An Overview of Planning Technology in Robotics

Malik Ghallab
LAAS - CNRS, Toulouse, France
Malik.Ghallab@laas.fr

Abstract: We present here an overview of several planning techniques in robotics. We will not be concerned with the synthesis of abstract mission and task plans, using well known classical and other domain-independent planning techniques. We will mainly focus on to how refine such abstract plans into robust sensory-motor actions and on some planning techniques that can be useful for that.

The paper introduces the important and mature area of path and motion planning. It illustrates the usefulness of HTN and MDP planning techniques for the design of a high level controller for a mobile robot.[1]

1 Introduction

A robot integrates several sensory-motor functions, together with communication and information-processing capabilities into cognitive functions, in order to perform a collection of tasks with some level of autonomy and flexibility, in some class of environments. The sensory-motor functions in a robot are, for example:

- locomotion on wheels, legs, or wings,
- manipulation with one or several mechanical arms, grippers and hands,
- localization with odomoters, sonars, laser, inertial and GPS sensors,
- scene analysis and environment modeling with a stereo-vision system on a pan-and-tilt platform.

A robot can be designed for tasks and environments such as:

- manufacturing: painting, welding, loading/unloading a power-press or a machine-tool, assembling parts,
- servicing a store, a warehouse or a factory: maintaining, surveying, or cleaning the area, transporting objects,
- exploring an unknown natural area, e.g., in planetary exploration: building a map with characterized landmarks, extracting samples and setting various measurement devices,
- assisting a person in an office, a public area, or at home,
- performing tele-operated surgical operations, as in the so-called minimal invasive surgery.

Robotics is a reasonably mature technology when, for example

[1] This article is based on a revised material from the Chapter 20 in [17].

- a robot is restricted to operate within a well known and well engineered environments, e.g., as in manufacturing robotics,
- a robot is restricted to perform a single simple task, e.g., vacuum cleaning or lawn mowing.

For more diverse tasks and open-ended environments, robotics remains a very active research field.

A robot may or may not integrate planning capabilities. For example, most of the one million manufacturing robots deployed today in the manufacturing industry do not perform planning *per se*. Using a robot without planning capabilities basically requires handcoding the environment model, and the robot's skills and strategies into a *reactive controller*. This is a perfectly sensible approach as long as this handcoding is inexpensive and reliable enough for the application at hand, which is the case if the environment is well-structured and stable and if the robot's tasks are restricted in scope and diversity, with only a limited man-robot interaction.

Programming aids such as hardware tools, e.g., devices for memorizing the motion of a pantomime, and software systems, e.g., graphical programming interfaces, allow for an easy development of a robot's reactive controller. Learning capabilities, supervised or autonomous, significantly extend the scope of applicability of the approach by allowing a generic controller to adapt to the specifics of its environment. This can be done, for example, by estimating and fine-tuning control parameters and rules, or by acquiring a map of the environment.

However, if a robot has to face a diversity of tasks and/or a variety of environments, then planning will make it simpler to program a robot. It will augment the robot's usefulness and robustness. Planning should not be seen as opposed to the reactive capabilities of a robot, handcoded or learned, neither should it be seen as opposed to its learning capabilities. It should to be closely integrated to them.

The specific requirements of planning in robotics, as compared to other application domains of planning, are mainly the need to handle:

- online input from sensors and communication channels;
- heterogeneous partial models of the environment and of the robot, as well as noisy and partial knowledge of the state from information acquired through sensors and communication channels;
- direct integration of planning with acting, sensing, and learning.

These very demanding requirements advocate for addressing planning in robotics through domain-specific representations and techniques. Indeed, when planning is integrated within a robot, it usually takes several forms and is implemented throughout different systems. Among these various forms of robot planning, there is in particular *path and motion planning*, *perception planning*, *navigation planning*, *manipulation planning*, and domain independent planning.

Today, the maturity of robot planning is mainly at the level of its domain-specific planners. Path and motion planning is a mature area that relies on computational geometry and efficiently uses probabilistic algorithms. It is already deployed in robotics and other

application areas such as CAD or computer animation. Perception planning is a younger and much more open area, although some focused problems are well advanced, e.g. the viewpoint selection problem with mathematical programming techniques.

Domain-independent planning is not widely deployed in robotics for various reasons, among which are the restrictive assumptions and expressiveness of the classical planning framework. In robotics, task planning should ideally deal with time and resource allocation, dynamic environments, uncertainty and partial knowledge, and incremental planning with consistent integration to acting and sensing. The mature planning techniques available today are mostly effective at the abstract level of *mission planning*. Primitives for these plans are tasks such as "navigate to location5", "retrieve and pick-up object2". These tasks are far from being *primitive* sensory-motor functions. Their design is very complex.

Several rule-based or procedure-based systems, such as PRS, RAP, Propice, or SRCs, enable to program manually closed-loop controllers for these tasks that handle the uncertainty and the integration between acting and sensing. These high level reactive controllers permit preprogrammed goal-directed and event-reactive modalities.

However, planning representations and techniques can also be very helpful for the design of high-level reactive controllers performing these tasks. They enable to generate, off-line, several alternative complex plans for achieving the task with robustness. They are useful for finding a policy that chooses, in each state, the best such a plan for pursuing the activity.

The rest of this article presents the important and mature area of path and motion planning (Section 2). It then illustrates the usefulness of planning techniques, for the design of a high level navigation controller for a mobile robot (Section 3). The approach is not limited to navigation tasks. It can be pursued for a wide variety of robotics tasks, such as object manipulation or cleaning. Several sensory-motor functions will be presented and discussed in Section 3.1; an approach that exemplifies the use of planning techniques for synthesizing alternative plans and policies for a navigation task is described. The last section refers to more detailed and focused descriptions of the techniques presented in this overview.

2 Path and Motion Planning

Path planning is the problem of finding a *feasible geometric path* in some environment for moving a mobile system from a starting position to a goal position. A geometric CAD model of the environment with the obstacles and the free space is supposed to be given. A path is feasible if it meets the kinematics constraints of the mobile system and if it avoids collision with obstacles.

