5 Mobile Robot Localization

5.1 Introduction

Navigation is one of the most challenging competences required of a mobile robot. Success in navigation requires success at the four building blocks of navigation: *perception* (the robot must interpret its sensors to extract meaningful data); *localization* (the robot must determine its position in the environment, figure 5.1); *cognition* (the robot must decide how to act to achieve its goals); and *motion control* (the robot must modulate its motor outputs to achieve the desired trajectory).

Of these four components (figure 5.2), localization has received the greatest research attention in the past decade, and as a result, significant advances have been made on this front. In this chapter, we explore the successful localization methodologies of recent years. First, section 5.2 describes how sensor and effector uncertainty is responsible for the difficulties of localization. Section 5.3 describes two extreme approaches to dealing with the challenge of robot localization: avoiding localization altogether, and performing explicit map-based localization. The remainder of the chapter discusses the question of representa-

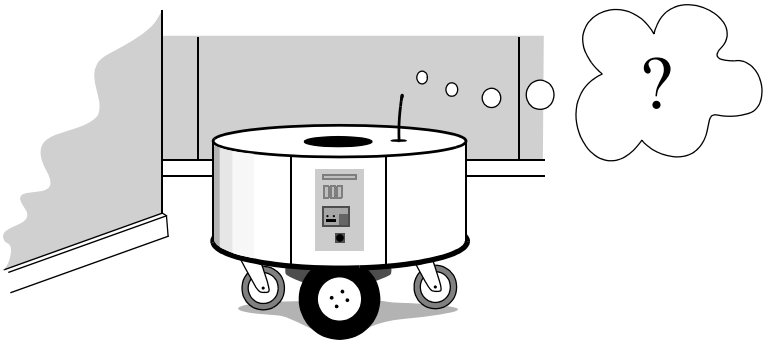


Figure 5.1
Where am I?

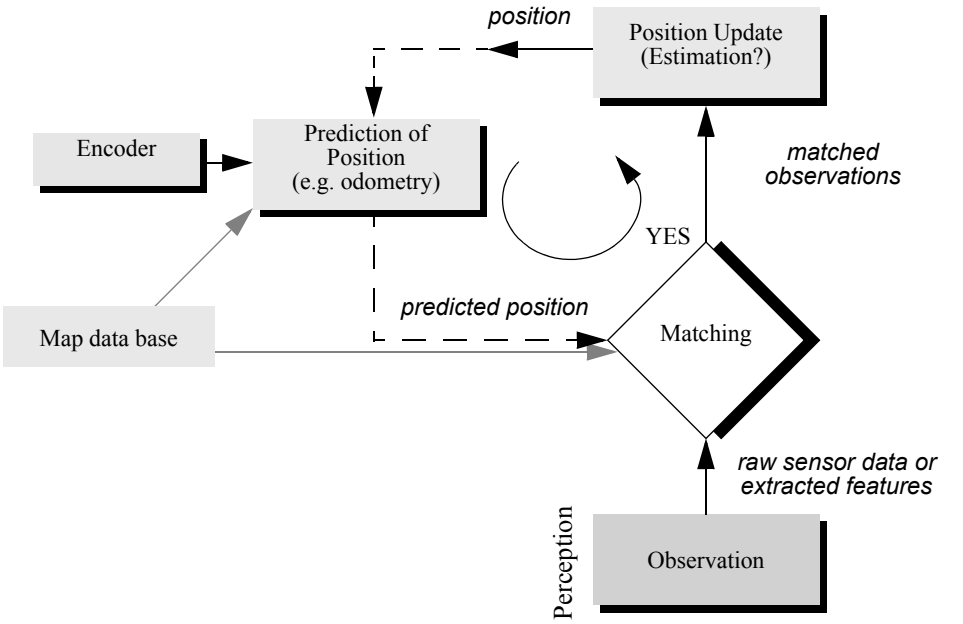


Figure 5.2
General schematic for mobile robot localization.

tion, then presents case studies of successful localization systems using a variety of representations and techniques to achieve mobile robot localization competence.

5.2 The Challenge of Localization: Noise and Aliasing

If one could attach an accurate GPS (global positioning system) sensor to a mobile robot, much of the localization problem would be obviated. The GPS would inform the robot of its exact position, indoors and outdoors, so that the answer to the question, "Where am I?" would always be immediately available. Unfortunately, such a sensor is not currently practical. The existing GPS network provides accuracy to within several meters, which is unacceptable for localizing human-scale mobile robots as well as miniature mobile robots such as desk robots and the body-navigating nanorobots of the future. Furthermore, GPS technologies cannot function indoors or in obstructed areas and are thus limited in their workspace.

