CHAPTER 20
Planning in Robotics

20.1 Introduction

A robot is a machine capable of performing a collection of tasks with some level of autonomy and flexibility, in some type of environment. To achieve this capability, a robot integrates one or several sensory-motor functions together with communication and information processing capabilities. Examples of sensory-motor functions in a robot include locomotion on wheels, legs, or wings; manipulation with one or several mechanical arms, grippers, and hands; localization with odomoters, sonars, lasers, inertial sensors, and/or fixed devices such as the Global Positioning System (GPS); and scene analysis and environment modeling with a stereovision system on a pan-and-tilt platform.

Robots can be designed for specific tasks and environments such as:

- Manufacturing tasks: painting, welding, loading/unloading a power press or a machine tool, assembling parts.
- Servicing stores, warehouses, and factories: maintaining, surveying, cleaning the area, transporting objects.
- Exploring an unknown natural areas, e.g., in planetary exploration: building a map with characterized landmarks, extracting samples, setting various measurement devices.
- Assisting people in offices, public areas, and homes.
- Helping in tele-operated surgical operations, as in the so-called minimal invasive surgery.

Robotics is a reasonably mature technology when robots are restricted to operating within well-known and well-engineered environments, e.g., as in manufacturing robotics, or to performing single simple tasks, 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 robots deployed today in the manufacturing industry do not perform planning per se.[1] Using a robot without planning capabilities basically requires hand-coding the environment model and the robot's skills and strategies into a *reactive controller.* This is a perfectly feasible approach as long as this hand-coding is inexpensive and reliable enough for the application at hand. This will be 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 human-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. For example, this can be done 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, and it will augment the robot's usefulness and robustness. Planning should not be seen as opposed to the reactive capabilities of a robot, whether hand-coded or learned; neither should it be seen as opposed to its learning capabilities. Planning should be closely integrated with these capabilities.

The specific requirements of planning in robotics, as compared with 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; and
- 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 and computer animation. Perception planning is a younger and much more open area, although some

1. Sophisticated path and manipulation planning may be deployed during the design stage.

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 handle time and resource allocation, dynamic environments uncertainty and partial knowledge, and incremental planning consistently integrated with 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 and retrieve and pick up object2. However, 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, and SRCs, enable manual programming of 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 offline generation of 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 plan for pursuing the activity.

This chapter illustrates the usefulness of planning techniques, namely HTNs and MDPs (Chapters 11 and 16, respectively), for the design of a high-level navigation controller for a mobile robot (Section 20.3). The approach is by no means limited to navigation tasks. It can be pursued for a wide variety of robotics tasks, such as object manipulation and cleaning. However, to keep the presentation technically grounded with enough details about the sensory-motor functions controlled, it is necessary to illustrate a specific task. Several sensory-motor functions are presented and discussed in Section 20.3.1; an approach that exemplifies the use of planning techniques for synthesizing alternative plans and policies for a navigation task is described. Before that, the chapter introduces the reader to the important and mature area of path and motion planning (Section 20.2). The use of the techniques described here as well as in Chapters 14 and 15 is then briefly illustrated in Section 20.4. Finally, the relevant references for this chapter, such as Latombe [353], Siméon *et al.* [477], Beetz *et al.* [58], and others, are discussed in Section 20.5.

20.2 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, the focus of the rest of this section.