Motion planning is the problem of finding a *feasible trajectory*, in space and time, i.e., a feasible path and a control law along that path that meets the dynamics constraints (speed and acceleration) of the mobile system. If one is not requiring an optimal trajectory, it is always possible to *label* temporally a feasible path in order to get a feasible trajectory. Consequently, motion planning relies on path planning, on which we focus the rest of this

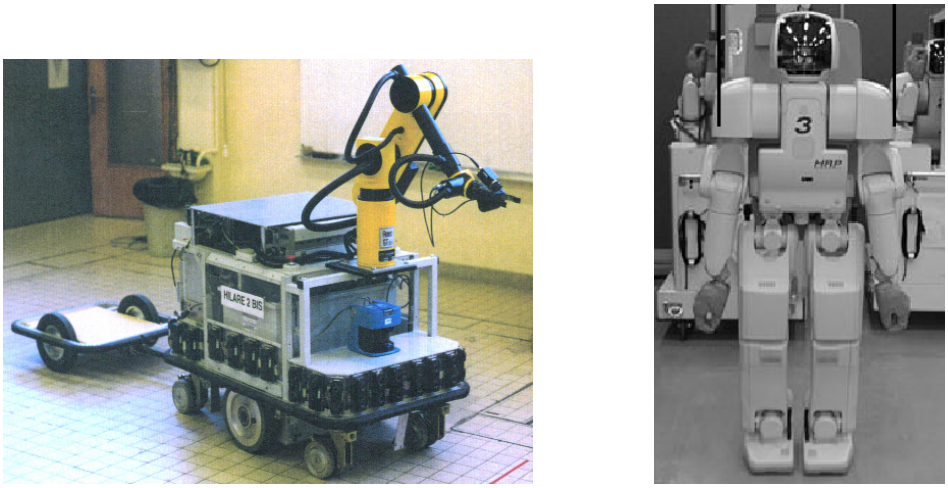

Figure 1: Hilare, a car like robot with an arm and a trailer (left); HRP, a humanoid robot (right).

section.

If the mobile system of interest is a *free-flying* rigid body, i.e., if it can move freely in space in any direction without any kinematics constraint, then six *configuration parameters* are needed to characterize its position: x, y, z and the three Euler angles. Path planning defines a path in this six-dimensional space. However, a robot is not a free-flying body. Its kinematics defines its possible motion. For example, a car-like robot has three configuration parameters, x, y, and θ. Usually these three parameters are not independent, e.g., the robot may or may not be able to turn on the spot (change θ while keeping x and y fixed), or be able to move sideway. A mechanical arm that has n rotational joins needs n configuration parameters to characterize its configuration in space, in addition to constraints such as the maximum and minimum values of each angular join. The car-like robot Hilare in Figure 1 (left) has a total of 10 configuration parameters: 6 for the arm and 4 for the mobile platform with the trailer [34]. The humanoid robot HRP in Figure 1 (right) has 52 configuration parameters: 2 for the head, 7 for each arm, 6 for each leg and 12 for each hand (four finger with 3 configuration parameters each) [26, 27].[2]

Given a robot with n configuration parameters and some environment, let us define:

- q, the *configuration* of the robot: an n-tuple of reals that specifies the n parameters needed to characterize the position in space of the robot,
- CS, the *configuration space* of the robot: the set of values that its configuration q may take,
- CS_{free}, the *free configuration space*: the subset of CS of configurations that are not in collision with the obstacles of the environment.

[2]The *degrees of freedom* of a mobile system are its control variables; an arm or the humanoid robot have as many degrees of freedom as configuration parameters, a car-like robot has 3 configuration parameters but only two degrees of freedom.

Path planning is the problem of finding a path in the free configuration space CS_{free} between an initial and a final configuration. If one could compute CS_{free} explicitly, then path planning would be a search for a path in this n-dimensional continuous space. However, the explicit definition of CS_{free} is a computationally difficult problem, theoretically (it is exponential in the dimension of CS) and practically. Fortunately, very efficient probabilistic techniques have been designed that solve path planning problems even for highly complex robots and environments. They rely on the two following operations:

- *collision checking*, which checks whether a configuration $q \in CS_{free}$, or whether a path between two configurations in CS is collision free, i.e., if it lies entirely in CS_{free},
- *kinematic steering* which finds a path between two configurations q and q' in CS that meets the kinematic constraints, without taking into account obstacles.

Both operations can be performed efficiently. Collision checking relies on computational geometry algorithms and data structures [19]. Kinematic steering may use one of several algorithms, depending on the type of kinematics constraints the robot has. For example, *Manhattan paths* are applied to systems that are required to move only one configuration parameter at a time. Special curves (called *Reed&Shepp* curves [43]) are applied to car-like robots that cannot move sideway. If the robot has no kinematic constraints, then straight line segments in CS from q to q' are used. Several such algorithms can be combined. For example, to plan paths for the robot Hilare in Figure 1 (left), straight line segments for the arm are combined with dedicated curves for the mobile platform with a trailer [34].

Let $\mathcal{L}(q, q')$ be the path in CS computed by the kinematics steering algorithm for the constraints of the robot of interest; $\mathcal{L}$ is assumed to be symmetrical.

Let $\mathcal{R}$ be a graph whose vertices are configurations in CS_{free}; two vertices q and q' are adjacent in $\mathcal{R}$ only if $\mathcal{L}(q, q')$ is in CS_{free}. $\mathcal{R}$ is called a *roadmap* for CS_{free}.

Since $\mathcal{L}$ is symmetrical, $\mathcal{R}$ is an undirected graph. Note that every pair of adjacent vertices in $\mathcal{R}$ is connected by a path in CS_{free} but the converse is not necessarily true. Given a roadmap for CS_{free} and two configuration q_i and q_g in CS_{free}, corresponding to an initial and goal configurations, a feasible path from q_i to q_g can be found as follows:

- find a configuration $q'_i \in \mathcal{R}$ such that $\mathcal{L}(q_i, q'_i) \in CS_{free}$,
- find a configuration $q'_g \in \mathcal{R}$ such that $\mathcal{L}(q_g, q'_g) \in CS_{free}$,
- find in $\mathcal{R}$ a sequence of adjacent configurations from q'_i to q'_g.

If these three steps succeed, then the planned path is the finite sequence of subpaths $\mathcal{L}(q_i, q'_i), \ldots, \mathcal{L}(q'_g, q_g)$. In a post-processing step, this sequence is easily optimized and smoothed locally by finding shortcuts in CS_{free} between successive legs.

Given a roadmap $\mathcal{R}$, path planning is reduced to a simple graph search problem, in addition to collision checking and kinematics steering operations. There remains the problem of finding a roadmap that *covers* CS_{free}, i.e., whenever there is a path in CS_{free} between two configurations, there is also a path through the roadmap. Finding such a roadmap using probabilistic techniques turns out to be easier than computing CS_{free} explicitly.