But, looking beyond the limitations of GPS, localization implies more than knowing one's absolute position in the Earth's reference frame. Consider a robot that is interacting with humans. This robot may need to identify its absolute position, but its relative position

with respect to target humans is equally important. Its localization task can include identifying humans using its sensor array, then computing its relative position to the humans. Furthermore, during the *cognition* step a robot will select a strategy for achieving its goals. If it intends to reach a particular location, then localization may not be enough. The robot may need to acquire or build an environmental model, a *map*, that aids it in planning a path to the goal. Once again, localization means more than simply determining an absolute pose in space; it means building a map, then identifying the robot's position relative to that map.

Clearly, the robot's sensors and effectors play an integral role in all the these forms of localization. It is because of the inaccuracy and incompleteness of these sensors and effectors that localization poses difficult challenges. This section identifies important aspects of this sensor and effector suboptimality.

5.2.1 Sensor noise

Sensors are the fundamental robot input for the process of *perception*, and therefore the degree to which sensors can discriminate the world state is critical. *Sensor noise* induces a limitation on the consistency of sensor readings in the same environmental state and, therefore, on the number of useful bits available from each sensor reading. Often, the source of sensor noise problems is that some environmental features are not captured by the robot's representation and are thus overlooked.

For example, a vision system used for indoor navigation in an office building may use the color values detected by its color CCD camera. When the sun is hidden by clouds, the illumination of the building's interior changes because of the windows throughout the building. As a result, hue values are not constant. The color CCD appears noisy from the robot's perspective as if subject to random error, and the hue values obtained from the CCD camera will be unusable, unless the robot is able to note the position of the sun and clouds in its representation.

Illumination dependence is only one example of the apparent noise in a vision-based sensor system. Picture jitter, signal gain, blooming, and blurring are all additional sources of noise, potentially reducing the useful content of a color video image.

Consider the noise level (i.e., apparent random error) of ultrasonic range-measuring sensors (e.g., sonars) as discussed in section 4.1.2.3. When a sonar transducer emits sound toward a relatively smooth and angled surface, much of the signal will coherently reflect away, failing to generate a return echo. Depending on the material characteristics, a small amount of energy may return nonetheless. When this level is close to the gain threshold of the sonar sensor, then the sonar will, at times, succeed and, at other times, fail to detect the object. From the robot's perspective, a virtually unchanged environmental state will result in two different possible sonar readings: one short and one long.

The poor signal-to-noise ratio of a sonar sensor is further confounded by interference between multiple sonar emitters. Often, research robots have between twelve and forty-

eight sonars on a single platform. In acoustically reflective environments, multipath interference is possible between the sonar emissions of one transducer and the echo detection circuitry of another transducer. The result can be dramatically large errors (i.e., underestimation) in ranging values due to a set of coincidental angles. Such errors occur rarely, less than 1% of the time, and are virtually random from the robot's perspective.

In conclusion, sensor noise reduces the useful information content of sensor readings. Clearly, the solution is to take multiple readings into account, employing temporal fusion or multisensor fusion to increase the overall information content of the robot's inputs.

5.2.2 Sensor aliasing

A second shortcoming of mobile robot sensors causes them to yield little information content, further exacerbating the problem of perception and, thus, localization. The problem, known as *sensor aliasing*, is a phenomenon that humans rarely encounter. The human sensory system, particularly the visual system, tends to receive unique inputs in each unique local state. In other words, every different place looks different. The power of this unique mapping is only apparent when one considers situations where this fails to hold. Consider moving through an unfamiliar building that is completely dark. When the visual system sees only black, one's localization system quickly degrades. Another useful example is that of a human-sized maze made from tall hedges. Such mazes have been created for centuries, and humans find them extremely difficult to solve without landmarks or clues because, without visual uniqueness, human localization competence degrades rapidly.

In robots, the nonuniqueness of sensor readings, or *sensor aliasing*, is the norm and not the exception. Consider a narrow-beam rangefinder such as an ultrasonic or infrared rangefinder. This sensor provides range information in a single direction without any additional data regarding material composition such as color, texture, and hardness. Even for a robot with several such sensors in an array, there are a variety of environmental states that would trigger the same sensor values across the array. Formally, there is a many-to-one mapping from environmental states to the robot's perceptual inputs. Thus, the robot's percepts cannot distinguish from among these many states. A classic problem with sonar-based robots involves distinguishing between humans and inanimate objects in an indoor setting. When facing an apparent obstacle in front of itself, should the robot say "Excuse me" because the obstacle may be a moving human, or should the robot plan a path around the object because it may be a cardboard box? With sonar alone, these states are aliased, and differentiation is impossible.

