An Overview of Problems and Approaches in Machine Intelligence

Malik Ghallab, Félix Ingrand

An Overview of Problems and Approaches in Machine Intelligence

Malik Ghallab et Félix Ingrand
LAAS-CNRS, University of Toulouse

February 29, 2016

Abstract. *Robotics is an interdisciplinary research field leveraging on control theory, mechanical engineering, electronic engineering and computer science. It aims at designing machines able to perceive, move around and interact with their environment in order to perform useful tasks. Artificial Intelligence (AI) is an area of computer science, overlapping with but significantly distinct from robotics. Its purpose is to understand intelligence, through effective computational models, design and experiment with systems which implement these models .*

There is a significant convergence between Robotics and AI. Their intersection, qualified here as Machine Intelligence, is critical for both areas. Robots implement the so-called "perception - decision - action" loop; the intelligence or decision making part is central in that loop for tackling more variable and complex environments and tasks. On the other hand, AI is moving from abstract intelligence, such as in playing chess, to addressing embodied intelligence.

This paper introduces the reader to some of the research issues and approaches in Machine Intelligence. It surveys the state of the art in key issues such as planning and acting deliberately on the basis of tasks and world models, learning these models, and organizing the sensory-motor and cognitive functions of a robot into resilient and scalable architectures.

Key words: Robotics, Artificial Intelligence.

Contents

1 Introduction

Robotics and *Artificial Intelligence* are two overlapping but quite distinct research fields. *Machine Intelligence* refers to their intersection. This paper surveys the state of the art at this intersection. Its purpose is to introduce the reader to the synergies between Robotics and Artificial Intelligence and to demonstrate that Machine Intelligence is a very rich and fruitful in scientific problems.

Robotics aims at designing machines which are able to perceive, move around and interact with their environment in order to perform some specified useful tasks. It is an interdisciplinary research field, which covers several disciplines, primarily control theory, mechanical engineering, electronic engineering and computer science. Its recent links with life sciences or materials sciences have opened new and exciting perspectives. It entertains growing synergies with neuroscience for the development of cognitive models and functions (e.g., Wolpert and Flanagan [129, 128], Wolpert and Ghahramani [130]). Robotics, as an enabling technology, provides a significant technical and conceptual support for the development of several other research fields such as medicine (e.g., surgery, biomechanics), or environment and space sciences (e.g., oceanography or planetology). It addresses a wide spectrum of applications.

Artificial Intelligence (AI) is a research area of computer science, mostly independent from robotics. Its purpose is to understand intelligence through effective computational models, design systems which implement them, and experiment with these systems in order to scientifically evaluate and qualify the proposed models of intelligence. AI entertains interdisciplinary links with mathematical logics, psychology, neuroscience, linguistics, philosophy and other cognitive sciences. It already brought a wealth of mature technologies, such as machine learning techniques, that are now seamlessly integrated in many computerized devices such as smartphones, cameras, web browsers, search engines and semantic web applications.

Robotics is quite often referred to in AI research. It is a natural reference for work on embodied intelligence and for experimental validation. The early beginnings of AI are rich in pioneering projects of autonomous robots, such as Shakey at SRI of Rosen and Nilsson [101] or the Stanford Cart in the late 60s, and a few years later, Hilare at LAAS of Giralt et al. [56] or the CMU Rover of Moravec [90]. These, and many other projects since that early period, clearly lie at the intersection of Robotics and AI, seeking to understand, model and design machines that combine autonomous perception, decision and action.

AI has been less frequently referred to in robotics publications. This is due to the breadth of the robotics field. This is also due to the early challenges on which the robotics community has focused. Early robots had reduced autonomy and limited sensing, locomotion and manipulation capabilities. This naturally set the initial challenges more about sensory-motor functions than about deliberation

and cognitive functions. Significant progress during the last two decades on the sensory-motor level has, fortunately, put robotics deliberation problems on the limelight.

We are now witnessing a growing convergence between Robotics and AI. Their intersection in Machine Intelligence is critical for both areas. Robots have been defined as a *"perception - decision - action"* control loop. The decision part is central in that loop. On the other hand, AI is moving from abstract intelligence, such as playing chess, to addressing *embodied intelligence*. The intersection of Robotics and AI covers in particular the following issues:

- Perception, semantic interpretation of sensory data, environment modeling;
- Acting deliberately: planning and achieving autonomously complex tasks, including navigation in open unknown environments;
- Learning to perceive, to act and behave with improved performance;
- Organizing sensory-motor and deliberation functions in a robot.

For the sake of a focused survey, the first item is not covered in this paper, to the exception of a brief mention of some aspects of perception that are specific to robotics. The survey is primarily devoted to the last three items, addressed successively in:

- Sections 3, 4 and 5, which are devoted respectively to motion planning and execution, tasks planning and acting, and interaction with humans or robots;
- Section 6 on learning; and
- and Section 7 on organization and architecture issues.

For a good understanding of the problems discussed here, the paper starts with a general introduction to robotics and its applications (Section 2). It concludes with a short perspective on future research. In each section we have chosen to illustrate with enough technical details some basic techniques, and to refer the reader to the relevant publications for further deepening. A wide coverage of robotics can be found in the handbook of Siciliano and Khatib [105]. A similar coverage for AI is given in the textbook of Russell and Norvig [102].

(a) Baxter, a robot manipulator for manufacturing (Rethink Robotics)

(b) Autonomous vehicles for logistics applications (Kiva Systems)

Figure 1: Robots (a) for a fixed environment, and (b) for a single task.

2 Overview of the Field

A robot can be defined as a machine able to perform a set of *tasks* in a class of *environments* with some degree of *autonomy* and robustness. As for any natural being, the autonomous capabilities of a robot are relative to the *diversity* of the tasks and environments it can cope with. A robot integrates several components - actuators, sensors, computers, radio transmitters - which ensure in particular the following functions:

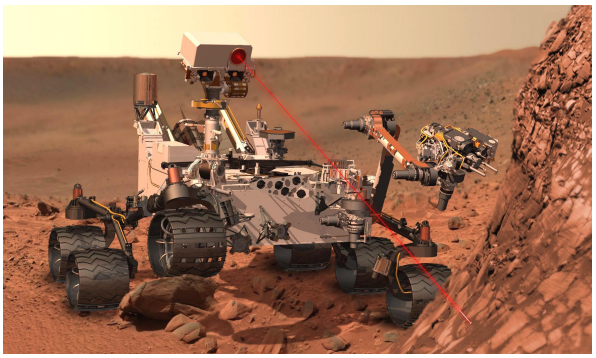
- *motion*, with wheels, legs, wings, propellers, caterpillars, fins;
- *manipulation*, with mechanical arms, clamps, hand, cups, specialized tools;
- *perception* by *proprioceptive* sensors which estimate the internal state of the machine: odometer and angular encoders, inclinometer, magnetometer, accelerometer, inertial measurement unit, GPS, and *exteroceptive* sensors, which estimate the environment: camera, laser, radar, spectrometer, IR or ultrasound range finder;
- *communication*, and
- *decision making*.

There are several classes of generic robotics applications corresponding to different classes of environments and tasks. Each such a class emphasizes specific problems depending on the level of autonomy desired for a robot. Well known examples are the following:

- *Manufacturing robots*: robot arms with adapted sensors at fixed positions for tasks such as painting, welding, assembly, loading and unloading a press or machine tools [59];
- *Exploration robots*: mobile robots in outdoor environments [40] performing terrain mapping, soil analysis, mining [33], intervention in a contaminated site, deployment of equipments at the bottom of the ocean [6], in Antartica or on Mars [131];
- *Service robots*: mobile robots in indoor environments for cleaning, surveillance, transportation in a shop, a workshop, a clean room or an hospital [55];
- *Personal robots*: mobile robots assisting people in professional environments or at home [98];
- *Medical robots*: robots specialized in assisting surgeons, in particular in "noninvasive surgery" [119];
- *Robot carried by human*: exoskeleton allowing the extension of the sensory-motor skills of their carrier [74].

This list is not exhaustive. Other classes of robotics applications, such as agriculture, ecology, construction, demining or military operations give rise to active research. Specific environments in one of the above application classes, e.g., aerial exploration robotics, lead to particular problems. Finally, cooperation and interaction when the tasks are carried out by several robots or by human - robot teams bring additional challenges.

A key notion in robotics is the *diversity of environments and tasks* a robot must face. The technology is relatively mature when there is no diversity, that is for robots specialized in a single environment, well modeled and instrumented, and on just one well specified task. If one considers manufacturing robots, millions robot arms are operating in the industry (Figure 1(a)). In service robotics, numerous autonomous ground vehicles are used in warehouses for logistic services [58] (Figure 1(b) and in the electronic or pharmaceutical industry. In both cases, the well-modeled stable environment



(a) Mars Rover Curiosity (NASA/ JPL)

(b) Surgical robotics assistance (DaVinci Intuitive Surgical)

Figure 2: Tele-operated robots.

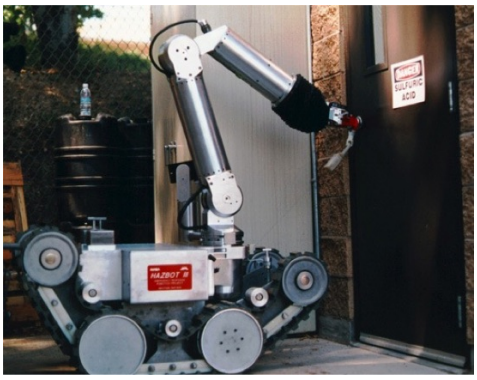
of the robot is the result of a significant engineering effort. The same remark applies to single-task robots, e.g., vacuum cleaner (more than 5 million sold) or lawn mower, which are a large commercial success.

When the environment or tasks are highly variable, the degree of autonomy of the robot becomes an important factor. We may distinguish three levels:

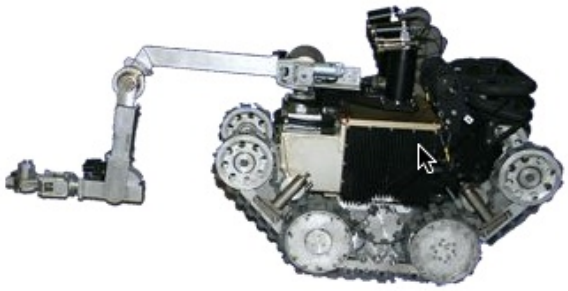
- no autonomy: the robot applies to its actuators pre-recorded or operator specified commands;
- tasks autonomy: the robot performs tasks precisely defined by the operator, e.g., goto point A then pick-up object O;
- autonomy to achieve missions specified in abstract terms, e.g., find and rescue injured persons in the area.

When there is no need for autonomy, many robotics technologies are already mature. This is due in particular to the highly simplified perception and deliberation problems. Robots tele-operated at the task level have been demonstrated in impressive experiments, e.g., in the exploration of planets (Figure 2(a)). They are also used in successful applications, e.g., robotics surgery systems have been deployed at several thousands sites, despite their high cost and complexity (Figure 2(b)). Remote manipulation has to address other technical challenges, such as how to provide good sensory feedback to a human operator to enable her to properly understand the state of the environment and the task, or how to reliably translate human commands to the robot actuators (*e.g.*, to filter the signal from the movements of the surgeon's hand to obtain a precise and safe trajectory of the scalpel and to control its motion with respect to the movement of the operated organ).

Limited autonomy simplifies perception and deliberation but it also constrains the tasks that can be performed by a tele-operated robot. Thus, Mars rovers of the previous generation, *Spirit* and *Opportunity*, were tele-operated at the motor control level. The communication delay (up to 40 minutes depending on the Mars-Earth configuration) limited their remote operation to a few meters per day. At a later stage of their mission, the introduction of autonomous motion has allowed these robots to traverse up to 140 meters per day. Today, Curiosity can perform up to 1.5 Km per day of autonomous navigation, but it is still tele-operated at the task level for its other activities. In some application, autonomy is not desired: the human operator wants to remain in full control of every command. However, it can be preferable to tele-operate a robot at the task level, *e.g.*, tell it to make a precise line of surgical sutures, or to close an underwater valve, leaving it up to the robot to translate the task



(a) NASA/JPL



(b) INTRA Group

Figure 3: Robots for hazardous environments.

into controlled commands, under the supervision of the operator. Here also, the state of the art has reached some maturity, illustrated for example by robots used in hazardous environments (Figure 3). Another illustration of the autonomy at the task level can be given by telepresence robots. These are mobile platforms carrying away the image and voice of the user, giving a visual and audible feedback, capable of simple tasks, e.g., find a person, asking her to lend an object and bringing it back to the robot's user (Figure 4).

One may try to use these and similar platforms to achieve more autonomous and varied missions. But the state of the art faces many open problems, in particular for the interpretation of the environment, for planning and acting with incomplete and uncertain models and noisy sensory data.

(a) Double Robotics

(b) PadBot

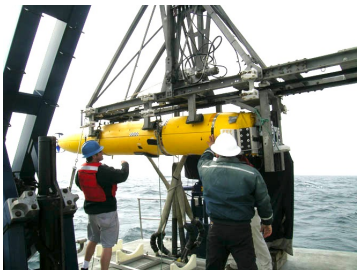
Figure 4: Telepresence robots

Autonomy at the mission level already achieves good experimental success when the tasks are well structured and constrained, even when the environment is highly variable. Driverless cars provide a

good illustration. The first success goes back to the 2005 "DARPA Grand Challenge": autonomous traversal of 320 km in the Mojave Desert in less than 7 hours (Figure 5(a) [123]), which was followed in 2006 by the "DARPA Urban Challenge". Since then, several companies reported millions of kilometers of autonomous driving on roads and highways (Figure 5(b)). Autonomous underwater vehicles (AUV) are another excellent example. Experimental AUVs are launched for up to 24 hours in a mission of mapping, water sampling, oceanographic and biological measurement; in case of a problem, the AUV surfaces and indicates its position to be retrieved by its operators (Figure 5(c) [84]).

(a) Stanley, DARPA challenge 2005 (Stanford U.)

(b) Autonomous Driving (Google)



(c) Dorado, AUV (MBARI)

Figure 5: Autonomous vehicles.

Robotics research relies significantly on experiments. The advance of the field has been conditioned by the availability of inexpensive reliable platforms with broad functionalities that are easily deployable and programmable. Significant progress has been witnessed in the last decade. A good illustration is provided by humanoid robots: many research groups have now access to biped robotic platforms of human size (Figure 6(a) and 6(b)). These robots demonstrate good motor skills as well as impressive mechatronics. Platforms on wheels with two arms, sometimes with an articulated trunk, also illustrate rich sensory-motor capabilities. These platforms are able for example to catch simultaneously two thrown balls (Figure 7(a)), to fold laundry or to play billiards (Figure 7(b)).

Several research competitions stimulated the progress of the field. In addition to autonomous driverless cars, there are several other competitions, e.g., in robotics assembly, aerial robotics or humanoid robotics. The robotics soccer competition "*RoboCup*" is very popular. One can be critical for the oversimplifications often introduced in these competitions (artificial or "micro-worlds" problems). However, their effects in terms of attractiveness, visibility and team commitment, especially among students, remain largely beneficial to the progress of robotics.

3 Motion Planning, Mapping and Navigation

Mobility is a critical and widely studied function for autonomous robots [78, 30, 80]. When the environment is well modeled, the movements of a robot can be planned and controlled in a robust manner. Otherwise, the robot has to explore its environment to acquire the needed geometrical and topological models. Let us discuss here these two problems of motion planning and environment modeling.

(a) HRP 4 (Kawada Industry)

(b) Atlas (Boston Dynamics)

Figure 6: Humanoid robots.