Let us define the *coverage domain* of a configuration q to be the set:

$$\mathcal{D}(q) = \{q' \in CS_{free} | \mathcal{L}(q, q') \subset CS_{free}\}.$$

A set of configurations Q *covers* CS_{free} if:

$$\bigcup_{q \in Q} \mathcal{D}(q) = CS_{free}.$$

The algorithm Probabilistic-Roadmap (Figure 2) starts initially with an empty roadmap. It generates randomly a configuration $q \in CS_{free}$; q is added to the current roadmap $\mathcal{R}$ iff either:

- q extends the coverage of $\mathcal{R}$, i.e., there is no other configuration in $\mathcal{R}$ whose coverage domain includes q, or
- q extends the connexity of $\mathcal{R}$, i.e., q enables to connect two configurations in $\mathcal{R}$ that are not already connected in $\mathcal{R}$.

```
Probabilistic-Roadmap(R)
    iterate until(termination condition)
        draw a random configuration q in CS_free
        if ∀q' ∈ R: L(q, q') ⊄ CS_free then add q to R
        else if there are q1 and q2 unconnected in R such that
                L(q, q1) ⊂ CS_free and L(q, q2) ⊂ CS_free
            then add q and the edges (q, q1) and (q, q2) to R
    end iteration
    return(R)
end
```

Figure 2: A probabilistic roadmap generation for path planning

Let us assume that there is a finite set Q that covers CS_{free}.[3] Consider the roadmap $\mathcal{R}$ that contains all the configurations in Q, and, for every pair q_1 and q_2 in Q such that $\mathcal{D}(q_1)$ and $\mathcal{D}(q_2)$ intersect, $\mathcal{R}$ also contains a configuration $q \in \mathcal{D}(q_1) \cap \mathcal{D}(q_1)$ and the two edges (q, q_1) and (q, q_2). It is possible to show that $\mathcal{R}$ meets the following property: if there exists a feasible path between two configurations q_i and q_g in CS_{free}, then there are two configurations q_i' and q_g' in the roadmap $\mathcal{R}$ such that $q_i \in \mathcal{D}(q_i')$, $q_g \in \mathcal{D}(q_g')$, and q_i' and q_g' are in the same connected component of $\mathcal{R}$. Note that the roadmap may have several connected components that reflects those of CS_{free}.

The Probabilistic-Roadmap algorithm will not generate a roadmap that meets the above property *deterministically*, but only up to some probability value, which is linked to the termination condition. Let k be the number of random draws since the last draw of a configuration q that has been added to the roadmap because q extends the coverage of the current $\mathcal{R}$ (q meets the first if clause in Figure 2). The termination condition is to stop when k reaches a preset value k_{max}. It has been shown that $1/k_{max}$ is a probabilistic estimate of the ratio between the part of CS_{free} not covered by $\mathcal{R}$ to the total CS_{free}. In other words, for $k_{max} = 1000$ the algorithm generates a roadmap that covers CS_{free} with a probability of .999.

[3] Depending on the shape of CS_{free} and the kinematics constraints handled in $\mathcal{L}$, there may or may not exist such a *finite* set of configurations that covers CS_{free} [29].

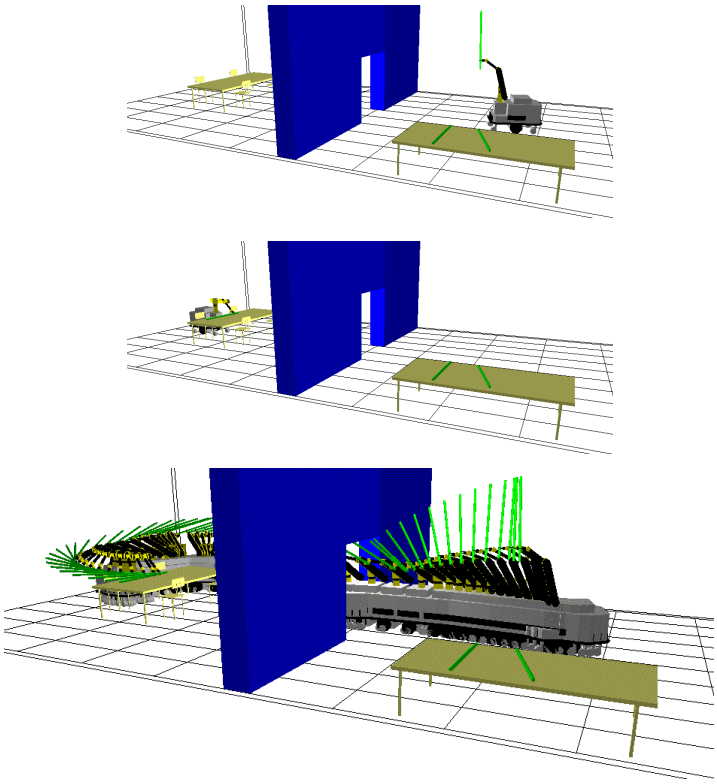

Figure 3: Initial and goal configurations (up left and right) of a path planning problem, and generated path (down).

From a practical point of view, the probabilistic roadmap technique illustrated by the previous algorithm has led to some very efficient implementations and to marketed products used in robotics, computer animation, CAD and manufacturing applications. Typically, for a complex robot and environment, and k_{max} in the order of few hundreds, it takes about a minute to generate a roadmap on a normal desktop machine; the size of $\mathcal{R}$ is about a hundred configurations; path planning with the roadmap takes few milliseconds. This is illustrated for the Hilare robot in Figure 3 where the task is to carry a long rod that constrains the path through the door: the roadmap in this 9-dimensional space has about 100 vertices and is generated in less than one minute. The same techniques have also been successfully applied to manipulation planning problems.

3 Planning for the Design of a Robust Controller

Consider an autonomous mobile robot in a structured environment, such as the robot in Figure 1 (left), which is equipped with several sensors — sonar, laser, vision — and actuators, and with an arm. The robot has also several software modules for the same sensory-motor (*sm*) function, e.g., for localization, for map building and updating, or for motion planning and control. These redundant *sm* functions are needed because of possible failures of a sensor, and because no single method or sensor has a universal coverage. Each has its weak points and drawbacks. Robustness requires a diversity of means for achieving an *sm* function. Robustness also requires the capability to combine consistently several such *sm* functions into a plan appropriate for the current context.

The planning techniques described in this section illustrates this capability. They enables a designer to specify, off-line, very robust ways of performing a task such as "navigate to". The designer specifies a collection of Hierarchical Tasks Networks, as illustrated in Figure 4, that are complex plans, called *modes of behavior*, or *modalities* for short,[4] whose primitives are *sm* functions. Each modality is a possible way of combining a few of these *sm* functions to achieve the desired task. A modality has a rich context-dependent control structure. It includes alternatives whose selection depends on the data provided by *sm* functions.

Several modalities are available for a given task. The choice of the right modality for pursuing a task is far from being obvious. However, the relationship between control states and modalities can be expressed as a Markov Decision Process. This MDP characterizes the robot abilities for that task. The probability and cost distributions of this MDP are estimated by moving the robot in the environment. The controller is driven by policies extracted on-line from this MDP.