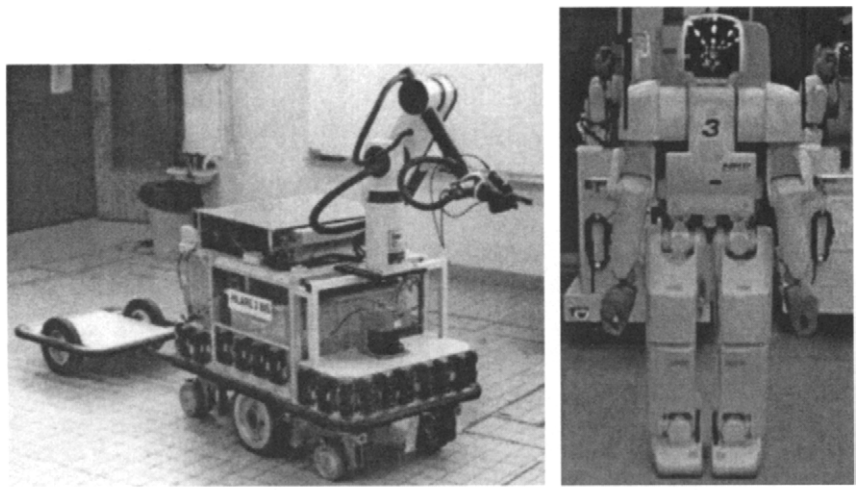
If the mobile system of interest is a *free-flying rigid body*, i.e., if it can move freely in space in any direction without any kinematics constraints, then six *configuration parameters* are needed to characterize its position: x, y, z, and the three Euler angles. Path planning defines a path in this six-dimensional space. However, a robot is not a free-flying body. Its kinematics defines its possible motion. For example, a carlike 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 sideways. 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 carlike robot Hilare in Figure 20.1 (a) has a total of 10 configuration parameters: 6 for the arm and 4 for the mobile platform with the trailer [349]. The humanoid robot HRP in Figure 20.1 (b) has 52 configuration parameters: 2 for the head, 7 for each arm, 6 for each leg and 12 for each hand (four fingers, each with three configurations) [280, 308].[2]

Given a robot with n configuration parameters and some environment, let us define the following variables.

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

Path planning is the problem of finding a path in the free configuration space CS_{free} between an initial configuration 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.[3] However, the explicit definition of CS_{free} is a computationally difficult problem, theoretically (it is exponential in the dimension of CS) and practically. For example, the configuration space corresponding to the

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 carlike robot has three configuration parameters but only two degrees of freedom.
3. Recall our remark in Section 1.5: *if we are given explicitly the graph Σ, then classical planning is a simple graph search problem.* However, the explicit definition of a continuous search space by a set of equations does not provide its connexity structure. The search in CS_{free} is not as easy as a graph search.



(a) (b)

Figure 20.1 Hilare, a carlike robot with an arm and a trailer (a); HRP, a humanoid robot (b).

simple carlike robot shown in Figure 20.2 (a) is the complex three-dimensional structure shown in Figure 20.2 (b). 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.

1. *Collision checking* checks whether a configuration $q \in CS_{free}$ or whether a path between two configurations in CS is collision free, i.e., whether it lies entirely in CS_{free}.
2. *Kinematics steering* 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 [245]. 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 *Reede&Shepp curves* [454]) are applied to carlike robots that cannot move sideways. If the robot has no kinematics 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 20.1 (a), straight line segments for the arm are combined with dedicated curves for the mobile platform with a trailer [349].

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

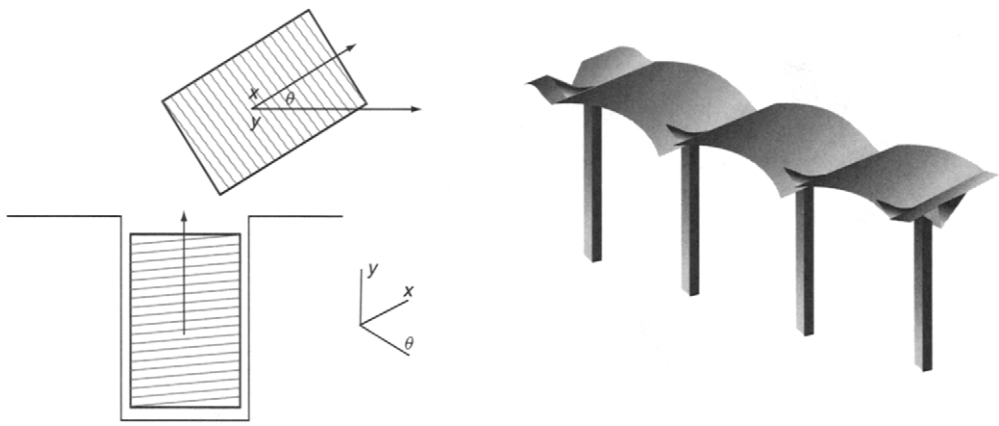


Figure 20.2 A simple carlike robot that is an $l \times l'$ rectangle with three configuration parameters, x, y, and θ, in a 2D environment (a) that looks like two-dimensional a pit of width L and depth h, and the corresponding configuration space (b).