3.1 Motion planning with probabilistic road maps

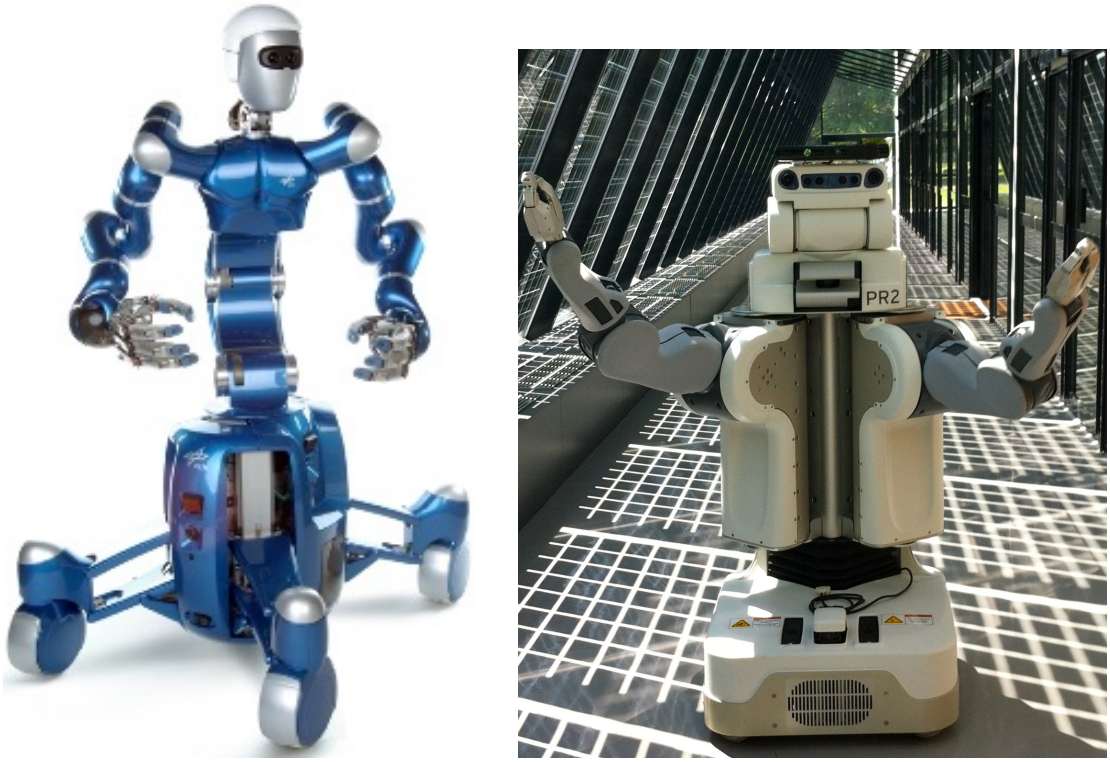
We assume that the environment is described by a geometric model (such as a *Computer-Aided Design* model), which specifies the geometry of the obstacles and the free space. The robot is modeled by its kinematics, *i.e.*, the set of degrees of freedom and the constraints of its moving limbs, as well as its dynamics, *i.e.*, masses and inertia of its components, and the forces and torques of its actuators.

Motion planning consist in finding a trajectory for connecting an initial position to a goal position. This trajectory should be feasible in space and time. The problem is usually decomposed into two steps: *(i)* find a feasible path that satisfies the kinematics constraints of the robot and the geometric constraints of the environment, and *(ii)* find a control law along that path that satisfies the dynamic constraints of the robot. In simple cases these two problems *(i)* and *(ii)* can be solved independently. When there are no moving obstacles and the robot dynamic constraints are weak (e.g., slow motion), it is generally easy to map a feasible path into a feasible trajectory with simple control laws. Motion planning in robotics reduces mainly to a path planning problem, which we detail below.

A free rigid object in Euclidean space without kinematic constraint is characterized by six configuration parameters: (x, y, z) for the position of a reference point and three angles for the orientation of the solid in space. But a robot has kinematic constraints that restrict its movements. For example, a car in the plan has three configuration parameters (x, y and orientation θ), which generally are not independent (a car cannot move laterally). The PR-2 robot (Figure 7(b)) has 20 configuration parameters (3 for the base, one for the trunk, 2 for the head, and 7 per arm). The humanoid robot HRP-4 (Figure 6(a)) has 32 configuration parameters plus five for each hand.

For a robot with n configuration parameters in a given environment let us define:

- $q \in \Re^n$, the configuration of the robot, a vector of n real values that specifies the n parameters characterizing the position of the robot in a reference frame;



(a) Justin (DLR) (b) PR2 at LAAS (Willow Garage)

Figure 7: Mobile robots with two arms.

- $\mathcal{C}$, the configuration space of the robot, which describes all possible values of q in $\Re^n$ given the kinematic constraints, such as the max and min angular positions that each joint can have, and the dependencies between configuration parameters;
- $\mathcal{C}_f \subseteq \mathcal{C}$, the free configuration space which gives all possible values of $q \in \mathcal{C}$ given the constraints of the environment, *i.e.*, the set of configurations for which the robot does not collide with obstacles.

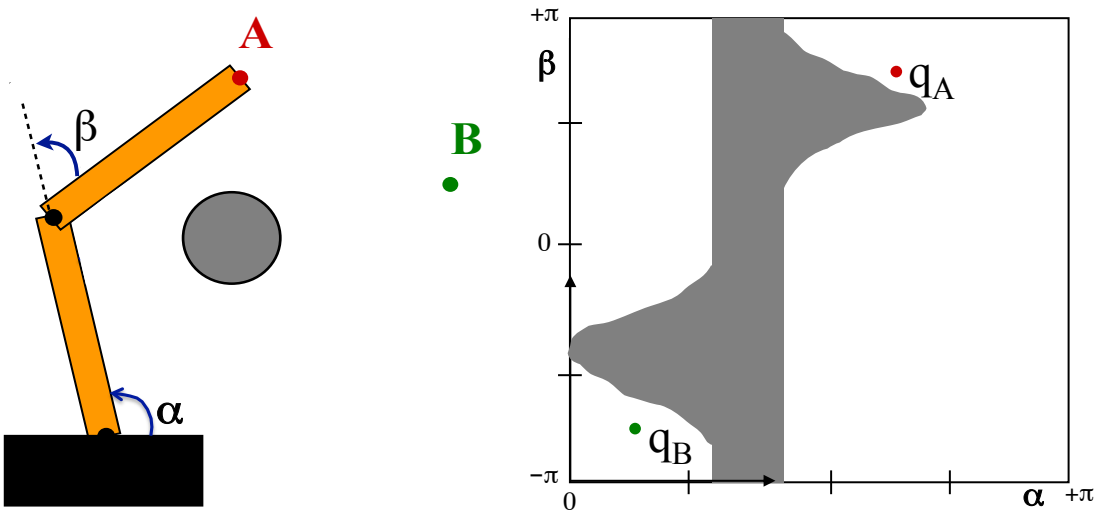
These concepts are illustrated in Figure 8 for a robot with two degrees of freedom.[1]

Planning a motion between an origin configuration q_o and a goal configuration q_g, both in $\mathcal{C}_f$, consists in finding a path between q_o and q_g in this n dimensional continuous space. The major difficulty here, as for any other planning problem, is that the search space $\mathcal{C}_f$ is not known explicitly. The explicit definition of $\mathcal{C}_f$ from the geometric model of the environment and the kinematic constraints of robot is an extremely complex problem, difficult to solve even for very simple robots and environments. In the trivial 2D case of the previous example, this problem corresponds to finding the analytical definition of the grey area in Figure 8(b). Significant research in computational geometry addressed this representation problem, see e.g., Schwartz et al. [104]. It opened the way to sampling-base approaches that helped to circumvent the problem, in particular with the following method.

The Probabilistic Roadmap algorithm of Kavraki et al. [73] relies on two easily computable operations:

- *kinematic guidance*: find a direct kinematic path $\mathcal{L}(q, q')$ between two configurations q and $q' \in \mathcal{C}$ without worrying about environment constraints, *i.e.*, $\mathcal{L}(q, q')$ satisfies the kinematic constraints

[1] Figure adapted from http://www.cs.cmu.edu/motionplanning/



(a) (b)

Figure 8: (a) A planar robot with two angular joints, α and β facing a circular obstacle. (b) Corresponding configuration space: the projection of the obstacle in $\mathcal{C}$ shows that the two configuration q_A and q_B are not connected : no motion of the robot can move it from points A to B.

but not necessarily the constraints of non-collision with obstacles. The techniques used for that are specific to the type of the robot kinematic constraints, *e.g.* composition of straight lines and curves;

- *collision test*: check whether a configuration q does or does not collide with obstacles, *i.e.*, if $q \in \mathcal{C}_f$; check whether a path $\mathcal{L}(q, q')$ between two configurations is collision-free, *i.e.*, if it passes entirely in $\mathcal{C}_f$. This relies on basic techniques of computational geometry.

A roadmap $\mathcal{G}$ in $\mathcal{C}_f$ is a graph whose vertices are configurations in $\mathcal{C}_f$; two vertices q and q' are adjacent in $\mathcal{G}$ iff there exists a path without collision $\mathcal{L}(q, q')$ in $\mathcal{C}_f$.

If a roadmap $\mathcal{G}$ in $\mathcal{C}_f$ is known, then planning a path between an origin configuration q_o and a goal configuration q_g can be solved with the three following steps:

- find a vertex q in $\mathcal{G}$ such that q is accessible from q_o *i.e.*, $\mathcal{L}(q_o, q) \in \mathcal{C}_f$;
- find a vertex q' in $\mathcal{G}$ such that q_g is accessible from q', *i.e.*, $\mathcal{L}(q', q_g) \in \mathcal{C}_f$;
- find a sequence of adjacent vertices in $\mathcal{G}$ between q and q'.

Path planning is then reduced to a simpler problem of finding a path in graph. If such a sequence of configurations is found, efficient algorithms allow to smooth and optimize locally this sequence of configurations in $\mathcal{G}$ into a kinematic path. It remains therefore to find a map $\mathcal{G}$ covering adequately $\mathcal{C}_f$, *i.e.*, if there is a path in $\mathcal{C}_f$ then there is also a path in the roadmap $\mathcal{G}$ using the previous three steps.

The algorithm in Figure 9 [108] provides a graph $\mathcal{G}$ which *probabilistically* satisfies this coverage property. This algorithm incrementally generates $\mathcal{G}$ starting with an empty roadmap. It adds to the map under construction a randomly drawn configuration q in the following two cases:

- if q belongs to the free space and extends the coverage of $\mathcal{G}$, allowing to reach parts of $\mathcal{C}_f$ not yet covered (step $\triangleright$*(i)*), or
- if q belongs to the free space and extends the connectivity of $\mathcal{G}$, allowing to connect two components not currently connected in the roadmap (step $\triangleright$*(ii)*).

Probabilistic Roadmap ($\mathcal{G}$)
  iterate until *Termination*
    sample a random configuration $q \in \mathcal{C}$
    if $q \in \mathcal{C}_f$ then do
      if $\forall q' \in \mathcal{G}$: $\mathcal{L}(q, q') \notin \mathcal{C}_f$ then add q in $\mathcal{G}$ ▷ *(i)*
      else if $\exists q_1, q_2$ in $\mathcal{G}$ such that q_1 and q_2 are not connected
          and $\mathcal{L}(q, q_1) \subset \mathcal{C}_f$ and $\mathcal{L}(q, q_2) \subset \mathcal{C}_f$
          then add q and the edges (q, q_1) and (q, q_2) to $\mathcal{G}$ ▷ *(ii)*
  Return($\mathcal{G}$)

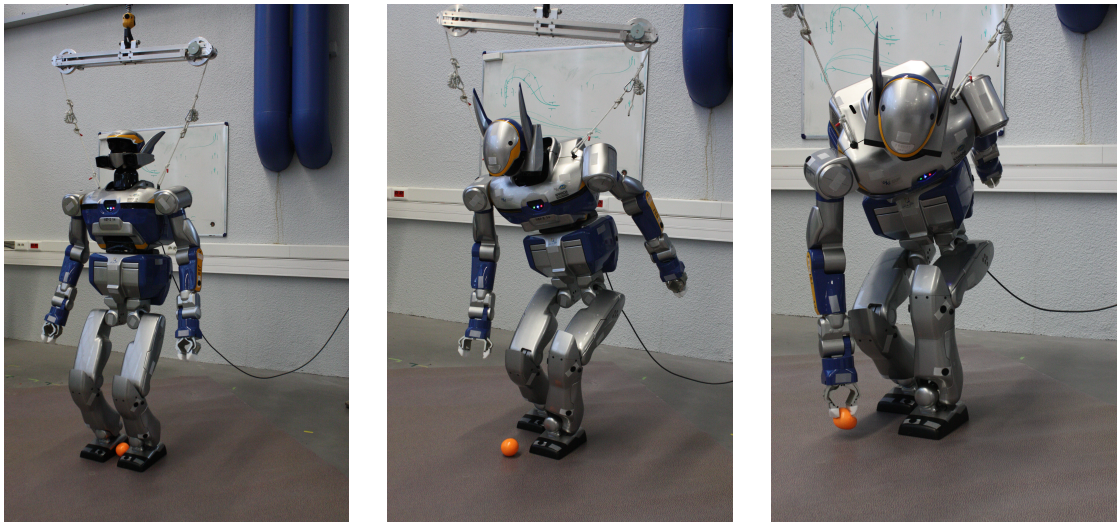
Figure 9: Probabilistic roadmap algorithm for path planning

The *Termination* condition is based on the number of consecutive samples of unsuccessful random free configurations that do not add anything to the map. If k_{max} is such a number, then the probability that the resulting graph covers $\mathcal{C}_f$ is estimated by $(1-1/k_{max})$. In practice, this algorithm is very efficient. The probabilistic roadmap technique and its incremental variants (called RRT for " Rapidly exploring Random Trees" [80]) are now widely used in robotics. They are also used in other application areas such as mechanical design, video animation, or computational biology for molecular docking problems to find whether a ligand can bind to a protein. They have been extended to take into account dynamic environments.

These techniques have advanced significantly the state of the art but they do not solve all motion planning problems in robotics. Many open problems remain, in particular for handling the robot dynamics. Further, one needs to synthesize plans that are robust to the uncertainty of the models and to the sensory-motor noise in the robot localization and motion control. For example, we may want a path that relies on known landmarks to maintain the localization uncertainty below an acceptable threshold. In addition, we need to restate the problem for concrete tasks. The previous formulation refers to a completely specified motion problem, *i.e.*, from a configuration q_o to a configuration q_g. In practice, the problem arises with respect to a task, *e.g.*, grasp an object. This leads to several open problems [107]. A grasp allows to infer the configuration of the end effector (hand and fingers) from the position of the object to be grasped. But the configuration of the end effector gives only a part of q_g. It is possible to decompose the problem into: *(i)* plan the movement of the base of the robot to a configuration "close" to the object, then *(ii)* plan a movement of the arm to a grasp position. However, the manipulation of an object can require intermediate poses at different moment with respect to the object, or the manipulation of other interfering objects. It is then necessary to change the structure of the search space according to the grasps and poses of objects handled. In addition, the above decomposition is not always feasible. For example, a humanoid robot requires a coordinated movement of its body and all limbs [72] (Figure 10). Further, sensing and visibility issues bring additional constraints, e.g., planning a motion that avoids occultation between a camera carried by the robot's head and its hand, to allow for visual servoing [28].

3.2 Simultaneous Localization and Mapping

The execution of a planned motion requires the control of the actuators for achieving a trajectory, possibly with avoidance of unexpected obstacles. The synthesis of this control is done with models and methods from control theory. Robotics raises very interesting problems in automatic control, e.g., in the control of non-holonomic systems. These issues are not within the scope of this paper. We refer the reader for example to the book of LaValle [80] or the synthesis of Minguez et al. [87].



(a) (b) (c)

Figure 10: Picking up a ball requires a coordinated whole body motion planning; here the synthesized plan led the robot to step back, bend and extend opposite arm to maintain its balance (LAAS).