To summarize, this approach involves three components:

- Sensory-motor functions, which are the primitive actions.
- Modalities that are HTN plans. Alternate modalities offer different ways of combining the *sm* functions within a task,
- MDPs whose policies are used by the controller to achieve the task.

Let us describe these three levels successively.

3.1 Sensory-Motor Functions

The sensory-motor functions illustrated here and the control system itself rely on a model of the environment learned and maintained by the robot. The basic model is a 2D map of obstacle edges acquired from the laser range data. The so-called Simultaneous Localization and Mapping (SLAM) technique is used to generate and maintain the map of the environment.

A labeled topological graph of the environment is associated with the 2D map. Cells

[4] *Behaviors* have generally in robotics a meaning different from our modalities.

are polygons that partition the metric map. Each cell is characterized by its name and a *color* that corresponds to navigation features such as Corridor, Corridor with landmarks, Large Door, Narrow Door, Confined Area, Open Area, Open Area with fixed localization devices.[5] Edges of the topological graph are labeled by estimates of the transition length from one cell to the next and by heuristic estimates of how easy such a transition is.

An *sm* function returns to the controller a report indicating either the end of a normal execution, or giving additional information about non-nominal execution. In order to give to the reader an idea of the "low level"primitives available on a robot, of their strong and weak points and how they can be used from a planning point of view, let us discuss some of these *sm* functions.

Segment-based localization. This function relies on the map maintained by the robot from laser range data. The SLAM technique uses a data estimation approach called Extended Kalman Filtering in order to match the local perception with the previously built model. It offers a continuous position-updating mode, used when a good probabilistic estimate of the robot position is available. This *sm* function estimates the inaccuracy of the robot localization. When the robot is lost, a re-localization mode can be performed. A constraint relaxation on the position inaccuracy extends the search space until a good matching with the map is found.

This *sm* function is generally reliable and robust to partial occlusions, and much more precise than odometry. However, occlusion of the laser beam by obstacles gives unreliable data. This case occurs when dense unexpected obstacles are gathered in front of the robot. Moreover, in long corridors the laser obtains no data along the corridor axis. The inaccuracy increases along the corridor axis. Restarting the position updating loop in a long corridor can prove to be difficult. A feedback from this *sm* function can be a report of bad localization which warns that the inaccuracy of the robot position has exceeded an allowed threshold. The robot stops, turns on the spot and re-activates the re-localization mode. This can be repeated in order to find a non-ambiguous corner in the environment to restart the localization loop.

Localization on Visual Landmarks. This function relies on a calibrated monocular vision to detect known landmarks such as doors or wall posters. It derives from the perceptual data a very accurate estimation of the robot position. The setting up is simple: a few wall posters and characteristic planar features on walls are learned in supervised mode. However, landmarks are available and visible only in a few areas of the environment. Hence this *sm* function is mainly used to update from time to time the last known robot position. A feedback from this *sm* function is a report of a potentially visible landmark which indicates that the robot enters an area of visibility of a landmark. The robot stops, turns towards the expected landmark; it searches it using the pan-tilt mount. A failure report notifies that the landmark was not identified. Eventually, the robot retries from a second predefined position in the landmark visibility area.

[5] Some environment modeling techniques that enable to automatically acquire such a topological graph with the cells and their labels exist. They are discussed in Section 4. However, in the work referred to here, the topological graph is hand-programmed.

Absolute Localization. The environment may have areas equipped with calibrated fixed devices, such as infrared reflectors, cameras, or even areas where a differential GPS signal is available. These devices permit a very accurate and robust localization. But the *sm* function works only when the robot is within a covered area.

Elastic Band for Plan Execution. This *sm* function updates and maintains dynamically a flexible trajectory as an *elastic band* or a sequence of configurations from the current robot position to the goal. Connexity between configurations relies on a set of internal forces that are used to optimize the global shape of the path. External forces are associated with obstacles and are applied to all configurations in the band in order to dynamically modify the path to take it away from obstacles. This *sm* function takes into account the planed path, the map and the on-line input from the laser data. It gives a robust method for long range navigation. However, the band deformation is a local optimization between internal and external forces; the techniques may fail into local minima. This is the case when a mobile obstacle blocks the band against another obstacle. Furthermore, it is a costly process which may limit the reactivity in certain cluttered, dynamic environments. This also limits the band length.

The feedback may warn that the band execution is blocked by a temporary obstacle that cannot be avoided (e.g. a closed door, an obstacle in a corridor). This obstacle is perceived by the laser and is not represented in the map. If the band relies on a planed path, the new obstacle is added to the map. A new trajectory taking into account the unexpected obstacle is computed, and a new elastic band is executed. Another report may warn that the actual band is no longer adapted to the planed path. In this case, a new band has to be created.

Reactive Obstacle Avoidance. This *sm* function provides a reactive motion capability towards a goal without needing a planned path. It extracts from sensory data a description of free regions. It selects the closest region to the goal, taking into account the distance to the obstacles. it computes and tries to achieve a motion command to that region.

This *sm* function offers a reactive motion capability that remains efficient in a cluttered space. However, like all the reactive methods, it may fall into local minima. It is not appropriate for long range navigation. Its feedback is a failure report generated when the reactive execution is blocked.

Finally, let us mention that a path planner (as described in Section 2) may also be seen as a *sm* function from the viewpoint of a high-level navigation controller. Note that a planned path doesn't take into account environment changes and new obstacles. Furthermore, a path planner may not succeed in finding a path. This may happen when the initial or goal configurations are too close to obstacles: because of the inaccuracy of the robot position, these configuration are detected as being outside of CS_{free}. The robot has to move away from the obstacles by using a reactive motion *sm* function before a new path is queried.

3.2 Modalities

A navigation task such as `(Goto x y` θ`)` given by a mission planning step requires an integrated use of several *sm* functions among those presented earlier. Each consistent combination of these *sm* functions is a particular plan called a *modality*. A navigation modality is a one way of performing the navigation task. A modality has specific characteristics that make it more appropriate for some contexts or environments, and less for others. We will discuss later how the controller choses the appropriate modality. Let us exemplify some of such modalities for the navigation task before giving the detail of the HTN representation for modalities and the associated control system.

Modality M_1 uses 3 *sm* functions: the path planner, the elastic band for the dynamic motion execution, and the laser-based localization. When M_1 is chosen to carry out a navigation, the laser-based localization is initialized. The robot position is maintained dynamically. A path is computed to reach the goal position. The path is carried out by the elastic band *sm* function. Stopping the modality interrupts the band execution and the localization loop; it restores the initial state of the map if temporary obstacles have been added to it. Suspending the modality stops the band execution. The path, the band, the localization loop are maintained. A suspended modality can be resumed by restarting the execution of the current elastic band.