Definition 20.1 A *roadmap* $\mathcal{R}$ for CS_{free} is a graph whose finite set of vertices are configurations in CS_{free}; two such vertices q and q' are adjacent in $\mathcal{R}$ only if $\mathcal{L}(q, q')$ is in CS_{free}. ■

Because $\mathcal{L}$ 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 configurations q_i and q_g in CS_{free}, a feasible path from q_i to q_g can be found as follows.

1. Find a configuration $q_i' \in \mathcal{R}$ such that $\mathcal{L}(q_i, q_i') \in CS_{free}$.
2. Find a configuration $q_g' \in \mathcal{R}$ such that $\mathcal{L}(q_g, q_g') \in CS_{free}$.
3. 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 postprocessing 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 kinematic 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 this set

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

```
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 20.3 A probabilistic roadmap generation algorithm for path planning.

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

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

The algorithm Probabilistic-Roadmap (Figure 20.3) 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 the connection of two configurations in $\mathcal{R}$ that are not already connected in $\mathcal{R}$.

Let us assume that there is a finite set Q that covers CS_{free}.[4] 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 $q \in \mathcal{D}(q_1) \cap \mathcal{D}(q_2)$ 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 reflect those of CS_{free}.

The Probabilistic-Roadmap algorithm generates a roadmap that meets this property not *deterministically* but only up to some probability value, which is linked to

4. 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} [320].

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 20.3). 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.

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 a 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 a few milliseconds. This is illustrated for the Hilare robot in Figure 20.4, where the task is to carry a long rod that constrains the path through the door: the roadmap in this nine-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.

To summarize, the principle of the roadmap technique is to cover the search space at a preprocessing stage with a set of connected nodes and, during the planning stage, to connect the initial and goal states to the roadmap and to search through the roadmap. A natural question then is whether this technique can be applied to

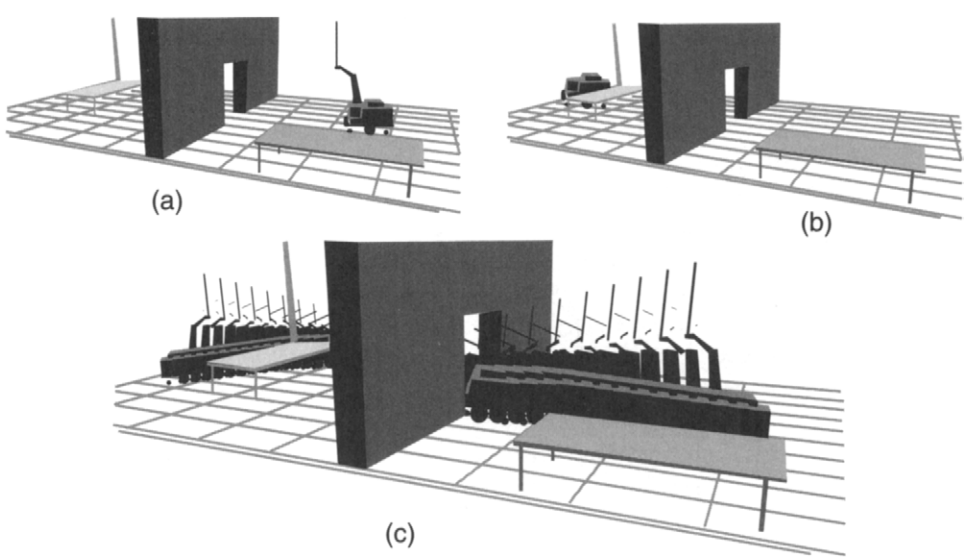


Figure 20.4 Initial (a) and goal (b) configurations of a path-planning problem and the generated path (c).

the state space Σ in domain-independent planning. Although Σ does not have the nice topological properties of CS that enable $\mathcal{L}(q, q')$ to be computed easily, it turns out that the roadmap principle can be successfully applied to classical planning, as shown by Guere and Alami [251].

20.3 Planning for the Design of a Robust Controller

Consider an autonomous mobile robot in a structured environment, such as the robot in Figure 20.1 (a), which is equipped with several sensors—sonar, laser, vision—and actuators, and with an arm. The robot also has several software modules for the same sensory-motor (*sm*) function, e.g., for localization, for map building and updating, and for motion planning and control. These redundant *sm* functions are needed because of possible sensor failures and because no single method or sensor has 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 here illustrate this capability. They enable a designer to specify, offline, very robust ways to perform a task such as navigate to. The designer specifies a collection of HTNs (see Chapter 11), as illustrated later in this chapter in Figure 20.5, that are complex plans, called *modes of behavior*, or *modalities* for short,[5] whose primitives are *sm* functions. Each modality is a possible way to combine 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 obvious. However, the relationship between control states and modalities can be expressed as a Markov Decision Process (see Chapter 16). This MDP characterizes the robot's 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 online from this MDP.

