol. 5928, pp. 1013–1022. Springer (2009). DOI 10.1007/978-3-642-10817-4
31. McNaughton, B., Battaglia, F., Jensen, O., Moser, E., Moser, M.: Path integration and the neural basis of the'cognitive map'. Nature Reviews Neuroscience **7**(8), 663–678 (2006)
32. Melhuish, C., Holland, O., Hoddell, S.: Collective sorting and segregation in robots with minimal sensing. In: Proceedings of the fifth international conference on simulation of adaptive behavior on From animals to animats, vol. 5, pp. 465–470 (1998)
33. Melhuish, C., Welsby, J., Edwards, C.: Using templates for defensive wall building with autonomous mobile ant-like robots. In: Proceedings of Towards Intelligent Autonomous Mobile Robots, vol. 99 (1999)
34. Molfino, R.M., Razzoli, R.P., Zoppi, M.: Autonomous drilling robot for landslide monitoring and consolidation. Automation in Construction **17**(2), 111–121 (2008). DOI 10.1016/j.autcon.2006.12.004
35. Nau, D., Ghallab, M., Traverso, P., Corporation, E.: Automated Planning: Theory & Practice. Morgan Kaufmann Publishers (2004)
36. Nejati, N., Könik, T., Kuter, U.: A goal- and dependency-directed algorithm for learning hierarchical task networks. In: K-CAP, pp. 113–120 (2009)
37. Nilsson, N.J.: Shakey the robot. Technical Note 323, Artificial Intelligence Center, Computer Science and Technology Division, SRI International (1984)
38. Nüchter, A., Hertzberg, J.: Towards semantic maps for mobile robots. Robotics and Autonomous Systems **56**(11), 915–926 (2008)
39. Parker, C.A.C., Zhang, H.: Collective Robotic Site Preparation. Adaptive Behavior **14**(1), 5–19 (2006). DOI 10.1177/105971230601400101
40. Petersen, K., Nagpal, R., Werfel, J.: Termes: An autonomous robotic system for three-dimensional collective construction. In: Proceedings of Robotics: Science and Systems. Los Angeles, CA, USA (2011)
41. Philippsen, R.: A Light Formulation of the E* Interpolated Path Replanner. Tech. rep., Autonomous Systems Lab, Ecole Polytechnique Federale de Lausanne (2006)
42. Philippsen, R., Nejati, N., Sentis, L.: Bridging the gap between semantic planning and continuous control for mobile manipulation using a graph-based world representation. In: 1st International Workshop on Hybrid Control of Autonomous Systems (HYCAS) held in conjunction with the International Joint Conference on Artificial Intelligence (IJCAI). Pasadena, California, USA (2009)
43. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: ROS: an open-source Robot Operating System. In: International Conference on Robotics and Automation. IEEE Press (2009)
44. Ramanathan, G., Morandi, B., West, S., Nanz, S., Meyer, B.: Deriving concurrent control software from behavioral specifications. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1994–1999. IEEE (2010)
45. Rochat, F., Schoeneich, P., Bonani, M., Magnenat, S., Mondada, F., Bleuler, H., Christoph, H.: Design of Magnetic Switchable Device (MSD) and applications in climbing robot. In: H. Fujimoto, M.O. Tokhi, H. Mochiyama, G.S. Virk (eds.) Emerging Trends in Mobile Robotics, pp. 375–382. World Scientific (2010)
46. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, second edn. Prentice Hall (2003)
47. Sabatta, D., Scaramuzza, D., Siegwart, R.: Improved Appearance-Based Matching in Similar and Dynamic Environments using a Vocabulary Tree. In: International Conference on Robotics and Automation (ICRA), pp. 1008–1013. IEEE Press (2010). DOI 10.1109/ROBOT.2010.5509382
48. Sellner, B., Heger, F., Hiatt, L., Simmons, R., Singh, S.: Coordinated multiagent teams and sliding autonomy for large-scale assembly. Proceedings of the IEEE **94**(7), 1425–1444 (2006)
49. Shen, W., Will, P., Khoshnevis, B.: Self-assembly in space via self-reconfigurable robots. In: IEEE International Conference on Robotics and Automation, 2003. Proceedings. ICRA'03, pp. 2516–2521. IEEE Press (2003)
50. Skibniewski, M.J., Wooldridge, S.C.: Robotic materials handling for automated building construction technology. Automation in Construction **1**(3), 251–266 (1992). DOI DOI:10.1016/0926-5805(92)90017-E
51. Stachniss, C.: Robotic mapping and exploration. Springer Verlag (2009)
52. Stewart, R.L., Russell, R.A.: A Distributed Feedback Mechanism to Regulate Wall Construction by a Robotic Swarm. Adaptive Behavior **14**(1), 21–51 (2006). DOI 10.1177/105971230601400104
53. Strachey, C., Wadsworth, C.: Continuations: A mathematical semantics for handling full jumps. Higher-order and symbolic computation **13**(1), 135–152 (2000)
54. Stroupe, A., Okon, A., Robinson, M., Huntsberger, T., Aghazarian, H., Baumgartner, E.: Sustainable cooperative robotic technologies for human and robotic outpost infrastructure construction and maintenance. Autonomous Robots **20**(2), 113–123 (2006)
55. Terada, Y., Murata, S.: Automatic modular assembly system and its distributed control. The International Journal of Robotics Research **27**(3-4), 445–462 (2008)
56. Thrun, S.: Probabilistic robotics. ACM (2002)
57. Tismer, C.: Continuations and stackless Python. In: Proceedings of the 8th International Python Conference. Citeseer (2000)
58. Ueno, H., Nishimaki, T., Oda, M., Inaba, N.: Autonomous cooperative robots for space structure assembly and maintenance. In: Proc. 7th Int. Symp. on Artificial Intelligence, Robotics, and Automation in Space: i-SAIRAS (2003)
59. Wawerla, J., Sukhatme, G., Mataric, M.: Collective construction with multiple robots. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 3, pp. 2696–2701. IEEE Press (2002)
60. Werfel, J., Bar-Yam, Y., Rus, D., Nagpal, R.: Distributed construction by mobile robots with enhanced building blocks. In: Proceedings of 2006 IEEE International Conference on Robotics and Automation, pp. 2787–2794. IEEE Press (2006). DOI 10.1109/ROBOT.2006.1642123
61. Werfel, J., Nagpal, R.: Extended stigmergy in collective construction. IEEE Intelligent Systems **21**(2), 20–28 (2006)

62. Woo, S., Hong, D., Lee, W.C., Chung, J.H., Kim, T.H.: A robotic system for road lane painting. Automation in Construction **17**(2), 122–129 (2008). DOI 10.1016/j.autcon.2006.12.003. 22nd Symposium on Automation and Robotics in Construction, ISARC 2005
63. Zavadskas, E.K.: Automation and robotics in construction: International research and achievements. Automation in Construction **19**(3), 286–290 (2010). DOI 10.1016/j.autcon.2009.12.011
64. Zender, H., Martínez Mozos, O., Jensfelt, P., Kruijff, G., Burgard, W.: Conceptual spatial representations for indoor mobile robots. Robotics and Autonomous Systems **56**(6), 493–502 (2008)
65. Zhuo, H.H., Yang, Q., Hu, D.H., Li, L.: Learning complex action models with quantifiers and logical implications. Artificial Intelligence **174**(18), 1540–1569 (2010). DOI 10.1016/j.artint.2010.09.007