Modality M_2 uses 3 *sm* functions: the path planner, the reactive obstacle avoidance and the laser-based localization. The path planner provides way-points (vertices of the trajectory) to the reactive motion function. Despite these way-points the reactive motion can be trapped into local minima in cluttered environments. Its avoidance capability is higher than that of the elastic band *sm* function. However, the reactivity to obstacles and the attraction to way-points may lead to oscillations and to a discontinuous motion that confuses the localization *sm* function. This is a clear drawback for M_2 in long corridors.

Modality M_3 is like M_2 but without path planning and with a reduced speed in obstacle avoidance. It starts with the reactive motion and the laser-based localization loop. It offers an efficient alternative in narrow environments like offices, and in cluttered spaces where path planning may fail. It can be preferred to the modality M_1 in order to avoid unreliable re-planning steps if the elastic band is blocked by a cluttered environment. Navigation is only reactive, hence with a local minima problem. The weakness of the laser localization in long corridors is also a drawback for M_3.

Modality M_4 uses the reactive obstacle avoidance *sm* function with the odometer and the visual landmark localization *sm* functions. The odometer inaccuracy can be locally reset by the visual localization *sm* function when the robot goes by a known landmark. Reactive navigation between landmarks allows to cross a corridor without an accurate knowledge of the robot position. Typically this M_2 modality can be used in long corridors. The growing inaccuracy can make it difficult to find out the next landmark. The search method allows for some inaccuracy on the robot position by moving the cameras but this inaccuracy cannot exceed one meter. For this reason landmarks should not to be too far apart with respect to the required updating of odometry estimate. Furthermore, the reactive navigation of M_2 may fall into a local minima.

Modality M_5 relies on the reactive obstacle avoidance *sm* function and the absolute localization *sm* function when the robot is within an area equipped with absolute localization devices.

Modalities are represented as Hierarchical Task Networks. The HTN formalism is adapted to modalities because of its expressiveness and its flexible control structure. HTNs offer a middle ground between programming and automated planning, allowing the designer to express the control knowledge which is available here.

An internal node of the HTN And/Or tree is a task or a subtask that can be pursued in different context-dependent ways, which are the *Or-connectors*. Each such Or-connector is a possible decomposition of the task into a conjunction of subtasks. There are two types of *AND-connectors*: with sequential or with parallel branches. Branches linked by a sequential AND-connector are traversed sequentially in the usual depth-first manner. Branches linked by a parallel AND-connector are traversed in parallel. The leaves of the tree are primitive actions, each corresponding to a unique query to a *sm* function. Thus, a root task is dynamically decomposed, according to the context, into a set of primitive actions organized as concurrent or sequential subsets. Execution starts as soon as the decomposition process reaches a leaf, even if the entire decomposition process of the tree is not complete.

A primitive action can be *blocking* or *non-blocking*. In blocking mode, the control flow waits until the end of this action is reported before starting the next action in the sequence flow. In non-blocking mode, actions in a sequence are triggered sequentially without waiting for a feedback. A blocking primitive action is considered ended after a report has been issued by the *sm* function and after that report has been processed by the control system. The report from a non-blocking primitive action may occur and be processed after an unpredictable delay.

The modality tree illustrated in Figure 4 starts with 6 Or-connectors labeled `start`, `stop`, `suspend`, `resume`, `succeed` and `fail`. The `start` connector represents the nominal modality execution; the `stop` connector the way to stop the modality and to restore the neutral state, characterized by the lack of any *sm* function execution. Furthermore, the environment model modified by the modality execution recovers its previous form. The `suspend` and `resume` connectors are triggered by the control system described below. The `suspend` connector allows to stop the execution by freezing the state of the active *sm* functions. The `resume` connector restarts the modality execution from such a frozen state. The `fail` (resp. `succeed`) connector is followed when the modality execution reaches a failure (resp. a success) end. These connectors are used to restore the neutral state and to allow certain executions required in these specific cases.

The feedback from *sm* functions to modalities has to be controlled as well as the resource sharing of parallel activities. The control system catches and reacts appropriately to reports emitted by *sm* functions. Reports from *sm* functions play the same role in the control system as tasks in modalities. A report of some type activates its own dedicated control HTN in a reactive way. A control tree represents a temporary modality and cannot be interrupted. A nominal report signal a normal execution. Otherwise a non-nominal report signals a particular type of *sm* function execution. The aim of the corresponding control

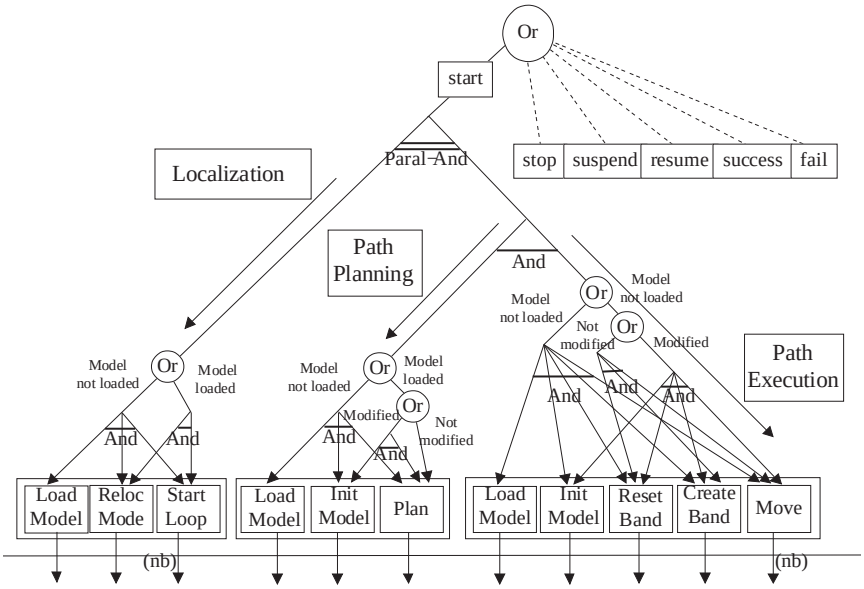

Figure 4: Part of modality M_1

tree is to recover to a nominal modality execution. Some non-nominal reports can be non recoverable failures. In these cases, the corresponding control sends a ”fail”message to the modality pursuing this *sm* function. Nominal reports may notify the success of the global task. In this case, the ”success”alternative of the modality is activated.

Resources to be managed are either physical non-sharable resources (e.g. motors, cameras, pan-and-tilt mount) or logical resources (the environment model that can be temporally modified). The execution of a set of concurrent non-blocking actions can imply the simultaneous execution of different *sm* functions. Because of that, several reports may appear at the same time, and induce the simultaneous activation of several control activities. These concurrent executions may generate a resource conflict. To manage this conflict, a resource manager organizes the resource sharing with semaphores and priorities.