To summarize, this approach involves three components.

1. Sensory-motor functions are the primitive actions.
2. Modalities are HTN plans. Alternate modalities offer different ways to combine the *sm* functions within a task.
3. MDP policies are used by the controller to achieve the task.

Let us describe these three levels successively.

5. Generally, *behavior* in robotics has a meaning different than our modalities.

20.3.1 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 two-dimensional 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 two-dimensional map. Cells 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.[6] 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 either indicating the end of a normal execution or giving additional information about nonnominal execution. In order to give 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 relocalization 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. 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 reactivates the relocalization mode. This can be repeated in order to find a nonambiguous corner in the environment to restart the localization loop.

Localization on Visual Landmarks. This function relies on calibrated monocular vision to detect known landmarks such as doors and wall posters. It derives from

6. Some environment modeling techniques enable one to automatically acquire such a topological graph with the cells and their labels. They are discussed in Section 20.5. However, in the work referred to here, the topological graph is hand-programmed.

the perceptual data a very accurate estimation of the robot position. The setup is simple: a few wall posters and characteristic planar features on walls are learned in supervised mode. However, landmarks are generally 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. Feedback from this *sm* function is a report of a potentially visible landmark, which indicates that the robot has entered an area of visibility of a landmark. The robot stops and turns toward the expected landmark; it searches using the pan-and-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.

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. Connectivity between configurations relies on a set of internal forces 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 the robot away from obstacles. This *sm* function takes into account the planned path, the map, and the online 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 that may limit the reactivity in certain cluttered, dynamic environments. This also limits the band length.

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 planned 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 planned path. In this case, a new band has to be created.

Reactive Obstacle Avoidance. This *sm* function provides a reactive motion capability toward 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 20.2) may also be seen as an *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 configurations 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.

20.3.2 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 one way to perform 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 in Section 20.3.3 how the controller chooses the appropriate modality. Let us exemplify some such modalities for the navigation task before giving the details of the HTN representation for modalities and the associated control system.