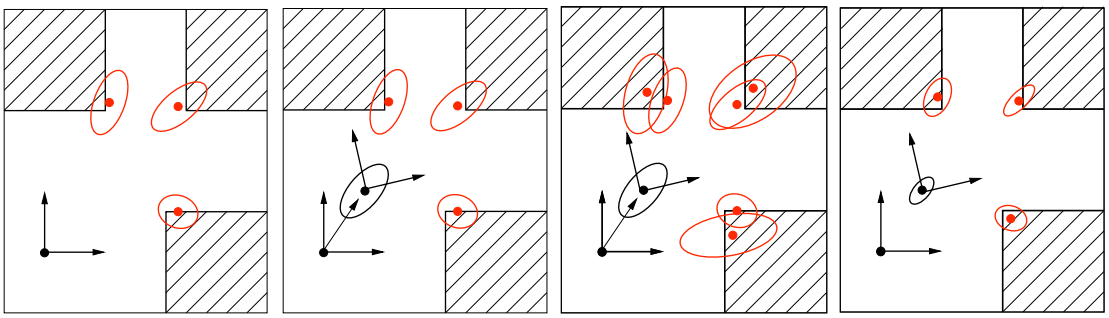
The execution of a planned motion requires also to maintain a good estimate of the state of the robot throughout the execution of the command. In particular, the robot must always know where it is in the environment. Sometimes, one may use absolute localisation, as given by a GPS or a radio-positioning system if the environment provides the adequate infrastructure. However, to operate autonomously in a diversity of environments, a robot must be able to locate itself directly from the perceived natural elements of its environment and a map of this environment. Further, this map is generally partially known, or even unknown. In general a robot is faced with a problem called *simultaneous localization and mapping* (SLAM). This problem has been identified quite early [27, 113], and has been since a very active research topic in robotics.[2]

To define the problem, let us discuss its two subproblems:

- *Localization*: the robot is localized in a fully known environment, modeled by k landmarks that are easily recognizable and perfectly positioned in space (2D or 3D). At time t, the robot is in a position estimated by $\tilde{x}_t$. It moves with the command u_t (giving the movement speed and orientation between t and t'). This allows to estimate the new position $\tilde{x}'$. The robot observes k landmarks where it expects to find them (from the estimated $\tilde{x}'$). It updates its position in relation to each recognized landmark. The observed positions of the landmarks are combined into a new estimated position of the robot $\tilde{x}_{t+1}$. The process is repeated at each time step as long as the robot remains within a fully known environment. The intermediate estimate $\tilde{x}'$ serves only to find landmarks. The localization error takes into account the sensing errors in the landmark observed positions, but it does not increase with time as long as the landmark locations in the map are error free. The error associated with the motor command u_t does not affect the localization.
- *Mapping*: The robot builds a map of its environment assuming it knows precisely its successive positions. The j^{th} landmark is estimated at time t as $\tilde{x}_{j_t}$. The robot moves between t and $t+1$ to a new known position, from which it observes again the position of the j^{th} landmark as $\tilde{x}'_j$

[2] See, e.g., the software repository: http://www.openslam.org/

with sensing error. $\tilde{x}'_j$ and $\tilde{x}_{j_t}$ are combined into a more reliable estimate $\tilde{x}_{j_{t+1}}$. The map quality improves with time.



(a) (b) (c) (d)

Figure 11: SLAM procedure for a simple 2D robot: (a) Three landmarks (corners of obstacles) are detected and positioned with inaccuracy due to sensing noise. (b) The robot moves and estimates its position with a motion error. (c) The landmarks are observed and associated with the corresponding ones previously perceived. (d) Data fusion reduces the errors on the current position of the robot and the positions of the landmarks. The process is iterated for each new robot motion and sensing.

In practice, the two problems have to be addressed simultaneously. The initial map, if there is one, is never error free. Errors in the map entail localization errors. Symmetrically, the robot localization is noisy, which entails errors in its updates of the map. However, the two sources of error, from sensing and motion, are not correlated (see Figure 11). It is possible to combine the two subproblems into the simultaneous estimate of the positions of the robot and the landmarks.

One approach initially explored for solving the SLAM relies on *extended Kalman filters*. The technical details may seem complicated but a step by step presentation shows that the principle is simple. It is assumed that the environment is static and the sensors of the robot are properly calibrated and do not introduce systematic bias. Sensing errors are modeled as a Gaussian noise with zero mean and a standard deviation specific to each sensor. Let us assume two sensors, characterized respectively by σ_1 and σ_2, which both measure the distance to the same landmark. They return two values μ_1 and μ_2. We can estimate the true distance by averaging the returned values while giving more confidence to the most accurate sensor, i.e., the one with the smaller σ_i. Hence μ_i is weighted by $1/\sigma_i$. The estimated distance μ is associated with a standard deviation σ defined below (Equation 1). This estimates has good properties: it minimizes the mean squared error. The error resulting from the combination of the two measures decreases, since $\sigma <\min\{\sigma_1, \sigma_2\}$.

$$\begin{aligned} \mu &= \alpha(\mu_1/\sigma_1 + \mu_2/\sigma_2), \text{ with } \alpha = \sigma_1\sigma_2/(\sigma_1 + \sigma_2) \\ 1/\sigma &= 1/\sigma_1 + 1/\sigma_2 \end{aligned} \tag{1}$$

This process is applied incrementally. We combine the current estimate (μ', σ') to the new measure (μ_z, σ_z). The new estimate at time t (μ_t, σ_t) integrating the new measure is given by the same equation, rearranged easily into the following form (Equation 2):

$$\begin{aligned} \mu_t &= \mu' + K(\mu_z - \mu') \\ \sigma_t &= \sigma' - K\sigma' \\ K &= \sigma'/(\sigma_z + \sigma') \end{aligned} \tag{2}$$

Let us now introduce the robot's motion. At time $t-1$ the robot was in a position with respect to the landmark of interest estimated by $(\mu_{t-1}, \sigma_{t-1})$. Between $t-1$ and t the robot moves according to a command known with an uncertainty similarly modeled. Let (u_t, σ_u) be the estimate of this motion along the robot - landmark line. This estimate is given by the command sent to actuators and/or by the odometer. The relative distance to the landmark after the motion is estimated by (μ', σ'), noting that the error increases due to the motion:

$$\begin{aligned} \mu' &= \mu_{t-1} + u_t \\ \sigma' &= \sigma_{t-1} + \sigma_u \end{aligned} \tag{3}$$

We now can combine the two previous steps into a SLAM approach based on Kalman filtering. The estimate of the relative position robot - landmark is updated between $t-1$ and t in two steps:

(i) update due to motion (with Equation 3): $(\mu_{t-1}, \sigma_{t-1}) \rightarrow (\mu', \sigma')$
(ii) update due to sensing (with Equation 2): $(\mu', \sigma') \rightarrow (\mu_t, \sigma_t)$

In the general case, these updates are applied to vectors instead of simple scalar values. We run the above process to the update of the positions of the robot and the landmarks in the Euclidean space, 2D or 3D. The position of the robot does not necessarily include all its configuration parameters, but only the portion of q necessary for the localization of a reference point and for the positioning of its sensors. The map is characterized by many landmarks positioned in space. A vector μ_t, whose components are the robot configuration parameters and the positions of the landmarks, is updated at each step. The error is no longer a scalar σ_t but a covariance matrix Σ whose element σ_{ij} is the covariance components i and j of the parameters of μ. The error on the position of the robot is coupled to the errors of the map and symmetrically. Furthermore, the above approach applies only to linear relations. But the relationship between the command and the motion is not linear. We approximate a solution to this problem by linearizing around small motions. This leads finally to the extended Kalman filter formulation of SLAM:

$$\begin{aligned} \mu' &= A\mu_{t-1} + Bu_t \\ \mu_t &= \mu' + K_t(\mu_z - C\mu') \\ \Sigma' &= \sigma_{t-1} + \Sigma_u \\ \Sigma_t &= \Sigma' - K_t C\Sigma' \\ K_t &= \Sigma' C^T (C\Sigma' C^T + \Sigma_z)^{-1} \end{aligned} \tag{4}$$

Two update steps are easily identified:

(i) $(\mu_{t-1}, \sigma_{t-1}) \rightarrow (\mu', \Sigma')$: vector u_t, matrices A and B for the motion,
(ii) $(\mu', \Sigma') \rightarrow (\mu_t, \Sigma_t)$: vector μ_z, matrix C for the new measurements.

One also takes into account the covariance associated with the motion and the measurements (Σ_u and Σ_z). It should be noted that the first step uses the motion to update the position of the robot as well as those of the landmarks. Similarly, the second step integrates the new measurements for both, the localization and mapping.

This approach has been successfully implemented and frequently used [122]. It has many advantages. In particular, it maintains the robot localization and the corresponding bounds on the error. These bounds are very important in navigation: if the error grows beyond some threshold, specific action has to be taken. The method converges asymptotically to the true map, with a residual error

due to initial inaccuracies. Finally, the estimate is computed incrementally. In practice, the number of landmarks increases dynamically. The robot maintains a list of landmark candidates which are not integrated into the map (nor in the vector μ) until a sufficient number of observations of these landmarks have been made. If n is the dimension of the vector μ (*i.e.*, the number of landmarks), the complexity of the update by Equation 4 is $O(n^2)$. The computations can be done online and on board of the robot for n in the order of 10^3, which means a sparse map.

Particle filtering offers another approach to SLAM with additional advantages. Instead of estimating the Gaussian parameters (μ, Σ), the corresponding probability distributions are estimated through random sampling. Let $P(X_t|z_{1:t}, u_{1:t}) = \mathcal{N}(\mu_t \Sigma_t)$, where X_t is the state vector of the robot and landmark positions at the time t, $z_{1:t}$ and $u_{1:t}$ are the sequences of measures and commands from 1 to t. Similarly $P(z_t|X_{t-1}) = \mathcal{N}(\mu_z \Sigma_z)$.

Let us decompose the state vector X_t into two components related to the robot and the landmarks: $X_t = (r_t, \phi_1, ..., \phi_n)^T$, where r_t is the position of the robot at time t, and $\phi = (\phi_1, ..., \phi_n)^T$ the position of landmarks, which do not depend on time because the environment is assumed static.[3] The usual rules of joint probabilities entail the following:

$$\begin{aligned} P(X_t|z_{1:t}, u_{1:t}) &= P(r_t|z_{1:t}, u_{1:t}) P(\phi_1, \ldots, \phi_n|z_{1:t}, u_{1:t}, r_t) \\ &= P(r_t|z_{1:t}, u_{1:t}) \prod_{i=1,n} P(\phi_i|z_{1:t}, r_t) \end{aligned} \tag{5}$$

The second line results from the fact that, given the position r_t of the robot, the positions of the landmarks do not depend on u and are conditionally independent. The robot does not known precisely r_t but it assumes that $r_t \in R_t = \{r_t^{(1)}, \ldots, r_t^{(m)}\}$, a set of m position hypotheses (or particles). Each hypothesis $r_t^{(j)}$ is associated with a weight $w_t^{(j)}$. R_t and the corresponding weights are computed in each transition from $t-1$ to t by the following three steps:

- *Propagation*: for m' positions in R_{t-1} randomly sampled according to the weights $w_{t-1}^{(j)}$, we compute $r_t^{(j)}$ the position at time t of the resulting control u_t, with $m' > m$,
- *Weighting* : the weight $w_t^{(j)}$ of particle $r_t^{(j)}$ is computed taking into account the observation z_t from the product $P(z_t|\phi, r_t^{(j)}) P(\phi|z_{1:t-1}, r_{t-1}^{(j)})$.
- *Sampling*: the m most likely assumptions according to the new weights $w_t^{(j)}$ are kept in R_t.

For each of the m particles, the probability $P(\phi_i|z_{1:t}, r_t)$ is computed with a Kalman filter reduced to the 2 or 3 parameters necessary to the position ϕ_i. With good data structures for the map, this approach, called FastSLAM [88], reduces the complexity of each update to $O(n \log m)$ instead of $O(n^2)$ in the previous approach. In practice, one can keep a good accuracy for about $m \simeq 10^2$ particles, allowing to maintain online a map with $n \simeq 10^5$ landmarks.

The main limitation of these approaches is due to a well known and difficult problem of *data association*. At each step of the incremental localization process, one must be sure not to confuse the landmarks: associated measurements should be related to the same landmark. An update of the map and the robot positions with measurements related to distinct landmarks can lead to important errors, well beyond the sensory-motor errors. This argument, together with the computational complexity issue, favors sparse maps with few discriminating and easily recognizable landmarks. On a small motion between $t-1$ and t, the landmarks in the sensory field of the robot are likely to be

[3] Note that in μ_t the estimate ϕ evolves with t, but not the position of the landmarks.

recognized without association errors. But after a long journey, if the robot views some previously seen landmarks, a robust implementation of the approach requires a good algorithm for solving the data association problem.[4] In the particle filtering approach, the probability distribution of R_t is very different when the robot discovers a new place (equally likely distribution) from the case where it retraces its steps. This fact is used by active mapping approaches, which make the robot retrace back its steps as frequently as needed [114].

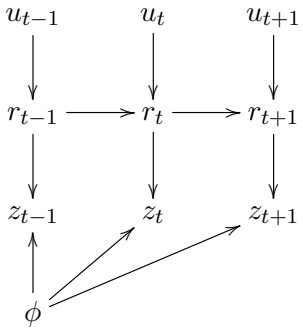


Figure 12: Formulation of SLAM with a dynamic Bayesian network; arcs stand for conditional dependencies between random variables, ϕ gives the positions of the landmarks (time-independent), u_t, r_t and z_t denote the command, the robot positions and the new measurements at time t.

In the general case, there is a need for an explicit data association step between the two stages *(i)* and *(ii)* corresponding to Equation 4. This step leads to maintain multiple association hypotheses. The SLAM approaches with Dynamic Bayesian Networks (DBN) for handling multi-hypotheses give good results. The DBN formulation of SLAM is quite natural. It results in a dependency graph (Figure 12) and the following recursive equation:

$$\begin{aligned} P(X_t|z_{1:t}, u_{1:t}) &= \alpha P(z_t|X_t) \int P(X_t|u_t, X_{t-1}) P(X_{t-1}|z_{1:t-1}, u_{1:t-1}) dX_{t-1} \\ &= \alpha P(z_t|X_t) \int P(r_t|u_t, r_{t-1}) P(X_{t-1}|z_{1:t-1}, u_{1:t-1}) dr_{t-1} \end{aligned} \tag{6}$$

Here, α is a simple normalization factor. The vector state is as above $X_t = (r_t, \phi_1, ..., \phi_n)^T$; the second line results from the fact that the environment is assumed static and that the robot motion and landmark positions are independent. The term $P(z_t|X_t)$ expresses the sensory model of the robot, and the term $P(r_t|u_t, r_{t-1})$ corresponds to its motion model. This formulation is solved by classical DBN techniques, using in particular the Expectation-Maximization algorithm (EM), as for example in Ghahramani [51], which provides a correct solution to the data association problem. However, online incremental implementation of EM are quite complex. Let us also mention another version of FastSLAM which takes this problem into account by an explicit optimization step over all possible associations [89].

Recent approaches to SLAM favor this DBN formulation with a global parameter estimation problem overs the set of landmarks and robot positions. The problem is solved by robust optimization methods. This general formulation is called the beam adjustment method, following the computational

[4] This is sometimes referred to as the *SLAM loop problem*.

vision and photogrammetry techniques [125]. Visual SLAM has also benefited from recent image processing features which are quite robust for localization and identification of landmarks [85, 93, 82].

Let us conclude this section by mentioning a few possible representations for the map of the environment. Landmarks can be any set of sensory attributes that are recognizable and localizable in space. They can be a simple collection of points. They can also be compound attributes, such as visual segments, planes, surfaces, or more complex objects. The most appropriate attributes are generally specific to the type of sensors used. The global map can be represented as a 2D occupancy grid. Simple 3D maps for indoor environments, such as the Indoor Manhattan Representation, combine vertical planes of walls between two horizontal planes for the floor and ceiling, [46]. They can be used with more elaborate representations integrating semantic and topological information (see next section).

3.3 Navigation

The previous approaches are limited to metric maps. They only take into account distances and positions in a global absolute reference. When the environment is large, it is important to explicitly represent its topology, possibly associated with semantic information. In this case, a map relies on hierarchical hybrid representations, with metric sub-maps in local reference frames, together with relationships and connectivity constraints between sub-maps. The robot re-locates itself precisely when arriving in a sub-map.

Navigation in this case is also hybrid. Within a sub-map, motion planning techniques are used. Between sub-maps other methods such as road following or beam heading are more relevant. Sensory aspects and place recognition play an important role in navigation methods for *semantic hierarchy of spatial representations* [76].