When a non-nominal report is issued, a control HTN starts its execution. It requests the resource it needs. If this resource is already in use by a start connector of a modality, the manager sends to this modality a suspend message, and leaves a resume message for the modality in the spooler according to its priority. The suspend alternative is executed freeing the resource, enabling the control HTN to be executed. If the control execution succeeds, waiting messages are removed and executed until the spooler becomes empty. If the control execution fails, the resume message is removed from the spooler and the fail alternative is executed for the modality.

3.3 The Controller

The Control Space. The controller has to choose a modality that is most appropriate to the current state for pursuing the task. In order to do this, a set of *control variables* have to reflect control information for the *sm* functions. The choice of these control variables is an important design issue. For example, in the navigation task, the control variables:

- The cluttering of the environment which is defined to be a weighted sum of the distances to nearest obstacles perceived by the laser, with a dominant weight along the robot motion axis. This is an important piece of information to establish the execution conditions of the motion and localization *sm* functions.
- The angular variation of the profile of the laser range data which characterizes the robot area. Close to a wall, the cluttering value is high but the angular variation remains low. But in an open area the cluttering is low while the angular variation may be high.
- The inaccuracy of the position estimate, as computed from the co-variance matrix maintained by each localization *sm* function.
- The confidence in the position estimate. The inaccuracy is not sufficient to qualify the localization. Each localization *sm* function supplies a confidence estimate about the last processed position.
- The navigation *color* of current area. When the robot position estimate falls within some labeled cell of the topological graph, the corresponding labels are taken into account,e.g., *Corridor, Corridor with landmarks, Large door, Narrow door, Confined area, Open area, Area with fixed localization*.
- The current modality. This information is essential to assess the control state and possible transitions between modalities.

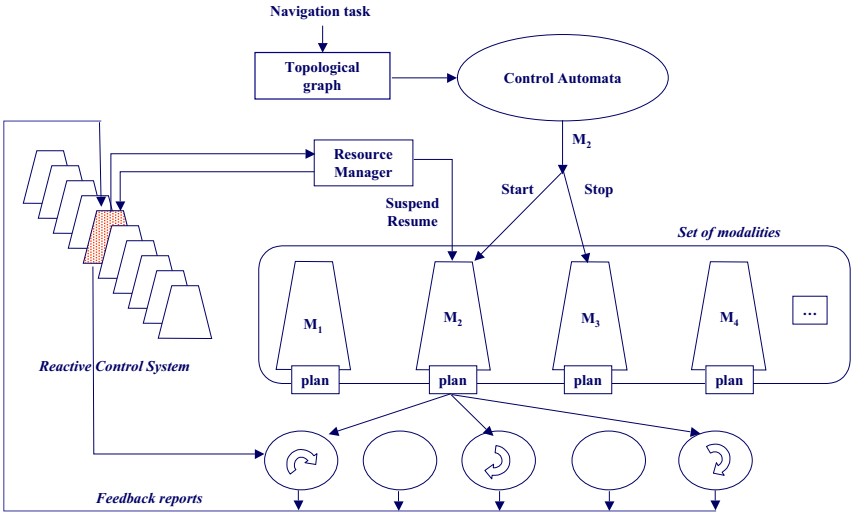

Figure 5: The ROBEL control system

A control state is characterized by the values of these control variables. Continuous variables are discretized over a few significant intervals. In addition, there is a global failure state that is reached whenever the control of a modality reports a failure. We finally end-up with a discrete control space which enables to define a *control automaton.*

The Control Automaton. The control automaton is nondeterministic: unpredictable external events may modify the environment, e.g. someone passing by may change the value of the cluttering variable, or the localization inaccuracy variable. Therefore the execution of the same modality in a given state may lead to different adjacent states. This nondeterministic control automaton is defined as the tuple $\Sigma = \{S, A, P, C\}$:

- S is a finite set of control states,
- A is a finite set of modalities,
- $P : S \times A \times S \rightarrow [0, 1]$ is a probability distribution on the state-transition *sm* function, $P_a(s'|s)$ is the probability that the execution of modality a in state s leads to state s',
- $C : A \times S \times S \rightarrow \Re^+$ is a positive cost function, $c(a, s, s')$ corresponds to the average cost of performing the state transition from s to s' with to the modality a.

A and S are given by design from the definition of the set of modalities and of the control variables. In the navigation system illustrated here, there are 5 modalities and about a few thousand states. P and C are obtained from observed statistics during a learning phase.

The Control automaton Σ is a Markov Decision Process. As an MDP, Σ could be used reactively on the basis of a universal policy π which selects for a given state s the best modality $\pi(s)$ to be executed. However, a universal policy will not take into account the current navigation goal. A more precise approach takes into account explicitly the navigation goal, transposed into Σ as a set S_g of goal states in the control space. This set S_g is given by a look-ahead mechanism based on a search for a path in Σ that reflects a topological route to the navigation goal (see Figure 5).

Goal States in the Control Space. Given a navigation task, a search in the topological graph provides an optimal route r to the goal, taking into account estimated cost of edges between topological cells. This route will help finding in the control automaton possible goal control states for planning a policy. The route r is characterized by the pair (σ_r, l_r), where $\sigma_r = \langle c_1 c_2 \dots c_k \rangle$ is the sequence of colors of traversed cells, and l_r is the length of r.

Now, a path between two states in Σ defines also a sequence of colors σ_{path}, those of traversed states; it has a total cost, that is the sum $\sum_{path} C(a, s, s')$ over all traversed arcs. A path in Σ from the current control state s_0 to a state s corresponds to the planned route when the path *matches* the features of the route (σ_r, l_r) in the following way:

- $\sum_{path} c(a, s, s') \geq K l_r$, K being a constant ratio between the cost of a state-transition in the control automaton to corresponding route length,
- σ_{path} corresponds to the same sequence of colors as σ_r with possible repetition factors, i.e., there are factors $i_1 > 0, \dots, i_k > 0$ such that $\sigma_{path} = \langle c_1^{i_1}, c_2^{i_2}, \dots, c_k^{i_k} \rangle$ when $\sigma_r = \langle c_1, c_2, \dots, c_k \rangle$.

This last condition requires that we will be traversing in Σ control states having the same color as the planned route. A repetition factor corresponds to the number of control states, at least one, required for traversing a topological cell. The first condition enables to prune paths in Σ that meet the condition on the sequence of colors but cannot correspond to the planned route. However, paths in Σ that contain a loop (i.e. involving a repeated control sequence) necessarily meet the first condition.