Modality M_1 uses three *sm* functions: the path planner, the elastic band for the dynamic motion execution, and 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. Suspending the modality stops the band execution. The path, the band, and the localization loop are maintained. A suspended modality can be resumed by restarting the execution of the current elastic band.

Modality M_2 uses three *sm* functions: the path planner, reactive obstacle avoidance, and 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

modality M_1 in order to avoid unreliable replanning 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 crossing a corridor without accurate knowledge of the robot position. Typically this M_4 modality can be used in long corridors. The growing inaccuracy can make it difficult to find 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 the odometry estimate. Furthermore, the reactive navigation of M_4 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 HTNs. 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 available here.

An internal node of the HTN AND/OR tree is a task or 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 branches 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 an *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 *nonblocking*. 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 nonblocking mode, actions in a sequence are triggered sequentially without waiting for 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 nonblocking primitive action may occur and be processed after an unpredictable delay.

The modality tree illustrated in Figure 20.5 starts with six OR-connectors labeled start, stop, suspend, resume, success, 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

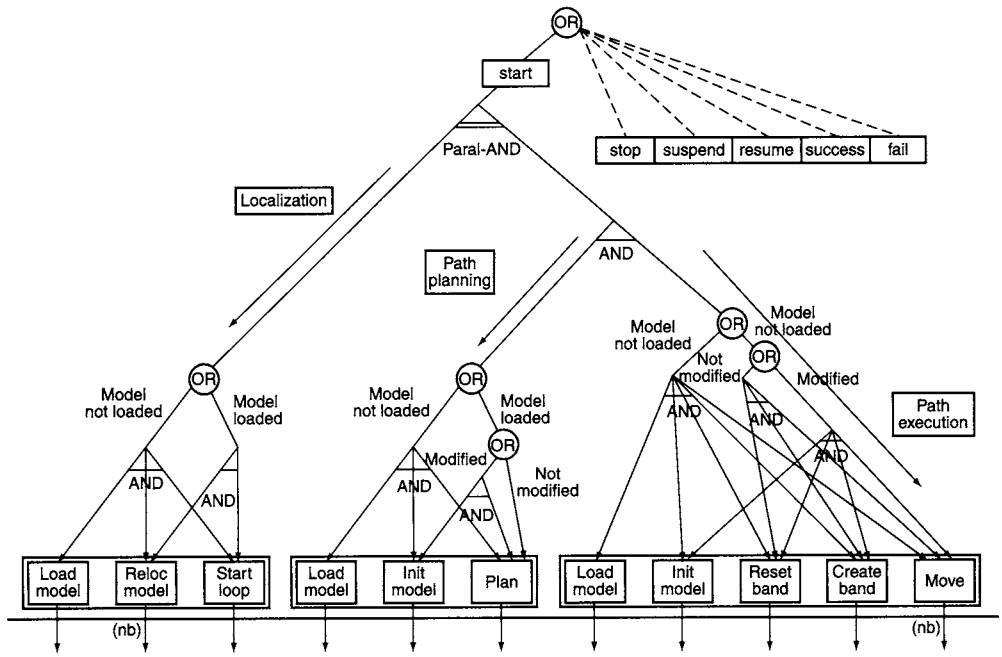


Figure 20.5 Part of modality M_1.

its previous form. The suspend and resume connectors are triggered by the control system described in the next paragraph. The suspend connector allows one 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 and success connectors are followed when the modality execution reaches a failure or a success, respectively. 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 signals a normal execution. Otherwise, a nonnominal report signals a particular type of *sm* function execution. The aim of the corresponding control tree is to recover to a nominal modality execution. Some nonnominal reports can be nonrecoverable 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 nonsharable resources (e.g., motors, cameras, pan-and-tilt mounts) or logical resources (the environment model that can be temporally modified). The execution of a set of concurrent nonblocking 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 nonnominal 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 and 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.

20.3.3 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* has 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 include the following.

- The *cluttering of the environment*, which is defined to be a weighted sum of the distances to the nearest obstacles perceived by the laser, with a dominant weight along the robot motion axis. This is an important piece of information for establishing 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. 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 covariance 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 the 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, Open Area with fixed localization devices.

- The *current modality.* This information is essential to assess the control state and possible transitions between modalities.

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 us 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 $\Sigma = \{S, A, P, C\}$, where:

- 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 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 five modalities and about a few thousand states. P and C are obtained from observed statistics during a learning phase.

The control automaton Σ is an MDP. As such, Σ could be used reactively on the basis of a universal policy π that 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 explicitly takes into account 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 20.6).

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 the estimated costs of edges between topological cells. This route will help find in the

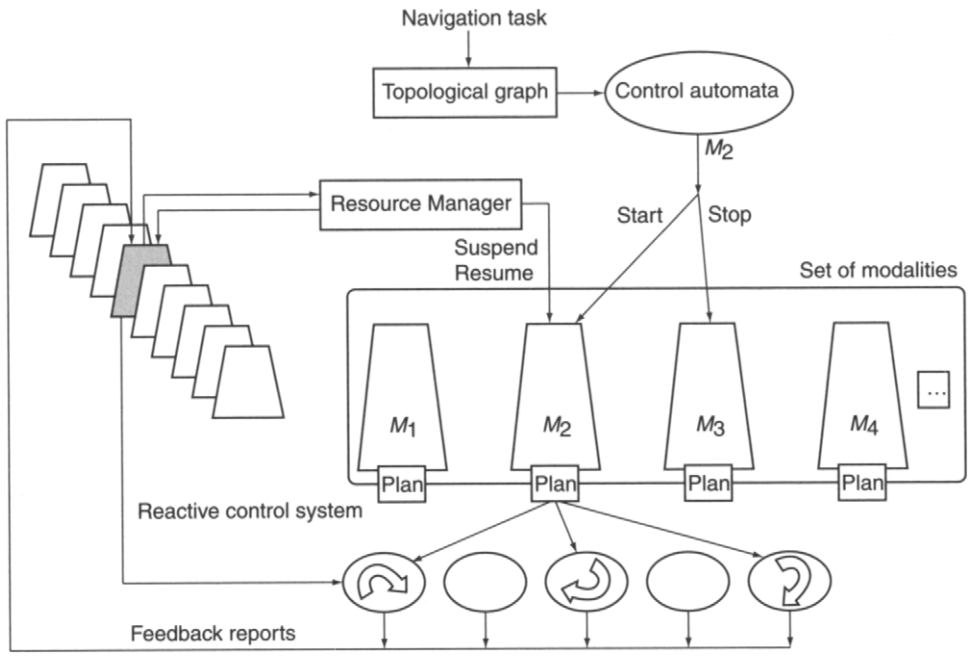


Figure 20.6 The ROBEL control system.

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, which 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.

- $\sum_{path} C(a, s, s') \geq Kl_r$, K being a constant ratio between the cost of a state transition in the control automaton and the 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 us 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 path, the predicate route(s_0, s) is checked in a straightforward way, which gives S_g.

It is important to note that this set S_g of control states is a *heuristic projection* of the planned route to the goal. There is no guarantee 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 will correspond to a sequence of control states that meets route(s_0, s). This is 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 (Figure 16.6) 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. First, the computed modality $\pi(s_0)$ is executed. Then, the robot observes the new control state s, updates its route r and its set S_g of goal states with respect to s, and 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 of 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 can be 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. Another option is to use Σ 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.

20.3.4 Analysis of the Approach

The system described here was deployed on an indoor mobile platform and experimented with in navigation tasks within a wide laboratory environment. 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.

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 because it enables one 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 online, 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.

20.4 Dock-Worker Robots

Up to here, the running example of the book has been the DWR domain, described as an abstract, highly simplified world. Now we can describe this application domain at a more concrete level.

A container is a large metallic cell of a standard size that can be conveniently piled on docks and loaded on ships, trains, and cargo planes. It is intended to allow safe transportation of some freight from a shipping point to a destination point. A significant part of the shipment cost lies in the transition phase between two transportation medias, e.g., when a container has to be moved between two ships, or from a ship to a train or a truck, usually via some storage area. The high cost of these *transshipment* operations explains the motivation of their automatization. Several sites, such as the Rotterdam Harbor (see Figure 20.7), already perform transshipment operations with Automated Ground Vehicles (AGVs). These AGVs are mostly teleoperated. They require fixed navigation equipment and a site specifically designed to suit their human-controlled operations. Ambitious projects aim at more flexible and autonomous operations.

One such project, the Martha project [10], has studied the use of autonomous robots evolving in already existing sites that are not designed specifically for AGVs

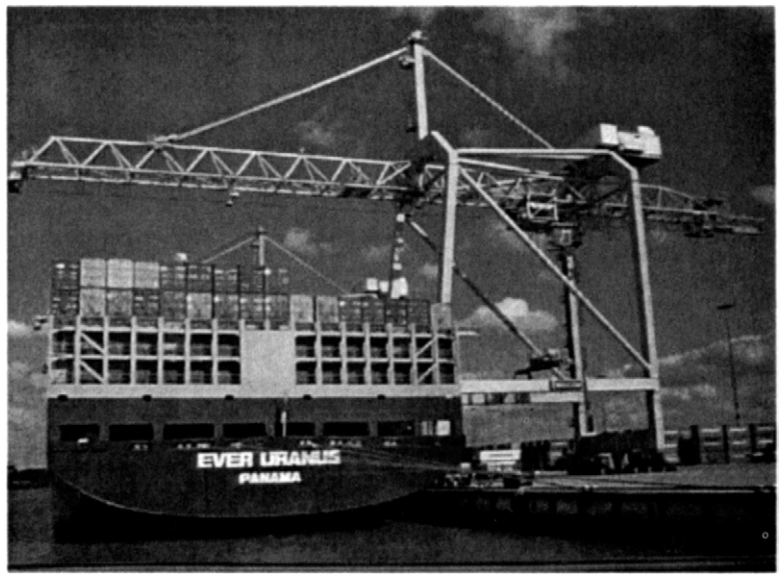


Figure 20.7 A crane loading containers on the *Ever Uranus* ship in the Rotterdam Harbor.