Mapping and map updates can be as flexible as in the case of SLAM through the updates of a graph of local sub-maps [77, 38]. Topological planning relies on path search techniques in graphs (using algorithms such as Dijsktra or A^*). It is associated with motion planning in sub-maps. Both types of planning can be combined incrementally. Topological planning gives a route which is updated and smoothed incrementally to optimize the motion giving the observed terrain while moving [75].

Topological planning in a graph or within a grid can be used with a partial knowledge of the environment. Extensions of the A^* algorithm (D^* [115], D^* Lite, or Focused D^*) compute shortest paths in the graph, but they use the robot sensing to update the topology and costs parameters for finding shortest paths.

Finally, a classical problem in any hybrid approach is that of the frontiers between levels and their granularity. Labels of places (doors, rooms, corridors) and topology can emerge naturally from sensing and/or from a uniform description of space into cells (grids, polygons or Delaunay triangles). Decomposition techniques by *quadtrees* (a partially occupied cell is decomposed recursively) are useful but can be computationally complex. Analysis of the levels of connectivity of a graph provides elegant solutions with low complexity when the topological graph is planar [64, 79].

4 Task Planning and Acting

Task planning is the problem of synthesizing a plan, i.e., a sequence or a structured set of actions, starting from the description of all possible actions that a robot can perform, and such that the synthesized plan achieves an intended objective. Task planning is supposed to be general enough to handle all kind of tasks, integrating mobility, manipulation, assembly, sensing, etc. A planner is a *predictive*

system: it chooses, among various projections of possible futures those likely to lead to the goal. For this, the models of possible actions are at some level of abstraction that allows easy predictions. They are mainly logical or relational models, which grasp the causal relationships between actions, their conditions, effects and the intended objectives. The plans produced are more like guidelines for acting than direct programs to execute in open loop: they seldom fully unfold as expected, along a nominal scenario. Once a plan is found, there are problems for acting according to that plan, i.e., transforming the abstract actions in the plan into commands adapted to the context, monitoring their execution, and if necessary, to taking corrective steps, including replanning.

Robotics was one of the first area that motivated the development of task planning. It led naturally to the issue of coupling of planning and acting – the STRIPS planner of Fikes and Nilsson [43], on the Shakey robot, associated with Planex [42] for the execution of plans, is a seminal work in this area.

The execution controller (controller for short) does not make prediction. It uses different types of models which allow monitoring and, possibly, diagnosis. It must know which actions, especially the sensory ones, are needed to launch a planned action and/or to observe the direct or indirect effects of the action. It must be able to update the state of the world required to monitor the plan execution. It must know the conditions which invalidate the current action, expressing the failure or absence of response time, and those which invalidate the current plan. In addition, the controller must be able to manage uncertainty and nondeterminism at various levels: the imprecision of sensory data and the uncertainty about their interpretations; the action duration; the nondeterminism inherent to the action outcomes, etc. Indeed, the controller launches the actions, but their effects and precise courses of execution depend upon conditions and contingent events partially modeled. Finally, by definition, the controller operates online: it must also be responsive to unforeseen events by the plan, and ensure some safety conditions.

The coupling of planning and acting requires a tradeoff between the constraints and models needed for the planner predictions and those needed for the acting online with action refinements, reactions, monitoring and revision. A description of a planning and acting system and how to achieve this tradeoff could be made on the basis of a hierarchical state transition system $\Sigma = (S, A, E, \gamma)$, where S, A and E are enumerable sets of *state* of *activities* and *events*, and γ is a function that describes the dynamics of the system $\gamma = S \times A \times E \rightarrow S^2$. Activities are decided and triggered by the robot, while events are not under its control; they give rise to changes in the environment which can be observed directly or indirectly. Σ is described with two levels of abstraction:

- the planner has an abstract model of Σ: its macro-states are subsets of S, its actions are subsets of activities; it rarely takes into account E;
- the controller has a finer model of Σ: it is able to refine each planned action in corresponding activities which are under its control; it knows how to launch activities and how to monitor their progress; it can trigger activities (e.g., monitoring, alarms) to observe the dynamics of S, and other activities to react to events.

A complete formalization of such a system depends on many conditions, especially the type of planning used, deterministic or non-deterministic and the system dynamics, *e.g.*, how to take into account the concurrency between activities and events within the function γ. A presentation of possible approaches is beyond the scope of this paper. We refer the reader to the textbook of Ghallab et al. [54] for a detailed coverage of tasks planning methods, and to the recent survey of Ingrand and Ghallab [66] for a broad perspective on deliberate actions in robotics . In the remainder of this section, let us review some of the main approaches for acting and execution control, focusing on relational and logic representations in deterministic and temporal approaches, and on Markov representations for nondeterministic approaches.

4.1 Deterministic Approaches

The approaches using a classical planner (as in STRIPS) often produce a plan π to which they associate a causal structure that help the controller follow the proper execution of the plan (*e.g.*, triangular tables). The purpose of these structures is to provide the conditions of use of the actions so that the controller can verify their applicability and their proper execution. If these conditions are not met the control can relaunch this action (or another) or it can call the planner to produce a new plan.

These causal structure to monitor the execution of a plan are quite limited. Richer formalisms have been proposed to permit the execution of plans. They can be classified into two broad families.

Imperative Languages such as RAP [45], PRS [65], or TDL [109]. They offer an imperative programming language that allows to specify procedures to be performed to meet some objectives (e.g. perform an action). These languages offer conventional programming control structures (test loop, recursion, parallelism, etc.), and often rely on concepts borrowed from logic programming (as in Prolog).

State Transition Systems such as SMACH, the ROS controller language of ROS [18]. The user provides a set of hierarchical finite state machines. Each state corresponds to an activity involving one or more components of the robot. According to the returned values of executions, the controller performs the appropriate transition to the next state. The overall state of the system corresponds to the composition of the hierarchical automata.

These systems, based on automatons or procedures are very useful and necessary in setting complex robot experiments where one must coordinate many software components. However, these models, used to refine actions in activities, must be directly programmed by procedures or automatons developers, and are not inferred from specifications. This is a problem with respect to their validation and verification.

Planning with Hierarchical Task Network (HTN) [118, 37] naturally incorporates a refinement process of abstract tasks in elementary actions. HTNs represent decomposition methods of task as a network (often an and/or tree) of elementary actions. The specification of knowledge in these approaches appears natural to the programmer. These approaches seldom provide ways to refine planned actions into commands, and to repair refinements when an execution failure occurs. However, several HTN systems are used in robotics and extend the formalism in various ways. For example SIPE [126] can produce plans where the duration of actions is taken into account. TCA/TDL [109] integrates execution and decomposition during the execution of tasks in plans. Xfrm [13] can produce plans following an HTN approach, but also allows the modification/repair of these plans while executing them (transformational planing).

4.2 Timed Approaches

The controller of an autonomous robot must explicitly take into account time. A state transition approach is not sufficient. Indeed, the activities of the robot are not instantaneous (motions, taking images, etc). Often, they must be executed in parallel, synchronized, and bounded with earliest and latest date. These motivations lead to explicitly include time and temporal constraints in the models: the plan produced will be more robust with respect to execution.

Several planning approaches based on temporal intervals or events formalisms [5, 52] have been developed, e.g., IxTeT [53] HSTS [92], Europa [48], APSI [49]. They led to extensions that take into account execution. They produce plans in the form of a lattice of instants (the beginnings and ends of actions) or intervals. A timeline represents the temporal evolution of a state variable (e.g., the position

of the robot); it is composed of instants or intervals in which the variable keeps a value (e.g., the robot does not move), or changes its value (the robot moves). The search for a solution plan is in the space of partial plans (where each state is a partial plan with a set of partially instantiated and ordered actions), with a least commitment strategy.

These approaches have many advantages for planning and execution in robotics. They properly manage concurrency or parallel execution. Furthermore, they generally produce plans that are temporally flexible, leaving to the execution the choices of the exact dates of occurrence (controllable or non-controllable but observable). For this, the execution controller must continually propagate the time constraints based on the date of occurrence actually observed to ensure that the plan remains consistent and repairable in case of inconsistency.

Some approaches (e.g., IDEA and T-ReX) offer a paradigm where the planner and the controller are tightly coupled in a set of reactors, each with its own horizon for planning and execution.

For events as well as intervals, these approaches rely on Simple Temporal Networks (STN) to model the temporal constraints between the events considered. An STN is a constraint network whose variables are events; constraints between two events t_i and t_j are of the form: $min_{ij} \leq t_j - t_i \leq max_{ij}$. The Allen Algebra of intervals [5] (using relations such as *before, meets, overlaps, starts, during, finishes*, their symmetrical and equality) can easily be transformed into an equivalent STN. One has just to translate the relations in precedence (or equalities) on the beginnings and ends of each interval.

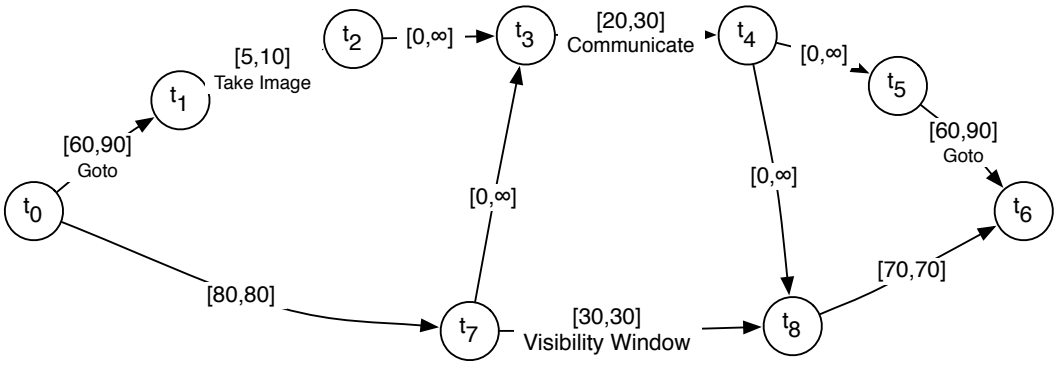
The plan produced is an STN described by the corresponding constraint. Figure 13(a) shows the STN plan of a Mars rover that must go to a given location, take a picture, communicate the result to an orbiter during a window visibility, then return to its base. The network can be transformed into a distance graph (see Figure 13(b) where arcs correspond to the inequalities $t_j - t_i \leq max_{ij}$ and $t_i - t_j \leq -min_{ij}$). One finds the minimum using Floyd-Warshall algorithm 14(a). Here $dist[i, j]$ is the minimum distance from i to j, initialized with an infinite value when i and j are not constrained. One then obtains the graph in Figure 13(c).

When an STN is taken as a task to perform, the execution controller must incrementally propagate the update using algorithm 14(b) (which is of a lesser complexity, $O(n + n^2)$ instead of $O(n^3)$). In the example above, if the first Goto takes exactly 70 seconds, we get the STN in Figure 13(d) and after propagation the graph in Figure 13(e).

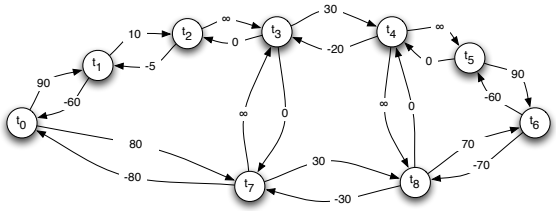
These approaches have been successfully implemented in many robotic experiments (e.g., MBARI [100], Willow Garage [83], NASA [44] and LAAS [81]) but their development faces the following difficulties:

- writing the planning models and debugging them is difficult, especially when one wants to take into account nonnominal execution situations (i.e., failures and error recovery),
- the search for solutions in the partial plans space must be guided by adapted heuristics,
- the temporal controllability of the STN must be taken into account. Indeed, these STNs contain *controllable* variables but also *contingent* variable. The values of the formers are selected by the robot, whereas the values of the latters are contingent and determined by the environment within their domain [5]. An STN is controllable if there a possible value assignment for controllable events depending on the values of the contingent ones. Strong controllability ensures that there exists an assignment of values of controllable events for all possible values of contingent ones. Weak

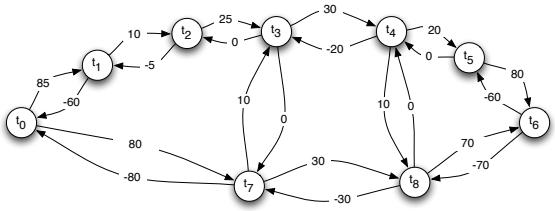
[5]For example, in the graph Figure 13(c) to move between t_0 and t_1, the starting time t_0 is controllable, but not the arrival time t_1. Travel time was reduced by propagation from 90 to 85 (Figure 13(c)), but in fact, only the observation after execution will give the exact value.



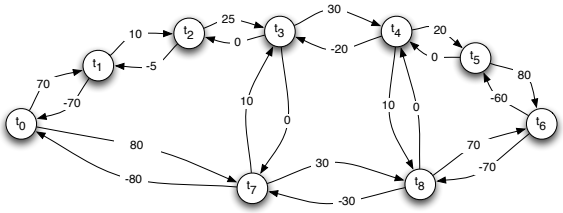
(a) Simple temporal network (STN)



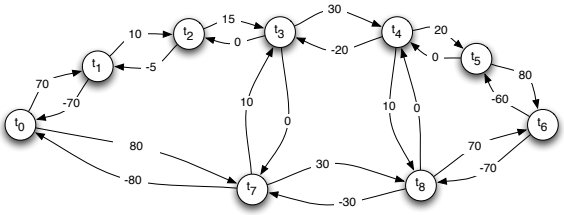
(b) Distance graph



(c) After a Floyd-Warshall propagation (Algo 14(a))



(d) After execution of the first Goto



(e) After incremental propagation (Algo 14(b))

Figure 13: Successive phases of planning and execution of a temporal plan for a Mars exploration rover.

Floyd-Warshall$(dist, n)$
  for k from 1 to n
    for i from 1 to n
      for j from 1 to n
        $dist[i,j] \leftarrow min\{dist[i,j],$
          $dist[i,k] + dist[k,j]\}$

(a) Initiale propagation: Floyd-Warshall Algorithm

Incremental$(dist, n, i_0, j_0)$
  for i from 1 to n
    $dist[i,j_0] \leftarrow min\{dist[i,j_0],$
      $dist[i,i_0] + dist[i_0,j_0]\}$
  for i from 1 to n
    for j from 1 to n
      $dist[i,j] \leftarrow min\{dist[i,j_0],$
        $dist[j_0,j] + dist[i,j]\}$

(b) Incremental propagation: after changing the constrain between two events i_0 et j_0

Figure 14: Temporal contraints propagation algorithms.

controllability ensures that there is a possible value assignment for the controllable ones for all the values of the contingent ones, if they are known in advance (unrealistic). Dynamic controllability

ensures that there is an assignment for controllable ones for the values of the past contingent ones. This last property keeps the flexibility while making sure that a solution remains.

Other approaches (e.g., Aspen/Casper [29]) based on a temporal model produce complete plans without any flexibility. If a temporal (or a causal) failure occurs when executing the plan, the planner then repairs it using local search techniques.

4.3 Probabilistic Approaches

Nondeterminism is not an intrinsic property of a system but a property of its model. Interaction with the real world always involves some level of nondeterminism, that may of may not be grasped in its model. The same arguments that foster the need for autonomous deliberation in a robot, i.e., open and diverse environments and tasks, promote the use of nondeterministic models. These allow to handle various possible interactions between the robot actions and the environment own dynamics, possibly with probabilistic models. Markov Decision Processes (MDP) provide a convenient representation for planning under uncertainty. Let us introduce here the general MDP approach, which will be also useful for section 6 about learning.