Let route(s_0, s) be true whenever the optimal path in Σ from s_0 to s meets the two previous conditions, and let $S_g = \{s \in S \mid \text{route}(s_0, s)\}$. A Moore-Dijkstra algorithm starting from s_0 gives optimal paths to all states in Σ in $O(n^2)$. For every such a path, the predicate route(s_0, s) is checked in a straightforward way, which gives S_g.

It is important to notice that this set S_g of control states is a *heuristic projection* of the planned route to the goal. There is no guaranty that following blindly (i.e., in an open-loop control) a path in Σ that meets route(s_0, s) will lead to the goal; and there is no guarantee that every successful navigation to the goal corresponds to a sequence of control states that meets route(s_0, s). This only an efficient and reliable way of focusing the MDP cost function with respect to the navigation goal and to the planned route.

Finding a Control Policy. At this point we have to find the best modality to apply to the current state s_0 in order to reach a state in S_g, given the probability distribution function P and the cost function C.

A simple adaptation of the *Value Iteration* algorithm solves this problem. Here we only need to know $\pi(s_0)$. Hence the algorithm can be focused on a subset of states, basically those explored by the Moore-Dijkstra algorithm.

The closed-loop controller uses this policy as follows:

- the computed modality $\pi(s_0)$ is executed;
- the robot observes the new control state s, it updates its route r and its set S_g of goal states with respect to s, it finds the new modality to apply to s.

This is repeated until the control reports a success or a failure. Recovery from a failure state consists in trying from the parent state an untried modality. If none is available, a global failure of the task is reported.

Estimating the Parameters of the Control automaton. A sequence of randomly generated navigation goals can be given to the robot. During its motion, new control states are met and new transitions are recorded or updated. Each time a transition from s to s' with modality a is performed, the traversed distance and speed are recorded, and the average speed v of this transition is updated. The cost of the transition $C(a, s, s')$ can be defined as a weighted average of the traversal time for this transition taking into account the eventual control steps required during the execution of the modality a in s together with the outcome of that control. The statistics on $a(s)$ are recorded to update the probability distribution function.

Several strategies can be defined to learn P and C in Σ. For example:

- A modality is chosen randomly for a given task; this modality is pursued until either it succeeds or a fatal failure is notified. In this case, a new modality is chosen randomly and is executed according to the same principle. This strategy is used initially to expand Σ.
- Σ is used according to the normal control except in a state on which not enough data has been recorded; a modality is randomly applied to this state in order to augment known statistics, e.g, the random choice of an untried modality in that state.

3.4 Analysis of the Approach

The system described here has been deployed on the Diligent robot, an indoor mobile platform, and extensively experimented with in navigation tasks within a wide laboratory environment [38, 39]. The approach is fairly generic and illustrates the use of planning techniques in robotics, not for the synthesis of mission plans but for achieving a robust execution of their high-level steps. It is not limited to navigation; it can be deployed on other robot activities.

The HTN planning technique used for specifying detailed alternative plans to be followed by a controller for decomposing a complex task into primitive actions is fairly general and powerful. It can be widely applied in robotics since it enables to take into account closed-loop feedback from sensors and primitive actions. It extends significantly and can rely on the capabilities of the rule-based or procedure-based languages for programming reactive controllers, as in the system described here.

The MDP planning technique relies on an abstract dedicated space, namely the space of control states for the navigation task. The size of such a space is just a few thousand states. Consequently, the estimation of the parameter distributions in Σ is feasible in a reasonable time: the MDP algorithms can be used efficiently on-line, at each control step. The drawback of these advantages is the *ad hoc* definition of the control space which requires a very good knowledge of the sensory-motor functions and the navigation task. While in principle the system described here can be extended by the addition of new modalities for the same task, or for other tasks, it is not clear how easy it would be to update the control space or to define new spaces for other tasks.

4 Discussion

Robot motion planning is a very advanced research field [35, 28]. The early techniques in the eighties have been mostly dedicated to deterministic algorithms [36]. They led to a good understanding and formalization of the problem, as well as to several developments on related topics such as manipulation planning [2]. More recent approaches have built on this state of the art with probabilistic algorithms that permitted a significant scale up [4]. The probabilistic roadmap techniques introduced in [30] gave rise to several successful developments [9, 29, 20, 24, 10, 32, 46] which represent today the most efficient approaches

to path planning. Roadmap techniques are certainly not limited to navigation tasks; they have been deployed in other application areas, within robotics, e.g., for manipulation, or in CAD and graphics animation. The illustrations and performance figures in Section 2 are borrowed from Move3D, a state of the art system implementing roadmap techniques [45].

Sensory-motor functions are at the main core of robotics. They correspond to a very wide research area, ranging from signal processing, computer vision and learning, to biomechanics and neuroscience. Approaches relevant to the *sm* functions presented here are, for example,

- the techniques used for localization and mapping, e.g., the SLAM methods [40, 49, 14, 50],
- the methods for structuring the environment model into a topological map with areas labeled by different navigation colors [33, 48],
- the visual localization techniques, e.g., [22], and
- the flexible control techniques, e.g., [42, 44].

Several high-level reactive controllers are widely deployed in laboratory robots. They permit a preprogrammed goal-directed and event-reactive closed-loop control, integrating acting and sensing. They rely on rule-based or procedure-based systems, such as PRS, RAP, SRC and others [16, 25, 11, 15]. More recent developments on these systems, e.g., [13], aim at a closer integration to planning. The behavior-based controllers, e.g., [3], that usually focus on a more reactive set of concurrent activities, have also led to more goal-directed developments, e.g., [23].The *robot architecture*, that is the organization that enables to properly integrate the sensory-motoric functions, the reactive control system and the deliberative capabilities [1, 47] remains important issue.

The planning and robotics literature reports on several plan-based robot controllers with objectives similar to those discussed here, such as for example [5, 7, 6, 31, 8]. The approach of Beetz [6] has also been deployed for controlling an indoor robot carrying out the cores of an office courier. It relies on the SRCs reactive controllers. These are concurrent control routines that adapt to changing conditions by reasoning on and modifying plans. They rely on the XFRM system that manipulates reactive plans and is able to acquire them through learning with XFRMLEARN [7].

In addition to plan-based controllers, there is an active area of research that aims at interleaving task planning activities together with execution control and monitoring activities. Several approaches have been developed and applied, for example, to space and military applications, e.g., within the SIPE [41] or the CASPER [12] systems. Applications in robotics are for example the ROGUE system [21], and more recently the IxTeT-eXeC system [37] that integrates a sophisticated time and resource handling mechanism for planning and controlling the mission of an exploration robot.

References

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *International Journal of Robotics Research*, 17(4), 1998.

[2] R. Alami, J. P. Laumond, and T. Siméon. *Two Manipulation Planning Algorithms*, pages 109–125. In Goldberg et al. [18], 1995.