and that can be used by other vehicles. A container transportation robot is a trucklike robot, about 17 meters long, equipped with the type of range and localization sensors discussed in Section 20.3.1. A typical setting is that of a large harbour where a fleet of 50 to 100 such robots is in charge of loading and unloading containers from arriving vehicles to departing ones. In the Martha project, a map of the environment is provided to each robot (see Figure 20.8). This map is composed of metrical data and a topological graph labeled with a few types of attributes (routes, loading/unloading zones, docks, piles, etc.). The map is static, but the environment evolves. Because of other vehicles and nonmodeled obstacles, the current state of the environment may depart significantly from the map.

The planning and the execution control of a transshipment task are managed in the Martha project at several levels. At the highest level, the mission allocation is performed incrementally by a centralized planner that allocates to each robot the container transportation jobs it has to perform. This planner, detailed in Vidal *et al.* [535], relies on the scheduling techniques of Chapter 15. It views the robots as resources and container transportation tasks as jobs to be allocated to available robots. It can handle priorities and cost estimates of jobs, as well as a fairly flexible model of the uncertainty of the specified tasks, e.g., the uncertainty of the arrival time of the ships to be handled. The planner works incrementally, taking into account newly specified transportation tasks with respect to the running ones; it is able to modify part of the already allocated missions whose execution has not started yet.

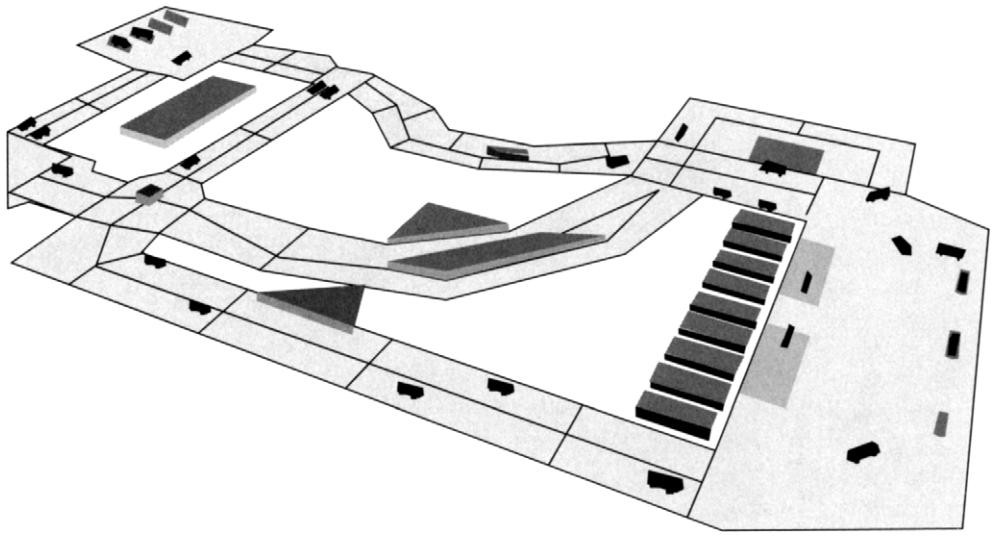


Figure 20.8 A dock work environment.

A robot plans and executes its allocated missions autonomously. Each job involves going to a loading position, taking a container, and moving it to an unloading position. However, a robot has to share many common resources of the site with the other robots of the fleet: routes, loading positions, and cranes. The Martha project developed an efficient distributed approach for coordinating without conflict a large set of plans: the *plan-merging technique.*

Let us assume that all plans currently being executed by the fleet of robots are conflict free. A robot r_i receives a new mission that allocates to it a sequence of jobs. While taking into account its current plan, robot r_i synthesizes a *provisional plan* for handling this mission. The provisional plan involves precise trajectories, generated with a planner similar to the one described in Section 20.2, and time-point variables. These temporal variables will be instantiated at the execution level by the actions of r_i as well as by external events, such as the actions of other robots. The provisional plan has temporal constraints between the time points of its actions and those of the external events.

Through an Ethernet radio link, the robot r_i advertises to the fleet of robots the resources it intends to use according to this provisional plan. Every robot that is going to use one or several of these resources returns to r_i its current plan. Robot r_i modifies its provisional plan in order to remove all resource conflicts. This is done by adding temporal constraints on the actions of the provisional plan of r_i, using the techniques of Chapter 14. This is the plan-merging step.

However, r_i may not succeed in finding a conflict-free plan because it cannot modify the plans of the other robots and because a robot may keep a resource

(e.g., stay idle in some position). In that case r_i waits for this resource to become available before planning further its new mission.[7] Furthermore, r_i checks that there is no deadlock loop of robots waiting for this resource. If there is such a deadlock, then a centralized plan is needed for the set of robots in the deadlock loop. The planning is performed by the robot who finds the deadlock.