Let S be a finite set of states, and A a finite set actions. If an action a is applicable in a state s, a can lead *nondeterministically* to any states in $F(s,a) \subseteq S$. Let $P(s'|s,a)$ be the probability of reaching state s' when action a is applied in s; $r(s,a) \geq 0$ is the reward associated with a in s. Let $\pi : S \rightarrow A$ be an application that associates to each state s the action to be performed in s. π is called a *policy*; it corresponds to a plan that tells the robot which action to carry in each state. π has possibly loops, i.e., following π from a state s may lead back to s after one or a few steps. The value function $V_\pi(s)$ of a state s under policy for π is the expected sum of rewards of this plan, weighted (to ensure convergence) by a decreasing coefficient:

$$\begin{aligned} V_\pi(s) &= E[\sum_{t=0}^{\infty} \xi^t r(s_t, \pi(s_t))], \text{with } \xi < 1 \\ &= r(s,\pi(s)) + \xi \sum_{s' \in F(s,\pi(s))} P(s'|s,\pi(s)) V_\pi(s') \end{aligned} \tag{7}$$

The optimal value function for a state s is $V^*(s)$ for the optimal policy π^*.

$$\begin{aligned} V^*(s) &= \max_\pi V_\pi(s) \\ &= \max_a \{Q^*(s,a)\}, \text{with} \\ Q^*(s,a) &= r(s,a) + \xi \sum_{s' \in F(s,a)} P(s'|s,a) V^*(s') \end{aligned} \tag{8}$$

Dynamic programming leads to a recursive formulation of V^* and provides easily implementable algorithms, such as Value Iteration (see Figure 15).

Value Iteration algorithm [16] terminates when a fixed point is reached, i.e., a full iteration over S without a change in any $V(s)$. It gives the optimal policy π^*. It can be initialized with an arbitrarily $V(s)$. In practice one does not need to loop until a fixed point. It is sufficient to make sure that all updates of $V(s)$ on some iteration over S remain below a threshold ϵ. The returned solution then deviates from the optimum by at most $2\epsilon \times \xi/(1-\xi)$.

The above formulation is not goal oriented: it seek an optimal policy for an infinite process. This formulation can be transformed into a goal-oriented approach by giving an initial state s_0, a set of goal states $S_g \subset S$, and by searching for an optimal policy that leads from s_0 to one of the states in

Value Iteration(S, A, P, r)
  until reaching a fixed point do
    for each $s \in S$ do
      for each a applicable in s do
        $Q(s,a) \leftarrow r(s,a) + \xi \sum_{s'} P(s'|s,a)V(s')$ ▷ *(i)*
      $V(s) \leftarrow \max_a\{Q(s,a)\}$
  for each $s \in S$ do
    $\pi(s) \leftarrow \text{argmax}_a\{Q(s,a)\}$

Figure 15: Value Iteration algorithm.

S_g. One can also integrate cost distribution on actions and variable rewards function for goal states. In such a formulation, one is not searching for policy defined everywhere, but for a partial policy, defined only in states reachable from s_0 by this policy. A *safe* policy π is guarantied to reach a goal from s_0. If a problem has a safe policy, then dynamic programming with $\xi = 1$ can find an optimal one. The algorithm Value Iteration applies to the case where there is a safe policy from every state. When this assumption does not hold, the problem is said to have *dead-ends*, i.e., states from which a goal is not reachable. Extensions to dynamic programming algorithms have been introduced, e.g., [20, 16, 9]. For example it is not necessary, nor possible iterate over all S. It is enough to search along states reachable from s_0 with a current policy. One may also estimate $Q(s,a)$ by sampling techniques [68]. The step ▷*(i)* is replaced by $Q(s,a) \leftarrow Q(s,a) + \alpha[r(s,a) + \xi \text{max}_{a'}\{Q(s',a')\} - Q(s,a)]$, where s' is taken in $F(s,a)$ by sampling according to the distribution $P(s'|s,a)$. This approach is very useful in reinforcement learning.

Value Iteration algorithm has a polynomial complexity in $|S|$ and $|A|$. Unfortunately, most of the time S has a huge size: it is exponential in the number of the state variables. There are a few more scalable approaches, using heuristics and hierarchical techniques, *e.g.*, [8, 121, 97, 96, 120]). Probabilistic planning is a very active research area with may open problems.

Given a policy π, the controller for an MDP is extremely simple. Just iterate over two steps:

- observe the state s
- execute the action $\pi(s)$

until reaching a goal state or some other stopping conditions.

The MDP approach offer several runtime advantages. It explicitly manages the nondeterminism and uncertainty. It can be extended to take into account Partially Observable domains [23]. Modeling a domain as an MDP is a difficult task, but the MDP formulation can be combined with learning techniques (see section 6). This explains the success of these approaches in many robotics applications which will be discussed later.

4.4 Integrating of Motion and Task Planning

Task planning and motion planning are two different problems that use distinct mathematical representations. The first is concerned with causal relationship regarding the effects of abstract actions, the second is concerned with computational geometry and dynamics. In simple cases a robot can decouple the two problems: task planning produces abstract actions whose refinement requires, possibly, motion planning. The two problems are however coupled for constrained environments and complex tasks. For example, moving objects in a storage room can make the motion impossible if the task is

not appropriately organized. Let us discuss here some approaches to the integration of motion and task planning.

The Asymov planner [25] combines a state-space search approach for task planning (using the FF planner [63]) with a search in the configuration space for motion planning. It defines a *place* as a state in the task planning space, as well as a range of free configurations in $\mathcal{C}_f$. A place is a bridge between the two search spaces. These two spaces are not explicitly constructed, but for every found task state, Asymov checks that there are some reachable configurations in $\mathcal{C}_f$. This approach has been extended to multi-robot problems cooperating over a joint task, e.g. two robots assembling a large furniture such as a diner table in a cluttered environment.

Another interesting technique uses hierarchical planning in a so-called *angelic* approach [127] (the term is borrowed from *"angelic nondeterminism"* which assumes that out of several issues, the best one can be chosen). An abstract action can be decomposed in different ways. An abstract plan is based on abstract actions; its set of possible decompositions is a subset of the product of all possible decompositions of its actions, some of which are not compatible. It is not necessary to ensure that all the decompositions are feasible. A plan is acceptable if it has at least one feasible decomposition. Indeed, the decomposition is not made randomly. The robot decomposes, when needed, each abstract action by choosing a feasible decomposition, if there is one. The idea is to rely on a lower bound of the set of feasible decompositions of an abstract plan such as to make sure that this set is not empty. These lower bounds are computed by running simulations of action decompositions into elementary steps, using random values of state variables. The planner relies on these estimates for searching in the abstract state space.

The approach of Kaelbling and Lozano-Perez [71] illustrates another hierarchical integration of task and motion planning. When planning at the abstract level, estimates regarding geometric information are computed with so-called *Geometric Advisers*. These advisers do not solve completely the motion planning problem submitted to them, but provide information about how feasible is a given step that enables the abstract search to continue until reaching a complete plan. When the produced plan is executed, each step that requires movements triggers a full motion planning. This approach relies on two strong assumptions: geometric preconditions of abstract actions can be calculated quickly and efficiently (by the geometric adviser); subgoals resulting from decomposition of action are executable in sequence. The approach is not complete, i.e., the geometric refinement of a planned abstract action may fail. However, for problems where actions are reversible at a reasonable cost (i.e., allowing for backtracking at the execution level) the approach is efficient and robust.

5 Interaction

Most of the approaches presented above make the assumption that there is a single agent in the environment: the robot performing the task. But complex missions may require the participation of several humans and robots. Several approaches address these issues of interaction. For example, Simmons et al. [111] proposes the Syndicate architecture, an extension to 3T [19], which allows the cooperation of several robots in collaboration with a human, for the assembly of large structures. Fong et al. [47] offers an architecture to define interaction models (tasks, teams, resources, human) needed for the cooperation of a team of astronauts and extra-planetary rovers. In the next two sections, we examine these increasingly common interactions and how they are accounted for in the planning process.

5.1 Multi-Robot Interaction

Sometimes, to achieve a complex mission, it is necessary to deploy multiple robots. Several approaches to the problems of mission planning and execution in a multi-robot framework have been developed. We may distinguish several types of problems depending on the following features:

- planning is centralized or distributed,
- plan execution by each agent is independent or coordinated,
- planning is done before acting or made as the robots proceed,
- execution failures are repaired, and if yes at which level,
- the robots can communicate between them for coordination and planning.

Many research focuses on multi-robot motion planning, with geometric and kinematic representations (see section 3), and decomposition techniques generic enough to lead to distributed implementations [36]. Recent results, e.g., [17], allow to efficiently take into account relative position constraints between the robots as well as missions featuring several sites to visit.

The Martha project illustrates an approach to manage a fleet of robots handling containers in ports and airports [4]. The allocation of tasks to robots is centralized, but on a limited horizon. Planning, execution, refinement and coordination needed for the navigation of robots and the sharing of spacial resources in the environment are distributed. Robots negotiate among themselves the navigation in the environment, which is divided into cells (e.g., intersection crossing, convoy mode, overtaking), and also negotiate their path inside these cells. The deployed system assumes a reliable local communication. Execution deadlocks between multiple robots are correctly detected by the coordination algorithm, and one of the robots automatically takes control and produces a plan that it redistributes to the other robots with which it conflicts.

Other works propose an allocation of tasks by an auction mechanism [35] to assign tasks to robots (cells crossing/surveying). Tovey et al. [124] propose a mechanism to generate appropriate auction rules adapted to the particular goal a group of exploration rover has (minimize the sum of the distances, minimize the maximum travelled distance of all robots, minimize the average time to the targets, etc.). In [132, 26], the authors apply a similar technique to tasks and subtasks of an HTN plan as it is built. Each robot can win the bids on a task, then decompose into sub-tasks following an HTN method, and auction all or part of the sub-tasks. After the initial distribution of tasks, robots maintain, during execution, the ability to auction tasks they failed to perform. Moreover, communication in these systems is not permanent and complete, thus the replanning/redistribution phases must be planned in advance.

5.2 Human - Robot Interaction

The development of service robots raises some issues with respect to human-robot interaction [57]. We focus here on approaches which are concerned with planning and the models they use.

Interactive planning (or *mixed initiative* planning), i.e., planning while keeping humans in the loop, is used in various areas. The operator takes part in the search for a plan to make choices and help the planner solve the problem.

Planning for human–robot interaction raises a completely different issue: the plan is generated automatically by the robot, but must explicitly take into account the interaction with the human while executing the plan and even in some cases, plan for a shared execution. To this end, the planner has some models (learned or programmed) of human behaviors [12]. These models specify how humans

behave with respect to the robot, what are the behaviors of the robot which are acceptable to humans. They also specify how planned actions translate into commands of the robot [116].

Various planners have been adapted to take into account the role of the human in the plans produced. Generally, these are multi-agents planners, which have been modified to consider the human as one of the agents. [24] propose an extension to GOLOG/GOLEX to plan the mission of a robot museum interacting with visitors. The approach used in [21] is based on MAPL, a PDDL based multi-agent planner to represent the beliefs of the various agents, the actions and the perception operators. A compiler then translates these PDDL models. Planning is then performed by the FF planner [63].

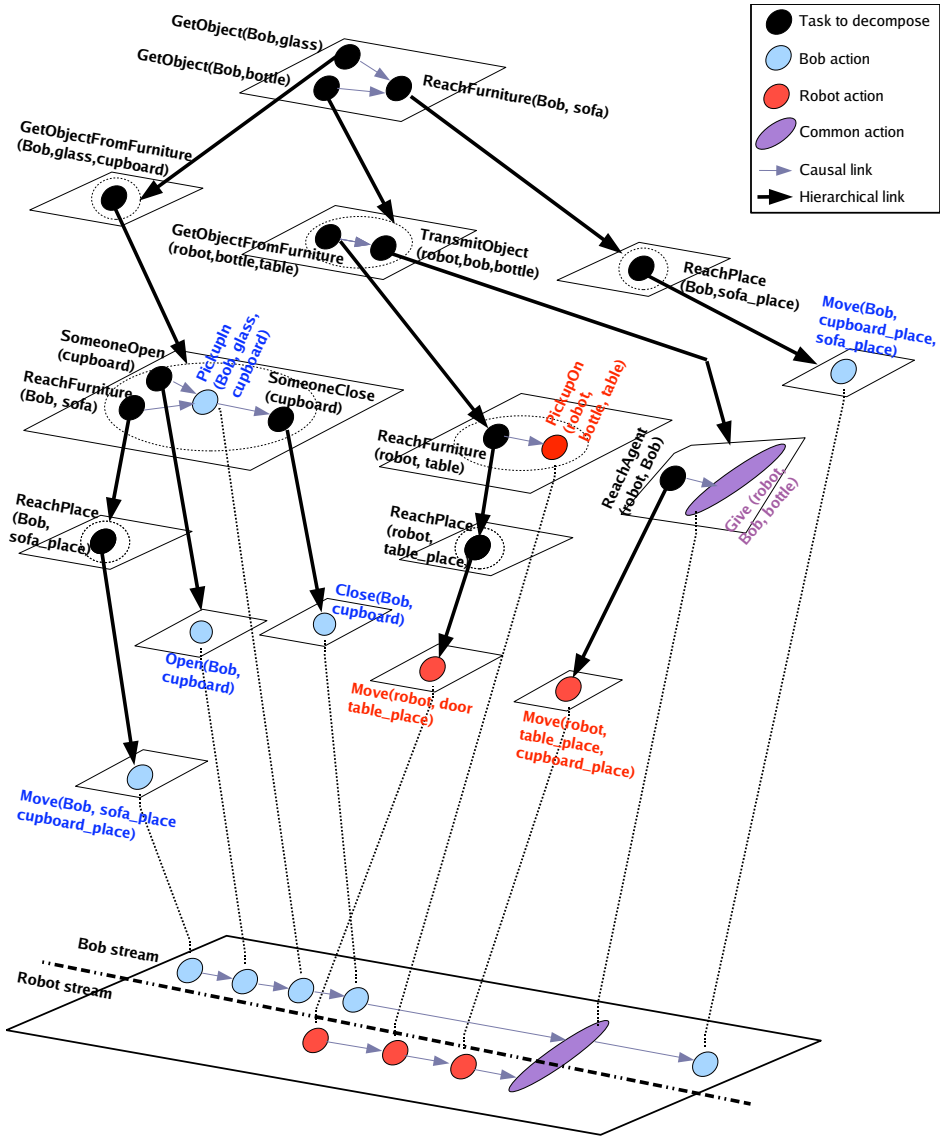


Figure 16: A plan produced by HATP with human–robot interaction: the tasks (in black) are decomposed into primitive actions for the robot (in blue), actions of the human (in red), and joint actions (in purple), which require a synchronization [3].

The HATP planner [3] plans in the context of human–robot interactions (e.g., for service robotics).

This planner extends the HTNs formalism to create plans containing two execution threads, one for the robot and one for the human who interacts with the robot. Figure 16 shows a plan produced by HATP. One can distinguish the execution of the robot thread (red) and the user thread (in blue). HATP differs from the classical HTN planning on several points. Task models and refinement methods involve human and robot. Furthermore, while the plan is produced, the system considers social rules to produce plans which are deemed acceptable and understandable to humans. For example, the robot will favor an action where it gives an object directly to the human rather than an action where it just lays the object before him. Similarly, when interacting with humans, the robot will favor a position where it faces the human, and make slower movements when it approaches him. When executing the plan, the robot must interpret and recognize human actions to properly carry out its plan. For example, if during a task the robot proposes a tool to human, and if the human loses interest, the robot should not insist, and wait for the attention of the human to return back to the robot. These good behavior recipes are not just cosmetic, they enable a more natural interaction between humans and robots.

6 Learning

Machine learning techniques have a very successful impact in many areas, and particularly in robotics. A variety of computational learning techniques are developed at various levels in robotics, from the sensory-motor level to the acquisition of tasks or environment models. A good coverage of recent learning techniques robotics can be found in [106]. We already covered environment mapping issues in section 3.2. Basic statistical learning techniques are quite useful, in particular for object recognition, but they are not specific to robotics. Let us review here two approaches that are more specific to robotics: reinforcement learning and learning from demonstration.

6.1 Reinforcement Learning

Reinforcement Learning (RL) refers to methods that improve the performance of a learner by direct interaction with the world [70, 117]. It is based on a trial and error approach. A robot learns how to act by maximizing the long term perceived benefit of its actions. Generally in RL, the learner has no teacher providing examples of good behaviors in certain situations or advices about how to choose actions. The only feedback given to the robot at each step is a scalar: the reward associated with the performed action. As long as the robot has not tried all feasible actions in all encountered situations, it will not be sure that it uses the best actions. Reinforcement learning has to solve the compromise of *exploration* vs *exploitation*: the robot must make the most of what it already knows to maximize the benefit of its behavior. To find the best one, it must explore the options that are not known enough.