The problem posed to navigation because of sensor aliasing is that, even with noise-free sensors, the amount of information is generally insufficient to identify the robot's position from a single-percept reading. Thus, techniques must be employed by the robot programmer that base the robot's localization on a series of readings and, thus, sufficient information to recover the robot's position over time.

5.2.3 Effector noise

The challenges of localization do not lie with sensor technologies alone. Just as robot sensors are noisy, limiting the information content of the signal, so robot effectors are also noisy. In particular, a single action taken by a mobile robot may have several different possible results, even though from the robot's point of view the initial state before the action was taken is well known.

In short, mobile robot effectors introduce uncertainty about future state. Therefore, the simple act of moving tends to increase the uncertainty of a mobile robot. There are, of course, exceptions. Using *cognition,* the motion can be carefully planned so as to minimize this effect, and indeed sometimes to actually result in more certainty. Furthermore, when the robot's actions are taken in concert with careful interpretation of sensory feedback, it can compensate for the uncertainty introduced by noisy actions using the information provided by the sensors.

First, however, it is important to understand the precise nature of the effector noise that impacts mobile robots. It is important to note that, from the robot's point of view, this error in motion is viewed as an error in odometry, or the robot's inability to estimate its own position over time using knowledge of its kinematics and dynamics. The true source of error generally lies in an incomplete model of the environment. For instance, the robot does not model the fact that the floor may be sloped, the wheels may slip, and a human may push the robot. All of these unmodeled sources of error result in inaccuracy between the physical motion of the robot, the intended motion of the robot, and the proprioceptive sensor estimates of motion.

In odometry (wheel sensors only) and dead reckoning (also heading sensors) the position update is based on *proprioceptive* sensors. The movement of the robot, sensed with wheel encoders or heading sensors or both, is integrated to compute position. Because the sensor measurement errors are integrated, the position error accumulates over time. Thus, the position has to be updated from time to time by other localization mechanisms. Otherwise the robot is not able to maintain a meaningful position estimate in the long run.

In the following we concentrate on odometry based on the wheel sensor readings of a differential-drive robot only (see also [5, 99, 102]). Using additional heading sensors (e.g., gyroscope) can help to reduce the cumulative errors, but the main problems remain the same.

There are many sources of odometric error, from environmental factors to resolution:

- Limited resolution during integration (time increments, measurement resolution, etc.);
- Misalignment of the wheels (deterministic);
- Uncertainty in the wheel diameter and in particular unequal wheel diameter (deterministic);
- Variation in the contact point of the wheel;

- Unequal floor contact (slipping, nonplanar surface, etc.).

Some of the errors might be *deterministic* (systematic); thus, they can be eliminated by proper calibration of the system. However, there are still a number of *nondeterministic* (random) errors that remain, leading to uncertainties in position estimation over time. From a geometric point of view, one can classify the errors into three types:

1. Range error: integrated path length (distance) of the robot's movement
   $\rightarrow$ sum of the wheel movements
2. Turn error: similar to range error, but for turns
   $\rightarrow$ difference of the wheel motions
3. Drift error: difference in the error of the wheels leads to an error in the robot's angular orientation

Over long periods of time, turn and drift errors far outweigh range errors, since their contribution to the overall position error is nonlinear. Consider a robot whose position is initially perfectly wellknown, moving forward in a straight line along the x-axis. The error in the y-position introduced by a move of d meters will have a component of $d \sin \Delta\theta$, which can be quite large as the angular error $\Delta\theta$ grows. Over time, as a mobile robot moves about the environment, the rotational error between its internal reference frame and its original reference frame grows quickly. As the robot moves away from the origin of these reference frames, the resulting linear error in position grows quite large. It is instructive to establish an error model for odometric accuracy and see how the errors propagate over time.

5.2.4 An error model for odometric position estimation

Generally the pose (position) of a robot is represented by the vector

$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}. \tag{5.1}$$

For a differential-drive robot (figure 5.3) the position can be estimated starting from a known position by integrating the movement (summing the incremental travel distances). For a discrete system with a fixed sampling interval Δt, the incremental travel distances $(\Delta x; \Delta y; \Delta\theta)$ are

$$\Delta x = \Delta s \cos(\theta + \Delta\theta/2), \tag{5.2}$$

$$\Delta y = \Delta s \sin(\theta + \Delta\theta/2), \tag{5.3}$$

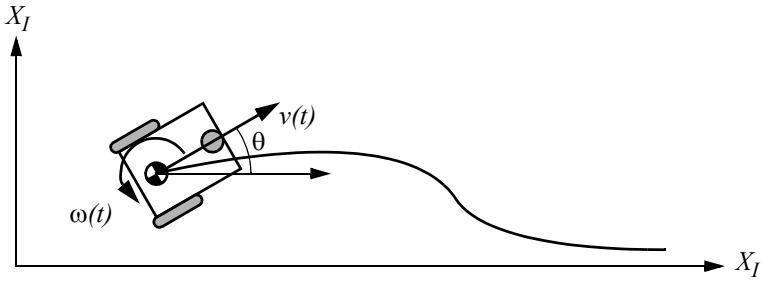


Figure 5.3
Movement of a differential-drive robot.