The total interaction between robots relies on a token ring protocol where only one robot at a time is performing plan merging. It can be shown that as long as there are no deadlines on the goals to be achieved, but only priorities and costs, the plan-merging technique is sound and complete.

The Martha project developed this approach while taking into account all the needed levels of execution and control.[8] It has been experimented with successfully on three robots. The approach was shown, in realistic simulations over a wide network of computers, to scale up successfully to 50 robots without any significant overhead due to the robots' interactions and plan sharing in the distributed planning schema.

20.5 Discussion and Historical Remarks

Robot motion planning is a very advanced research field [319, 354]. The early techniques in the 1980s were mostly dedicated to deterministic algorithms [353]. They led to a good understanding and formalization of the problem, as well as to several developments on related topics such as manipulation planning [11]. More recent approaches have built on this state of the art with probabilistic algorithms that permitted a significant scale-up [48]. The probabilistic roadmap techniques introduced by Kavraki *et al.* [321] gave rise to several successful developments [68, 85, 252, 273, 320, 338, 477], which today represent 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, and in CAD and graphics animation. The illustrations and performance figures in Section 20.2 are borrowed from Move3D, a state-of-the-art system implementing roadmap techniques [476].

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. Their detailed discussion is well beyond the scope of this section. However, several approaches are directly relevant to this chapter, such as the techniques used for localization and mapping, e.g., the SLAM methods [156, 404, 508, 509]; the methods for structuring the environment model into a topological map with areas labeled by different navigation colors [347, 507]; the visual localization techniques [264]; and the flexible control techniques [450, 462].

7. This waiting takes place at the planning level only, while the execution of already coordinated plans is being pursued.
8. Except for the operations of the cranes for loading the robots.

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 [93, 191, 214, 279]. More recent developments on these systems (e.g., [154]), aim at a closer integration to planning. The behavior-based controllers (e.g., [23]), which usually focus on a more reactive set of concurrent activities, have also led to more goal-directed developments (e.g., [268]). The *robot architecture* (i.e., the organization that enables proper integration of the sensory-motor functions, the reactive control system, and the deliberative capabilities [9, 479]) remains an important issue.

The planning and robotics literature reports on several plan-based robot controllers with objectives similar to those discussed in this chapter [55, 56, 57, 58, 337]. The approach of Beetz [56] has also been deployed for controlling an indoor robot carrying out the core tasks of an office courier. It relies on the SRC's reactive controllers. These are concurrent control routines that adapt to changing conditions by reasoning on and modifying plans. They rely on the XFRM system, which manipulates reactive plans and is able to acquire them through learning with XFRMLEARN [57]. The approach illustrated in Section 20.3 on navigation tasks was developed by Morisset and Ghallab [401, 402] and extensively experimented with on the Diligent robot, an indoor mobile platform. The approach is not limited to navigation; it can be deployed on other robot activities.