To introduce the approach, consider the very simple case where a single action solves completely the task at hand and has no impact on the next task. Suppose a stationary environment and nonnegative rewards. Let $r_i(a) > 0$ be the reward received after running an action a at the i^{th} time. We can estimate the quality $Q(a)$ of an action a that has been executed k_a times by its average award:

$$Q(a) = \begin{cases} q_0 & \text{if } k_a = 0, \\ \frac{1}{k_a} \sum_{i=0}^{k_a} r_i(a) & \text{otherwise.} \end{cases} \tag{9}$$

This estimate is maintained by incremental updates:

$$Q(a) \leftarrow Q(a) + \alpha[r_{k_a+1}(a) - Q(a)], \text{with} \quad \alpha = \frac{1}{k_a + 1} \tag{10}$$

When $\forall a, k_a \rightarrow \infty$, the choice of the action which maximizes the sum of rewards is given by $\text{argmax}_a\{Q(a)\}$. However, as long as the exploration of alternatives has not been sufficient, the robot will use other options, according to various heuristics. One may define a function $\text{Select}_a\{Q(a)\}$ by one of the following methods:

- $\text{Select}_a\{Q(a)\} = \text{argmax}_a\{Q(a)\}$ with probability $(1 - \epsilon)$, and a randomly drawn action other $\text{argmax}_a\{Q(a)\}$ with probability ϵ, where ϵ is decreasing with experience,
- $\text{Select}_a\{Q(a)\}$ chooses an action according to a probabilistic sampling distribution, for example, with Boltzmann sampling, according to a probability distribution given by $e^{\frac{Q(a)}{\tau}}$, where τ is decreasing with experience.

When the environment is stationary, the update of $Q(a)$ with Equation 10 after executing an action a becomes increasingly weak with large k_a. If the environment is not stationary, we can keep $\alpha < 1$ constant. Note also that the initialization value q_0 fosters exploration if q_0 is high with respect to other rewards. For example, if $q_0 = r_{max}$, the maximum reward, new actions will be preferred to those already tested.

With these basics notions, let us now consider the interesting case where a task is performed by the combination of several actions, each interfering with the following ones, influencing the overall success of the task and the sum of rewards. The framework generally used is that of Markov decision processes introduced previously (Section 4.3). The robot seeks to learn an optimal policy that maximizes the value $V(s)$ over all s. This value is estimated from the observed rewards of the chosen actions. A major problem is how to distribute rewards over the entire task. Indeed, the rewards give an immediate return in the short term, while the quality of achievement of the task to be maximized is described by the long term sum of rewards over some horizon.

One approach is to learn the MDP model then to apply planning techniques to find the optimal policy and use it. Learning a model means collecting enough statistics through an exploratory phase to estimate the probability distributions $P(s'|s, a)$ and the rewards $r(s, a)$. An interesting application of this direct approach combines a model learning technique with a receding horizon planning algorithm [91]. It was illustrated for learning indoor navigation skills, combining different motion, localization and control modalities. The approach is applicable to any task for which the robot has several alternative methods whose performance depend on local features of the environment. The performance function is difficult to model. It is learned as an MDP whose state space is an abstract control space, which focuses on the features of the environment and current task context (including the method in use); actions correspond to available methods for performing the task. The state space is of small size (a few thousands states) which allows computing an optimal policy at each step of a receding horizon planning.

This direct approach requires a costly exploratory phase to estimate the model. It is often better to start performing the task at hand, given what is known, while continuing to learn, *i.e.*, combine the two phases of acquiring a model and finding the best action for the current model. *Q-learning* algorithm meet this objectives while avoiding to build explicitly the MDP model.

Let us use the MDP notation introduced earlier, in particular $r(s, a)$ is the observed reward after performing action a in state s, and $Q(s, a)$ is the estimated quality a in s at current time. $Q^*(s, a)$, as given by Equation 8, is unknown but it can be estimated by the expression:

$$Q(s, a) = r(s, a) + \xi \sum_{s \in F(s,a)} P(s'|s, a) \text{max}_{a'}\{Q(s', a')\} \tag{11}$$

The basic idea of the *Q-learning* algorithm (17) is to perform an incremental update of $Q(s, a)$, similar

to Equation 10. This update does not use the unknown probability parameters of the model, but the quality of successor states s', as observed in the current experience. This update is given in line *(i)* in the algorithm below.

Q-learning
  until *Termination* do
    $a \leftarrow \text{Select}_a\{Q(s,a)\}$
    execute action a
    observe $r(s,a)$ and resulting state s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r(s,a) + \xi \text{max}_{a'}\{Q(s',a')\} - Q(s,a)]$ ▷ *(i)*
    $s \leftarrow s'$

Figure 17: Q-learning algorithm

The values of $Q(s,a)$ are initialized arbitrarily. The function $\text{Select}_a\{Q(s,a)\}$ favors $\text{argmax}_a\{Q(s,a)\}$ while allowing for the exploration of non maximal action with a frequency decreasing with experience. The parameter $\alpha \in [0,1]$ is set empirically. When it is close to 1, Q follows the last observed values by weighting down previous experience of a in s; when it is close to zero, the previous experience is more important and Q changes marginally. α can be decreasing with the number of instances (s,a) encountered.

A variant of this algorithm (known as " SARSA" for State, Action, Reward, State, Action) takes into account a sequence of two steps (s,a,s',a') before performing the update of the estimated quality of a in s by $Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a) + \xi Q(s',a') - Q(s,a)]$. One can prove the asymptotic convergence of these two algorithms to optimal policies.

Other model-free reinforcement learning algorithms proceed by updating the value function $V(s)$ rather then the function $Q(s,a)$. Updates are performed over tuples (s,a,s') in a similar way: $V(s) \leftarrow V(s) + \alpha[r(s,a) + \xi V(s') - V(s)]$. This algorithm called $TD(0)$, is combined with a Select function permitting exploration. It is part of a family of algorithms $TD(\lambda)$ which perform updates over all states, with a weight depending on the frequency of meeting each state.

Let us also mention the DYNA algorithm and its variants that combine learning and planning: it maintains and updates an estimate of probabilities $P(s'|s,a)$ and rewards $r(s,a)$; at each step two updates are performed, a *Q-learning* type with $Q(s,a) \leftarrow r(s,a) + \xi \sum_{s'} P(s'|s,a) \text{max}_{a'}\{Q(s',a')\}$, for the current s and a, and a Value-Iteration type for other couples (state, action) chosen randomly or according to certain priority rules, taking into account new estimates. Here, the experience allows to estimate the model and the current policy. The estimated model in turn allows to improve the policy. Each step is more computationally expensive than in *Q-Learning*, but the convergence occurs more rapidly in the number of experimental steps.

Reinforcement learning is widely used in robotics, but it is rarely implemented with explicit state space and tables of values $V(s)$ or $Q(s,a)$. The state space is generally continuous; it includes the configuration space of the robot and its environment. Even if one manages to discretize the state space appropriately (e.g., in grid type environment approaches), the astronomic size of S makes the explicit representation of S impractical. Moreover, the above algorithms are used to learn a good behavior for states encountered during learning phase, but they are not useful for states that have never been encountered: they do not allow to generalize. If one uses a continuous state space with a metric distance, one can make the reasonable assumption that *nearby* states are typically associated with close estimates of $V(s)$ or $Q(s,a)$, and thus use similar policies. Parametric approaches implement this idea.

Here S and A are described by two vectors of state and control variables. Let $\theta = (\theta_1, \ldots, \theta_n)$ be a vector of parameters. We assume that $Q(s,a)$ can be approximated parametrically by $Q_\theta(s,a)$, as a function of θ. This function is given a priori, e.g., a linear function of state and control variables. Learning involves estimating the parameters θ of this model. Q-Learning algorithm is the same as above, except that the update *(i)* does not change values in a table, but the parameters of $Q_\theta(s,a)$. The process generally involves minimizing the mean squared error of Q with respect to Q^*. The latter is estimated at each iteration from the last observed update. The gradient algorithm follows this formulation:

$$\begin{aligned} \theta &\leftarrow \theta - \frac{1}{2}\alpha\nabla_\theta[r(s,a) + \xi\max_{a'}\{Q_\theta(s',a')\} - Q_\theta(s,a)]^2 \\ &\leftarrow \theta + \alpha[r(s,a) + \xi\max_{a'}\{Q_\theta(s',a')\} - Q_\theta(s,a)]\frac{\partial Q_\theta(s,a)}{\partial\theta} \end{aligned} \tag{12}$$

This last expression replaces the *(i)* in the previous algorithm for each parameter θ_i. A similar formulation can be obtained for the estimate of V_θ.

Reinforcement learning with a parametric approach is used with success in robotics. It has been implemented in simple applications, for example to stabilize an inverse pendulum or to play darts, and in more complex demonstration, such as helicopter acrobatic flying [1, 31]. One of the main problems of these approaches is defining the action rewards.

Indeed, the previous algorithms indicates improperly "observe $r(s,a)$". But rewards are seldom directly observable by the the robot. One must provide the means to estimate the reward according to what is perceived. Sometimes the function $r(s,a)$ is easy to specify, for example as the deviation from equilibrium for a stabilization task, or the deviation from the target for tracking task. But often it is not, for example, how to specify the rewards of elementary actions for the task of driving a car?

This issue leads to the *inverse reinforcement learning* problem [2]. Here, the teacher gives the optimal behavior, the problem is to find the corresponding reward function that generates this behavior. In the case of an explicit finite MDP, the problem reduces to the following formulation (derived directly from Equation 8): we know $\pi^*(s)$ for all s; we can express $Q(s,a)$ as a function of the unknown values of $r(s,a)$ and we want $Q(s,a)$ to be maximal for $a = \pi^*(s)$. This formulation is under-specified: it has infinitely many solutions that are of no interest. It is extended with an additional criterion, for example maximize the expression: $\sum_s[Q(s,\pi^*(s)) - \max_{a\neq\pi^*(s)}Q(s,a)]$. The problem is solved by linear programming.

In parametric approaches we also define rewards r_θ as a function (usually linear) of state and control variables and seek to estimate its parameters. The previous formulation is not directly applicable because π^* is known for a small number of state samples. However the main constraint that the distribution of states generated by r_θ must be the same as the one provided by the teacher leads to a formulation that one can solve iteratively. Each iteration combines two steps, a quadratic programming optimization criterion and a dynamic programming similar to Value-Iteration.

As the reader has certainly noticed, these approaches are akin to the techniques used for inverse problems. They are also related to learning from demonstration techniques, discussed next.

6.2 Learning from Demonstration

As underlined above, the specification of the reward functions needed in reinforcement learning is far from obvious. Moreover, it is rare to have a fully observable Markov state space. We know how to transform a state space into a Markovian one, but this requires significant engineering and adds generally unobservable components. The complexity of learning and planning techniques in

partially observable MDP is prohibitive. Moreover, the experimental complexity (in the total number of needed trials) is generally much more expensive in robotics than the computational complexity. Reinforcement learning requires for converging a very large number of experiments. Finally, it is common that the task to learn cannot be treated as a simple sequence of pairs *(state, action)*. It requires a plan or a control structure, such as repeating a subsequence of actions until a termination condition. For these reasons, learning from demonstration is a good alternative when the robot can benefit of the demonstrations of a teacher.

In learning from demonstration (see the survey of Argall et al. [7]), a teacher gives to the robot the appropriate actions in well-chosen settings. This allows the teacher to control the learning process and gradually focus learning on the most difficult part of the task. The robot generalizes from the teacher demonstrations and learns the required behavior, which can be expressed as a policy in simple cases, or as a mapping from sensory states to plans in the general case. Learning from demonstration is akin to supervised learning. However in supervised learning, the teacher provides directly correct labels for training cases. Learning from demonstration involves other issues about how to map the teacher's sensing and acting spaces to those of the robot learner.

In the simplest setting, learning from demonstration reduces to acquiring the correct behavior from teleoperated training cases. The teacher acts directly in the actuator space and the proprioceptive sensor space of the robot. The latter learns actions directly as its own control environment. These approaches have resulted in many implementations, such as those presented by Sigaud and Peters [106] or Peters and Ng [95].

In the general case, the teacher acts with its own actuators rather than those of the robot to illustrate the movements and manipulations she wants to teach. The robot observes the teacher from outside. In order to learn, the robot must build up a double mapping:

- a sensory mapping to interpret the observed demonstrations, and
- a control mapping to transpose the demonstrated actions to its own actuators.

This double mapping is very complex. It often limits learning from demonstration and requires the teacher to have pedagogic skills, that is, to understand at a low level how the robot will be able to map the teacher demonstrations in its own actuation capabilities.

Moreover, learning from demonstration can be performed with or without the acquisition of a task model. The first case corresponds generally to inverse reinforcement learning. In the latter case, learning can give rise to the acquisition of a sensory-motor mapping. Here, the techniques use supervised learning, by classification or regression. Finally, learning can also lead to the acquisition of a mapping from sensory states to plans. These can be obtained by plan recognition methods. Plans can also be synthesized from the teacher specifications of operators and goals (final and intermediate) associated with observed sensory states.

Approaches relying plan recognition and synthesis allow to address a significantly more general class of behaviors that can be demonstrated by a teacher and acquired by a robot (including iterative actions and control structures). They also permit extended generalization since they lead to acquire basic principles and use the robot planning capabilities. They are finally more natural and easier for the teacher, since the teacher's actions are interpreted in terms of their effects on the environment rather than their sole order in a sequence of commands. They are illustrated for example in the work of Nicolescu and Mataric [94], Rybski et al. [103], but remain at a quite preliminary stage.

7 Integration and software architecture

Beyond the physical integration of mechanical, electrical, electronic, etc. systems, a robot is also a complex information processing system, from data acquisition to electronic commands. It integrates, various processing paradigms from real time control loops, with a hierarchy of response time, up to decisional functions conferring the autonomy and robustness required by the variability of tasks and environments. The integration of these processes should be based on architectures that defines how to articulate all these components, how they communicate and how they share data and computing resources. In any case, they must provide development methodologies to allow programming, integrating and testing of the different modules. They should provide tools and libraries to facilitate the development and deployment of the various components on the robot, especially those of interest to us: planning and execution control.

7.1 Architecture Paradigms

Most robot architectures are developed following different paradigms:

Reactive Architectures The reactive architectures, popularized by the subsumption architecture of Brooks [22], are conceptually simple. They are composed of modules which connect sensors and effectors through an internal state machine. These modules are hierarchically organized with the outputs of some which can inhibit or weight the outputs or the composition of others. These architectures were relatively popular because they are a priori easy to setup. They do not require a model of the world (the world is its own model) and are adapted to reactive simple tasks, without planning. A robot like the Roomba which has most likely been developed following this concept, achieves its task plan. But there is no quality nor efficiency objective formally pursued. Ultimately, these architectures still remain popular and are used in mono task applications. But for application associated to the variability of tasks and environments, the programming and setting of inhibitors/weights quickly becomes infeasible.

Hierarchical Architectures The hierarchical architectures and layered architectures remain the most popular in robotics. They propose to organize all robot software in two or three layers, from the functional level up to the decisional level (planning and acting). The former includes the sensory-motor functions to control sensors, effectors, and to perform the associated processing. In some instances, an intermediate level is used for execution control to verify safety conditions. Tools are typically associated with these architectures to ease the integration of the different components. Thus, the LAAS architecture [67] relies on GenoM to develop functional modules (see Figure 18), and various tools (R2C, OpenPRS, Transgen, IxTeT) for execution, supervision and tasks planning. The CLARATy architecture provides C++ basic classes which facilitate the development of the functional layer. TDL and ASPEN/Casper respectively implement the acting and the planning component.

Architectures Teleo-Reactive More recently, teleo-reactive architectures such as IDEA [44] and T-ReX [100] have emerged. They propose to decompose the problem in agents rather than in layers. Each agent [6] consists of a planner/actor tandem. It produces plans by establishing sequences of tokens on timelines representing the evolution of the state variables of the system, and ensures their execution. Planning is performed by a temporal planner (e.g. Europa [48] or APSI [49]) based on Allen [5]

[6] Agents are called reactors in the T-ReX terminology.

temporal intervals logic. These agents are organized depending on the relevant state variables. Each agent has an adapted latency, execution period and planning horizons. They communicate between them by sharing some timelines (with priority rules on which agent can change value on a shared timelines) with a dispatcher responsible for integrating the new values of token depending on the execution.