[3] R. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[4] J. Barraquand and J. C. Latombe. Robot motion planning: a distributed representation approach. *International Journal of Robotics Research*, 10(6), 1991.

[5] M. Beetz. Structured reactive controllers - a computational model of everyday activity. In *3rd Int. Conf. on Autonomous Agents*, pages 228–235, 1999.

[6] M. Beetz. *Plan-based control of robotics agents*, volume 2554 of *Lecture Notes in Artificial Intelligence (LNAI)*. Springer, 2002.

[7] M. Beetz and T. Belker. Environment and task adaptation for robotics agents. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 2000.

[8] M. Beetz, J. Hertzberg, M. Ghallab, and M.E. Pollack, editors. *Advances in plan-based control of robotics agents*. LNAI 2466, Springer-Verlag, 2002.

[9] P. Bessiere, J. Ahuactzin, E. Talbi, and E. Mazer. The ariadne's clew algorithm: global planning with local methods. In Goldberg et al. [18], pages 39–47.

[10] V. Boor, M. H. Overmars, and A. F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1999.

[11] John L. Bresina. Design of a reactive system based on classical planning. In *Foundations of Automatic Planning: The Classical Approach and Beyond: Papers from the 1993 AAAI Spring Symposium*, pages 5–9. AAAI Press, Menlo Park, California, 1993.

[12] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proceedings of the International Conference on AI Planning Systems (AIPS)*, 2000.

[13] O. Despouys and F. Ingrand. PropicePlan: Toward a Unified Framework for Planning and Execution. In *Proceedings of the European Conference on Planning (ECP)*, pages 280–292, 1999.

[14] M. W. M.G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.

[15] R. J. Firby. Task networks for controlling continuous processes. In *Proceedings of the International Conference on AI Planning Systems (AIPS)*, 1994.

[16] M.P. Georgeff and F.F. Ingrand. Decision-Making in an Embedded Reasoning System. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1989.

[17] M. Ghallab, D. Nau, and P. Traverson. *Automated planning, theory and practice*. Elsevier, Morgan Kauffman, 2004.

[18] K. Goldberg, D. Halperin, J. C. Latombe, and R. Wilson, editors. *Algorithmic Foundations of Robotics*. A K Peters, 1995.

[19] J. Goodman and J. ORourke. *Handbook of discrete and computational geometry*. CRC Press, 1997.

[20] K. Gupta and A. del Pobil, editors. *Practical motion planning in robotics*. Wiley, 1998.

[21] K. Z. Haigh and M. M. Veloso. Planning, execution and learning in a robotic agent. In *Proceedings of the International Conference on AI Planning Systems (AIPS)*, 1998.

[22] J.B. Hayet, F. Lerasle, and M. Devy. Planar landmarks to localize a mobile robot. In *SIRS'2000*, pages 163–169, 2000.

[23] J. Hertzberg, H. Jaeger, U. Zimmer, and Ph. Morignot. A framework for plan execution in behavior-based robots. In *Proc. of the 1998 IEEE Int. Symp. on Intell. Control*, pages 8–13, 1998.

[24] D. Hsu, L. Kavraki, JC. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In P. Agarwal et al., editor, *Robotics: The Algorithmic Perspective (WAFR98)*, 1998.

[25] F.F. Ingrand and M.P. Georgeff. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert*, 6:33–44, 1992.

[26] H. Inoue, S. Tachi, Y. Nakamura, K. Hirai, N. Ohyu, S. Hirai, K. Tanie, K. Yokoi, and H. Hirukawa. Overview of humanoid robotics project of METI. In *32nd International Symposium on Robotics*, 2001.

[27] F. Kanehiro, M. Inaba, H. Inoue, H. Hirukawa, and S. Hirai. Developmental software environment that is applicable to small-size humanoids and life-size humanoids. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[28] L. Kavraki. Algorithms in robotics: The motion planning perspective. In *Frontiers of Engineering Publication*, pages 90–93. National Academy of Engineering, 1999.

[29] L. Kavraki, M. Kolountzakis, and J.C. Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14(1):166–171, 1998.

[30] L. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[31] D. Kortenkamp, R.P. Bonasso, and R.R. Murphy, editors. *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, 1997.

[32] J. Kuffner and S. Lavalle. RRT-connect: an efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[33] S. Lacroix and R. Chatila. Motion and perception strategies for outdoor mobile robot navigation in unknown environments. In O. Khatib and J. K. Salisbury, editors, *International Symposium on Experimental Robotics*, pages 538–547. LNCIS 223, Springer-Verlag, 1997.

[34] F. Lamiraux, S. Sekhavat, and J.P. Laumond. Motion planning and control for hilare pulling a trailer. *IEEE Transactions on Robotics and Automation*, 15(4), 1999.

[35] J. C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, 18(11):1119–1128, 1999.

[36] J.C. Latombe. *Robot Motiton Planning*. Kluwer Academic Publishers, 1991.

[37] S. Lemai and F. Ingrand. Interleaving temporal planning and execution in robotics domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2004.

[38] B. Morisset and M. Ghallab. *Learning how to combine sensory-motor modalities for a robust behavior*, pages 157–178. In Beetz et al. [8], 2002.

[39] B. Morisset and M. Ghallab. Synthesis of supervision policies for robust sensory-motor behaviors. In *7th International Conference on Intelligent Autonomous Systems*, pages 236–243, 2002.

[40] P. Moutarlier and R. G. Chatila. Stochastic Multisensory Data Fusion for Mobile Robot Location and Environment Modelling. In *Proc. International Symposium on Robotics Research*, 1989.

[41] K. L. Myers. A Continuous Planning and Execution Framework. *AI Magazine*, pages 63–69, 1999.

[42] S. Quinlan and O. Khatib. Towards real-time execution of motion tasks. In R. G. Chatila and G. Hirzinger, editors, *Experimental Robotics 2*. Springer-Verlag, 1992.

[43] J. A. Reed and R. A. Shepp. Optimal paths for a car that goes both forward and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990.

[44] A. Saffiotti. Handling uncertainty in control of autonomous robots. In Wooldridge and Veloso, editors, *Artificial Intelligence Today*, pages 381–408. LNAI1600, Springer-Verlag, 1999.

[45] T. Siméon, J.P. Laumond, and F. Lamiraux. Move3d: a generic platform for path planning. In *4th International Symposium on Assembly and Task Planning*, 2001.

[46] T. Siméon, J.P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6), 2000.

[47] R. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, 1994.

[48] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[49] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Frölinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt. *Map learning and high-speed navigation in RHINO*. In Kortenkamp et al. [31], 1997.

[50] S. Thrun, W. Burgard, and D. Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, 31:29–53, 1998.