These architectures have two advantages. They have a unified agent architecture model (even functional modules are expected to be developed using this paradigm). They use the same modeling language, providing an overall consistency of models. This architecture has been deployed in a number of experiments, notably: an autonomous underwater vehicle [84], and on the PR2 robot from Willow Garage [83].

However, the deployment of these approaches is hindered by two problems. The first is performance. Agents are seldom able to properly plan fast enough (e.g. in less than a second), to be used to model functional modules. The second is the difficulty to develop the model (e.g. writing compatibilities and constraints), especially when modeling non-nominal cases.

Finally, not in these categories, there are reactive hybrid architectures that add one or more planners to reactive modules. The role of the planners is to propose plans to configure, via a coordination system, the activities of the reactive modules. The difficulty is to write this coordination module. Beaudry et al. [11] illustrate a proposal in this regard that combines a motion planner and an HTN planner which explicitly manage time; this approach seems promising for non-critical applications.

7.2 Robustness, Validation and Verification

The robustness of the software deployed on a robot poses a major problem. A first step is to robustify key components to overcome the environmental hazards, sensory noise, and the great variability of environments. One can require that a functional module, handling a sensory-motor function, knows his range of use. It should know and recognize when its data cannot be properly used, to allow corrective actions to be taken. For example, a component that makes stereo vision will recognize when its cameras are not properly calibrated; similarly, monitoring the torque on the wheel, a module that manages locomotion should detect wheel slippage or wheel blockage. Similarly, the components responsible for decision making and using formal approaches (e.g., constraints based, proposition logic,etc.) should ensure that the produced plans will not lead to undesirable states.

However, the composition of these components, as robust as they individually are, does not lead directly to an overall safety properties of the robot. For example the component taking scientific images and the locomotion component can both be correct, but all possible executions of these two components together may not be acceptable, e.g., the parameters to capture high-resolution images while moving are constrained (to avoid blurry images). The safety and robustness of embedded real-time systems [62] has been an active field for many year. With respect to robotics, one has also to consider the requirements for decisional autonomy. Modeling languages, such as UML [69] and AADL [39], can be used. They provide tools and specification methods. But we need to go further with formal approaches that provide validation and verification.

In the robotics domain, one should mention Orccad [112] and MAESTRO [34], which are based on the synchronous languages paradigm (Esterel) and have been used to implement robotic controller. Simmons et al. [110] propose a model checking approach to verify the robot controller written in the TDL language [109]. Within the LAAS architecture, the R2C [99] models all the constraints that we want to ensure and it formally checks at runtime that the commands sent by the decisional level are consistent with the model and the current state of the robot. Some research are also interested

in verifying that the code executed by the functional modules of a robot formally satisfy its logical specification [50] (at the cost of logically annotating all the code used in the module).

More recently, some work around GenoM intended to produce a formal model of the entire functional layer of a robot [15]. The modeling is based on the BIP formalism (Behavior, Interaction, Priority) [10] and exploits the fact that each GenoM module is an instance of a generic module (see Figure 18).

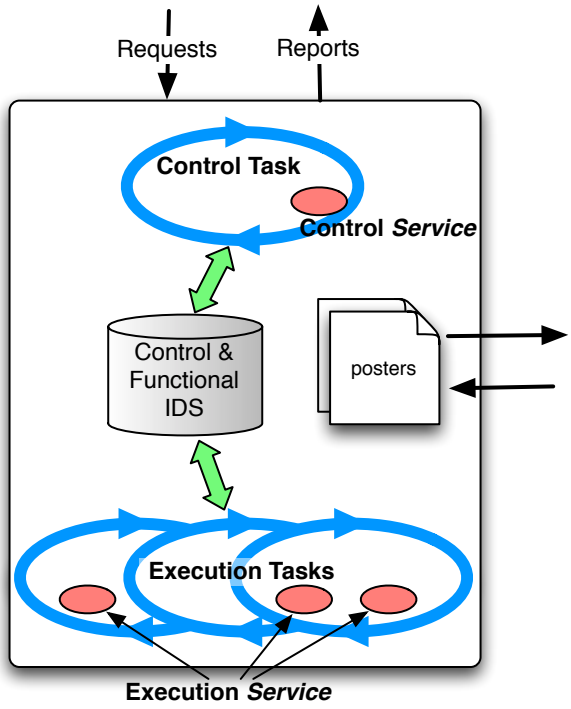


Figure 18: The internal organization of a GenoM module. The control flow is organized as follows: the control task receives requests and starts the execution of corresponding services in the execution tasks. When execution is complete, the control task returns a report to the caller. Writing or reading posters provide the data flow between the modules.

The BIP formalism [10] provides a methodology to model embedded systems from *(i)* atomic components; *(ii)* connectors that define the interactions possible between the ports of atomic components; and *(iii)* a priority relation, to select among the valid interactions. An atomic component is defined by: *(i)* a set of ports $P = \{p_1, \ldots, p_n\}$ which are used for synchronization with the other components; *(ii)* a set of states $S = \{s_1, \ldots, s_k\}$ representing states where the component awaits synchronizations; *(iii)* a set of local variables V , and *(iv)* a set of transitions. A transition is a tuple of the form (s, p, G_p, f_p, s'), representing a step from state s to s'. The transition may modify the local variables when executing the function $f_p : V \rightarrow V$. A transition is valid *iff* the guard G_p (boolean condition on V) is true and the interaction on p is possible. For example, the transition *empty* to *full* in Figure 19 is possible if $x > 0$, and if the interaction *in* is possible. The variable y then takes the value $f(x)$. The transition from *full* to *empty* has not guard, but requires an interaction on port *out*.

In this approach, all the components of a generic GenoM module are modeled in BIP. All modules of the functional layer are obtained by recomposition of these basic BIP models. It should be noted that the executable code associated to the state of the original GenoM service automata are now within

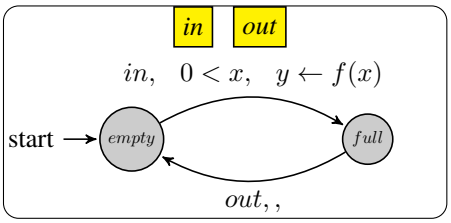


Figure 19: A simple example of an atomic BIP component comprising two states and two transitions. Transition from *empty* to *full* is associated with an interaction on port in, a boolean condition $0 < x$, and a change of the value of y. The transition in the other direction requires only an interaction with the out port.

the $f(x)$ transitions function in the BIP model. This approach is extended to the complete functional layer of a robot, and provides an extremely fine grained formal model of the system considered (e.g., the state in which each component is, the possible interactions at any time, etc.). This model is then used by the BIP Engine (an automata player which checks online guards and interactions of the entire model, and fire the valid transitions) to control the execution on the actual robot. This model can also be verified and validated with formal tools like D-Finder [14]. This formal verification method composes *component invariants* ϕ_i which define for each component a logical property it satisfies, and the *interaction invariants* Ψ that logically define the possible interactions γ between the components considered. The extraction of these invariants is automatic. The inference rule:

$$\text{if } (\bigwedge_i \phi_i) \wedge \Psi \Rightarrow \Phi \text{ then } \|_\gamma \{B_i\}_i < \Phi >$$

specifies that if the conjunction of invariants $(\bigwedge_i \phi_i) \wedge \Psi$ (overestimation of the reachable states) implies a formula Φ, then the parallel composition $\|_\gamma \{B_i\}_i$ also satisfies Φ.

This method allows, among other things, to verify that there is no deadlock in the system or to check safety properties. Note that this technique based on components and interactions invariants can potentially take into account search spaces larger than the ones model checking techniques can handle.

8 Conclusion

In this paper, we presented an overview of the state of the art at the intersection of two broad fields which are Robotics and Artificial Intelligence. We reviewed models and techniques for addressing problems of planning and execution control of movements and tasks, interaction and learning. We discussed how to integrate decision-making functions with sensory-motor functions within a robot architecture. Most of these issues have been outlined very synthetically. Some were slightly detailed to provide the reader with illustrative frequently used representations and algorithms.

As underlined in the introduction, robotics is a multidisciplinary field. Significant progress in robotics can be expected from major advances in its basic disciplines. Further, robot can be a catalyst research target to advance these disciplines. For example, a light and fast mechanical gripper with high dexterity, an inexpensive accurate 3D range sensor, or an image recognition algorithm with broad and reliable performances for ordinary objects that can be found in a house or store, will substantially enrich the functional capabilities of current platforms.

But, as we have also pointed out, robotics research is primarily integrative. One can certainly make progress in terms of basic components for the handling some particular task or environment. But the autonomy of a machine when facing a diversity of environments and tasks requires progress in the integrated *perception - decision - action* control loop.

This loop is at the core of research in robotics. It requires explicit models of objects at various levels, from their physical appearance to their functions. It also requires models of activities, events and processes that constitute the environment and its agents, including the robot. It requires knowledge representations adapted to these models. These models are mathematically heterogeneous, that is continuous/discrete, symbolic/numeric, geometric/topologic, deterministic/stochastic, etc. In robotics, the term "knowledge representations" is necessarily plural. It also requires a variety of learning techniques to acquire and improve these models. This is the research agenda, for which we have reviewed the progress over the past two or three decades, and on which more work remains to be done. This agenda is as relevant to *self-contained* robots, which integrate all their components on a single platform, as well as to *distributed* robots. Distribution is also an important item of this agenda. It concerns the distribution of cognitive functions over the components and functions of a single robot, as well as the distribution of robotics functions over a networks of sensors, actuators and processing resources on a large scale.

It can also be argued that the *perception - decision - action* control loop is at the core of AI research. Continues progress is being made in all individual subfields of AI. For example, statistical and hybrid techniques have led to dramatic advances in automatic natural language processing, illustrated for example by the victory of the WATSON system in the question/answer game "Jeopardy" [41]. Representations coupling first-order logic and uncertainty management, such as probabilistic first order logic [86], open remarkable opportunities, especially for the problems of planning and learning that we discussed here.

But the AI objective, namely *to understand, model and implement intelligence*, is seen by many researchers as being expressed in the *perception - decision - action* control loop. Consider the problem of "*anchoring*", i.e., maintaining a mapping between a symbol and the sensory data related to the same physical object [32], or the more general problem of "*symbol grounding*" [60], i.e., associating a symbol, in its context, to a signified content, object, concept or property. These problems requires the coupling of cognitive mechanisms to sensory-motor functions able to interact independently with the world to which symbols refer (the level T3 of the Turing test of Harnad [61]). For both fields, the coupling of Robotics and AI remains a very fertile research area.

References

[1] Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y. (2006). An Application of Reinforcement Learning to Aerobatic Helicopter Flight. In *Neural Information Processing Systems (NIPS)*, pages 1–8.

[2] Abbeel, P. and Ng, A. Y. (2010). Inverse Reinforcement Learning. In Sammut, C. and Webb, G. I., editors, *Encyclopedia of Machine Learning*, pages 554–558. Springer.

[3] Alami, R., Clodic, A., Montreuil, V., Sisbot, E., and Chatila, R. (2006). Toward Human-Aware Robot Task Planning. In *AAAI Spring Symposium: To boldly go where no human-robot team has gone before*, pages 39–46.

[4] Alami, R., Fleury, S., Herrb, M., Ingrand, F., and Robert, F. (1998). Multi-Robot Cooperation in the MARTHA Project. *IEEE Robotics and Automation Magazine*, 5(1):36 –47.

[5] Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123 – 154.

[6] Antonelli, G., Fossen, T. I., and Yoerger, D. R. (2008). Underwater Robotics. In [105], pages 987–1008.

[7] Argall, B., Chernova, S., Veloso, M. M., and Browning, B. (2009). A Survey of Robot Learning from Demonstration. *Robotics and Autonomous Systems*, 57(5):469–483.

[8] Barry, J. L., Kaelbling, L. P., and Lozano-Pérez, T. (2011). DetH*: Approximate Hierarchical Solution of Large Markov Decision Processes. In *International Joint Conference on Artificial intelligence (IJCAI)*, pages 1928–1935.

[9] Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to Act Using Real-Time Dynamic Programming. *Artificial Intelligence*, 72(1-2):81–138.

[10] Basu, A., Bozga, M., and Sifakis, J. (2006). Modeling Heterogeneous Real-Time Components in BIP. In *International Conference on Software Engineering and Formal Methods (SEFM)*, pages 3–12.

[11] Beaudry, E., Létourneau, D., Kabanza, F., and Michaud, F. (2008). Reactive planning as a motivational source in a behavior-based architecture. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1848–1853.

[12] Beetz, M., Jain, D., Mösenlechner, L., and Tenorth, M. (2010). Towards Performing Everyday Manipulation Activities. *Robotics and Autonomous Systems*, 58(9):1085–1095.

[13] Beetz, M. and Mcdermott, D. (1997). Expressing Transformations of Structured Reactive Plans. In *European Conference on Planning (ECP)*, pages 64–76. Springer Publishers.

[14] Bensalem, S., Bozga, M., Nguyen, T.-H., and Sifakis, J. (2009). D-Finder: A Tool for Compositional Deadlock Detection and Verification. In *Computer Aided Verification (CAV)*, pages 614–619.

[15] Bensalem, S., de Silva, L., Ingrand, F., and Yan, R. (2011). A Verifiable and Correct-by-Construction Controller for Robot Functional Levels. *Journal of Software Engineering for Robotics*, 1(2):1–19.

[16] Bertsekas, D. P. (1995). *Dynamic Programming and Optimal Control*. Athena Scientific, Vol. 1 and 2.

[17] Bhattacharya, S., Likhachev, M., and Kumar, V. (2010). Multi-agent path planning with multiple tasks and distance constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 953–959.

[18] Bohren, J. and Cousins, S. (2010). The SMACH High-Level Executive [ROS News]. *IEEE Robotics and Automation Magazine*, 17(4):18 –20.

[19] Bonasso, R., Firby, R., Gat, E., Kortenkamp, D., Miller, D., and Slack, M. (1997). Experiences with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2/3):237–256.

[20] Bonet, B. and Geffner, H. (2006). Learning Depth-First Search: A Unified Approach to Heuristic Search in Deterministic and Non-Deterministic Settings, and Its Application to MDPs. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 142–151.

[21] Brenner, M., Hawes, N., Kelleher, J., and Wyatt, J. (2007). Mediating Between Qualitative and Quantitative Representations for Task-Orientated Human-Robot Interaction. In *International Joint Conference on Artificial intelligence (IJCAI)*, volume 7.

[22] Brooks, R. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2:14–23.

[23] Buffet, O. and Sigaud, O., editors (2008). *Processus Décisionnels de Markov en Intelligence Artificielle*. Trait IC2, vol.1 et 2. Hermes - Lavoisier.

[24] Burgard, W., Cremers, A., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., and Thrun, S. (1998). The Interactive Museum Tour-Guide Robot. In *National Conference on Artificial Intelligence (AAAI)*, pages 11–18.

[25] Cambon, S., Alami, R., and Gravot, F. (2009). A Hybrid Approach to Intricate Motion, Manipulation and Task Planning. *International Journal of Robotics Research*, 28(1):104–126.

[26] Cao, H., Lacroix, S., Ingrand, F., and Alami, R. (2010). Complex Tasks Allocation for Multi Robot Teams under Communication Constraints. In *5th National Conference on Control Architectures of Robots*, Douai, France.

[27] Chatila, R. and Laumond, J.-P. (1985). Position referencing and consistent world modeling for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 138–145.

[28] Chaumette, F. and Hutchinson, S. (2008). Visual Servoing and Visual Tracking. In [105], pages 563–583.

[29] Chien, S., Knight, R., Stechert, A., Sherwood, R., and Rabideau, G. (2000). Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling. In *AI Planning and Scheduling (AIPS)*.

[30] Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., and Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press.

[31] Coates, A., Abbeel, P., and Ng, A. Y. (2009). Apprenticeship learning for helicopter control. *Communication ACM*, 52(7):97–105.

[32] Coradeschi, S. and Saffiotti, A. (2003). An Introduction to the Anchoring Problem. *Robotics and Autonomous Systems*, 43(2-3):85–96.

[33] Corke, P. I., Roberts, J. M., Cunningham, J., and Hainsworth, D. (2008). Mining Robotics. In [105], pages 1127–1150.

[34] Coste-Maniere, E. and Turro, N. (1997). The MAESTRO Language and its Environment: Specification, Validation and Control of Robotic Missions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 836 –841 vol.2.

[35] Dias, M., Zlot, R., Kalra, N., and Stentz, A. (2006). Market-Based Multirobot Coordination: A Survey and Analysis. *Proceedings of the IEEE*, 94(7):1257 –1270.

[36] Erdmann, M. and Lozano-Pérez, T. (1987). On Multiple Moving Objects. *Algorithmica*, 2:477–521.

[37] Erol, K., Hendler, J., and Nau, D. S. (1994). UMCP: a Sound and Complete Procedure for Hierarchical Task-Network Planning. In *AI Planning and Scheduling (AIPS)*, pages 249–254.

[38] Estrada, C., Neira, J., and Tardós, J. D. (2005). Hierarchical SLAM: Real-Time Accurate Mapping of Large Environments. *IEEE Transactions on Robotics and Automation*, 21(4):588–596.

[39] Feiler, P. H., Lewis, B. A., and Vestal, S. (2006). The SAE Architecture Analysis & Design Language (AADL) A Standard for Engineering Performance Critical Systems. In *IEEE International Symposium on Computer-Aided Control Systems Design*, pages 1206–1211.

[40] Feron, E. and Johnson, E. N. (2008). Aerial Robotics. In [105], pages 1009–1029.

[41] Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A. A., Lally, A., Murdock, J. W., Nyberg, E., Prager, J., Schlaefer, N., and Welty, C. (2010). Building Watson: An Overview of the DeepQA Project. *AI Magazine*, Fall:59–79.

[42] Fikes, R. (1971). Monitored Execution of Robot Plans Produced by STRIPS. In *IFIP Congress*.

[43] Fikes, R. and Nilsson, N. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial intelligence*, 2(3-4):189–208.

[44] Finzi, A., Ingrand, F., and Muscettola, N. (2004). Model-Based Executive Control Through Reactive Planning for Autonomous Rovers. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 879–884.

[45] Firby, R. J. (1987). An Investigation into Reactive Planning in Complex Domains. In *National Conference on Artificial Intelligence (AAAI)*, pages 202–206. AAAI Press.

[46] Flint, A., Murray, D., and Reid, I. (2011). Manhattan Scene Understanding Using Monocular, Stereo, and 3D Features. In *Proc. International Conference on Computer Vision*.

[47] Fong, T., Kunz, C., Hiatt, L., and Bugajska, M. (2006). The Human-Robot Interaction Operating System. In *ACM SIGCHI/SIGART Conference on Human-robot interaction*, pages 41–48.

[48] Frank, J. and Jónsson, A. (2003). Constraint-Based Attribute and Interval Planning. *Constraints*, 8(4):339–364.

[49] Fratini, S., Cesta, A., De Benedictis, R., Orlandini, A., and Rasconi, R. (2011). APSI-Based Deliberation in Goal Oriented Autonomous Controllers. In *11th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*.

[50] Frese, U., Hausmann, D., Luth, C., Taubig, H., and Walter, D. (2009). The importance of being formal. *Electronic Notes in Theoretical Computer Science*, 238(4):57–70.

[51] Ghahramani, Z. (1997). Learning Dynamic Bayesian Networks. *Lecture Notes In Computer Science*, 1387:168–197.

[52] Ghallab, M. and Alaoui, A. M. (1989). Managing Efficiently Temporal Relations Through Indexed Spanning Trees. In *International Joint Conference on Artificial intelligence (IJCAI)*, pages 1297–1303, San Francisco, CA, USA.

[53] Ghallab, M. and Laruelle, H. (1994). Representation and Control in IxTeT, a Temporal Planner. In *AI Planning and Scheduling (AIPS)*, pages 61–67.

[54] Ghallab, M., Nau, D. S., and Traverso, P. (2004). *Automated Planning - Theory and Practice*. Elsevier.

[55] Gini, M. L., Ohnishi, K., and Pagello, E. (2010). Advances in Autonomous Robots for Service and Entertainment. *Robotics and Autonomous Systems*, 58(7):829–832.

[56] Giralt, G., Sobek, R., and Chatila, R. (1979). A Multi-Level Planning and Navigation System for a Mobile Robot: A First Approach to HILARE. In *International Joint Conference on Artificial intelligence (IJCAI)*, pages 335–337.

[57] Goodrich, M. and Schultz, A. (2007). Human-Robot Interaction: A Survey. *Foundations and Trends in Human-Computer Interaction*, 1(3):203–275.

[58] Guizzo, E. (2008). Kiva Systems. *IEEE Spectrum*, pages 27–24.

[59] Hägele, M., Nilsson, K., and Pires, J. N. (2008). Industrial Robotics. In [105], pages 963–986.

[60] Harnad, S. (1990). The Symbol Grounding Problem. *Physica D*, pages 335–346.

[61] Harnad, S. (2001). Minds, Machines and Turing: The Indistinguishability of Indistinguishables. *Journal of Logic, Language, and Information (special issue on "Alan Turing and Artificial Intelligence"*.

[62] Henzinger, T. and Sifakis, J. (2006). The Embedded Systems Design Challenge. In *FM: Formal Methods*, Lecture Notes in Computer Science 4085, pages 1–15. Springer.

[63] Hoffmann, J. (2001). FF: The Fast-Forward Planning System. *AI magazine*, 22(3):57.

[64] Hopcroft, J. and Tarjan, R. (1973). Efficient Algorithms for Graph Manipulation. *Communications of the ACM*, 16:372–378.

[65] Ingrand, F., Chatila, R., Alami, R., and Robert, F. (1996). PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 43 –49 vol.1.

[66] Ingrand, F. and Ghallab, M. (2014). Deliberation for autonomous robots: A survey. *Artificial Intelligence*.

[67] Ingrand, F., Lacroix, S., Lemai-Chenevier, S., and Py, F. (2007). Decisional Autonomy of Planetary Rovers. *Journal of Field Robotics*, 24(7):559–580.

[68] Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). On the Convergence of Stochastic Iterative Dynamic Programming Algorithms. *Neural Computation*, 6(6):1185–1201.

[69] Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[70] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research (JAIR)*, 4.

[71] Kaelbling, L. P. and Lozano-Perez, T. (2011). Hierarchical Task and Motion Planning in the Now. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1470–1477.

[72] Kanoun, O., Laumond, J.-P., and Yoshida, E. (2011). Planning Foot Placements for a Humanoid Robot: A Problem of Inverse Kinematics. *International Journal of Robotics Research*, 30(4):476–485.

[73] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 12(4):556–580.

[74] Kazerooni, H. (2008). Exoskeletons for Human Performance Augmentation. In [105], pages 773–793.

[75] Konolige, K., Marder-Eppstein, E., and Marthi, B. (2011). Navigation in Hybrid Metric-Topological Maps. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[76] Kuipers, B. and Byun, Y.-T. (1991). A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations. *Robotics and Autonomous Systems*, 8(1-2):47–63.

[77] Kuipers, B., Modayil, J., Beeson, P., MacMahon, M., and Savelli, F. (2004). Local Metrical and Global Topological Maps in the Hybrid Spatial Semantic Hierarchy. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4845–4851.

[78] Latombe, J.-C., editor (1991). *Robot Motion Planning*. Kluwer, Boston, MA.

[79] Laumond, J.-P. (1990). Connectivity of Plane Triangulation. *Information Processing Letters*, 34(2):87–96.

[80] LaValle, S., editor (2006). *Planning Algorithms*. Cambridge University Press.

[81] Lemai-Chenevier, S. and Ingrand, F. (2004). Interleaving Temporal Planning and Execution in Robotics Domains. In *National Conference on Artificial Intelligence (AAAI)*.

[82] Martínez-Carranza, J. and Calway, A. (2010). Unifying Planar and Point Mapping in Monocular SLAM. In *British Machine Vision Conference (BMVC)*, pages 1–11.

[83] McGann, C., Berger, E., Boren, J., Chitta, S., Gerkey, B., Glaser, S., Marder-Eppstein, E., Marthi, B., Meeussen, W., Pratkanis, T., et al. (2009). Model-based, hierarchical control of a mobile manipulation platform. In *4th Workshop on Planning and Plan Execution for Real World Systems at ICAPS*.

[84] McGann, C., Py, F., Rajan, K., Thomas, H., Henthorn, R., and McEwen, R. (2008). A deliberative architecture for AUV control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1049–1054.

[85] Mei, C. and Rives, P. (2007). Cartographie et Localisation Simultanée avec un Capteur de Vision. In *Journées Nationales de la Recherche en Robotique*.

[86] Milch, B. and Russell, S. J. (2007). First-Order Probabilistic Languages: Into the Unknown. In *Inductive Logic Programming*, volume 4455 of *Lecture Notes in Computer Science*, pages 10–24. Springer.

[87] Minguez, J., Lamiraux, F., and Laumond, J.-P. (2008). Motion Planning and Obstacle Avoidance. In [105], pages 827–852.

[88] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *National Conference on Artificial Intelligence (AAAI)*, pages 593–598.

[89] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2003). FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. In *International Joint Conference on Artificial intelligence (IJCAI)*, pages 1151–1156.

[90] Moravec, H. P. (1983). The Stanford Cart and the CMU Rover. Technical report, CMU.

[91] Morisset, B. and Ghallab, M. (2008). Learning How to Combine Sensory-Motor Functions Into a Robust Behavior. *Artificial Intelligence*, 172(4-5):392–412.

[92] Muscettola, N. (1994). HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M., editors, *Intelligent scheduling*. Morgan Kaufmann.

[93] Newcombe, R. A. and Davison, A. J. (2010). Live dense reconstruction with a single moving camera. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1498–1505.

[94] Nicolescu, M. N. and Mataric, M. J. (2003). Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 241–248.

[95] Peters, J. and Ng, A. Y., editors (2009). *Autonomous Robots, Special issue on robot learning*, volume 27(1-2). Springer.

[96] Pineau, J. and Gordon, G. J. (2005). POMDP Planning for Robust Robot Control. In *International Symposium on Robotics Research (ISRR)*, pages 69–82.

[97] Pineau, J., Gordon, G. J., and Thrun, S. (2003). Policy-contingent abstraction for robust robot control. In *Uncertainty in Artificial Intelligence*, pages 477–484.

[98] Prassler, E. and Kosuge, K. (2008). Domestic Robotics. In [105], pages 1253–1281.

[99] Py, F. and Ingrand, F. (2004). Dependable Execution Control for Autonomous Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1136–1141.

[100] Py, F., Rajan, K., and McGann, C. (2010). A Systematic Agent Framework for Situated Autonomous Systems. In *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 583–590.

[101] Rosen, C. A. and Nilsson, N. J. (1966). Application Of Intelligent Automata to Reconnaissance. Technical report, SRI.

[102] Russell, S. and Norvig, P. (2010). *Artificial Intelligence, A modern approach*. Prencice Hall.

[103] Rybski, P. E., Yoon, K., Stolarz, J., and Veloso, M. M. (2007). Interactive robot task training through dialog and demonstration. In *Conference on Human-Robot Interaction*, pages 49–56.

[104] Schwartz, J., Sharir, M., and Hopcroft, J., editors (1987). *Planning,Geometry and Complexity of Robot Motion*. Ablex Series in Artificial Intelligence. Ablex Publishing.

[105] Siciliano, B. and Khatib, O., editors (2008). *The Handbook of Robotics*. Springer.

[106] Sigaud, O. and Peters, J., editors (2010). *From Motor Learning to Interaction Learning in Robots*, volume 264 of *Studies in Computational Intelligence*. Springer.

[107] Siméon, T., Laumond, J.-P., Cortés, J., and Sahbani, A. (2004). Manipulation Planning with Probabilistic Roadmaps. *International Journal of Robotics Research*, 23(7-8):729–746.

[108] Siméon, T., Laumond, J.-P., and Nissoux, C. (2000). Visibility Based Probabilistic Roadmaps for Motion Planning. *Advanced Robotics Journal*, 14(6).

[109] Simmons, R. and Apfelbaum, D. (1998). A Task Description Language for Robot Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 1931 –1937 vol.3.

[110] Simmons, R., Pecheur, C., and Srinivasan, G. (2000). Towards Automatic Verification of Autonomous Systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1410 –1415 vol.2.

[111] Simmons, R., Singh, S., Heger, F., Hiatt, L., Koterba, S., Melchior, N., and Sellner, B. (2007). Human-Robot Teams for Large-Scale Assembly. In *Proceedings of the NASA Science Technology Conference*. Citeseer.

[112] Simon, D., Espiau, B., Kapellos, K., and Pissard-Gibollet, R. (1997). ORCCAD: Software Engineering for Real-Time Robotics. A Technical Insight. *Robotica*, 15:111–115.

[113] Smith, R., Self, M., and Cheeseman, P. (1986). Estimating uncertain spatial relationships in robotics. In *Proc. Uncertainty in Artificial Intelligence*, pages 435–461.

[114] Stachniss, C. and Burgard, W. (2004). Exploration with active loop-closing for FastSLAM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[115] Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3310 –3317 vol.4.

[116] Stulp, F. and Beetz, M. (2008). Combining Declarative, Procedural and Predictive Knowledge to Generate and Execute Robot Plans Efficiently and Robustly. *Robotics and Autonomous Systems (Special Issue on Semantic Knowledge)*.

[117] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

[118] Tate, A., Drabble, B., and Kirby, R. (1994). *O-Plan2: An Architecture for Command, Planning and Control*. Morgan-Kaufmann.

[119] Taylor, R. H., Menciassi, A., Fichtinger, G., and Dario, P. (2008). Medical Robotics and Computer-Integrated Surgery. In [105], pages 1199–1222.

[120] Teichteil-Königsbuch, F. and Fabiani, P. (2005). Symbolic Heuristic Policy Iteration Algorithms for Structured Decision-Theoretic Exploration Problems. In *Workshop on Planning Under Uncertainty for Autonomous Systems at ICAPS*, pages 66–74.

[121] Teichteil-Königsbuch, F., Kuter, U., and Infantes, G. (2010). Incremental plan aggregation for generating policies in MDPs. In *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1231–1238.

[122] Thrun, S. (2002). Robotic Mapping: A Survey. In Lakemeyer, G. and Nebel, B., editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann.

[123] Thrun, S. (2006). Stanley: The robot that won the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9):661–692.

[124] Tovey, C., Lagoudakis, M., Jain, S., and Koenig, S. (2005). The Generation of Bidding Rules for Auction-Based Robot Coordination. In Parker, L., Schneider, F., and Schultz, A., editors, *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pages 3–14. Springer Netherlands.

[125] Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (2000). Bundle Adjustment – A Modern Synthesis. In Triggs, B., Zisserman, A., and Szeliski, R., editors, *Vision Algorithms: Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 298–372. Springer-Verlag.

[126] Wilkins, D. (1988). *Practical Planning: Extending the classical AI planning paradigm*. Morgan Kaufmann.

[127] Wolfe, J., Marthi, B., and Russell, S. (2010). Combined Task and Motion Planning for Mobile Manipulation. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 5, page 2010, Toronto, Canada.

[128] Wolpert, D. M. and Flanagan, J. R. (2010). Q&A: Robotics as a Tool to Understand the Brain. *Bmc Biology*, 8.

[129] Wolpert, D. M. and Flanagan, J. R. (2016). Computations Underlying Sensorimotor Learning. *Current Opinion in Neurobiology*, 37:7–11.

[130] Wolpert, D. M. and Ghahramani, Z. (2000). Computational Principles of Movement Neuroscience. *Nature Neuroscience*, 3:1212–1217.

[131] Yoshida, K. and Wilcox, B. (2008). Space Robots and Systems. In [105], pages 1031–1063.

[132] Zlot, R. and Stentz, A. (2006). Market-Based Multirobot Coordination Using Task Abstraction. In Yuta, S., Asama, H., Prassler, E., Tsubouchi, T., and Thrun, S., editors, *Field and Service Robotics*, volume 24 of *Springer Tracts in Advanced Robotics*, pages 167–177. Springer Berlin / Heidelberg.