$$\Delta\theta = \frac{\Delta s_r - \Delta s_l}{b}, \tag{5.4}$$

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2}, \tag{5.5}$$

where

$(\Delta x;\Delta y;\Delta\theta)$ = path traveled in the last sampling interval;

$\Delta s_r;\Delta s_l$ = traveled distances for the right and left wheel respectively;

b = distance between the two wheels of differential-drive robot.

Thus we get the updated position p' :

$$p' = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = p + \begin{bmatrix} \Delta s \cos(\theta + \Delta\theta/2) \\ \Delta s \sin(\theta + \Delta\theta/2) \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta s \cos(\theta + \Delta\theta/2) \\ \Delta s \sin(\theta + \Delta\theta/2) \\ \Delta\theta \end{bmatrix}. \tag{5.6}$$

By using the relation for $(\Delta s;\Delta\theta)$ of equations (5.4) and (5.5) we further obtain the basic equation for odometric position update (for differential drive robots):

$$p' = f(x, y, \theta, \Delta s_r, \Delta s_l) = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} \cos\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r + \Delta s_l}{2} \sin\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r - \Delta s_l}{b} \end{bmatrix}. \quad (5.7)$$

As we discussed earlier, odometric position updates can give only a very rough estimate of the actual position. Owing to integration errors of the uncertainties of p and the motion errors during the incremental motion $(\Delta s_r;\Delta s_l)$, the position error based on odometry integration grows with time.

In the next step we will establish an error model for the integrated position p' to obtain the covariance matrix $\Sigma_{p'}$ of the odometric position estimate. To do so, we assume that at the starting point the initial covariance matrix Σ_p is known. For the motion increment $(\Delta s_r;\Delta s_l)$ we assume the following covariance matrix Σ_Δ:

$$\Sigma_\Delta = covar(\Delta s_r, \Delta s_l) = \begin{bmatrix} k_r|\Delta s_r| & 0 \\ 0 & k_l|\Delta s_l| \end{bmatrix}, \quad (5.8)$$

where Δs_r and Δs_l are the distances traveled by each wheel, and k_r, k_l are error constants representing the nondeterministic parameters of the motor drive and the wheel-floor interaction. As you can see, in equation (5.8) we made the following assumptions:

- The two errors of the individually driven wheels are independent,[22]
- The variance of the errors (left and right wheels) are proportional to the absolute value of the traveled distances $(\Delta s_r;\Delta s_l)$.

These assumptions, while not perfect, are suitable and will thus be used for the further development of the error model. The *motion errors* are due to imprecise movement because of deformation of wheel, slippage, unequal floor, errors in encoders, and so on. The values for the error constants k_r and k_l depend on the robot and the environment and should be experimentally established by performing and analyzing representative movements.

If we assume that p and $\Delta_{rl} = [\Delta s_r, \Delta s_l]^T$ are uncorrelated and the derivation of f (equation [5.7]) is reasonably approximated by the first-order Taylor expansion (linearization), we conclude, using the error propagation law (see section 4.1.3.2),

22. If there is more knowledge regarding the actual robot kinematics, the correlation terms of the covariance matrix could also be used.

$$\Sigma_{p'} = \nabla_p f \cdot \Sigma_p \cdot \nabla_p f^T + \nabla_{\Delta_{rl}} f \cdot \Sigma_\Delta \cdot \nabla_{\Delta_{rl}} f^T. \tag{5.9}$$

The covariance matrix Σ_p is, of course, always given by the $\Sigma_{p'}$ of the previous step, and can thus be calculated after specifying an initial value (e.g., 0).

Using equation (5.7) we can develop the two *Jacobians*, $F_p = \nabla_p f$ and $F_{\Delta_{rl}} = \nabla_{\Delta_{rl}} f$:

$$F_p = \nabla_p f = \nabla_p (f^T) = \left[\frac{\partial f}{\partial x} \; \frac{\partial f}{\partial y} \; \frac{\partial f}{\partial \theta}\right] = \begin{bmatrix} 1 & 0 & -\Delta s \sin(\theta + \Delta\theta/2) \\ 0 & 1 & \Delta s \cos(\theta + \Delta\theta/2) \\ 0 & 0 & 1 \end{bmatrix}, \tag{5.10}$$

$$F_{\Delta_{rl}} = \begin{bmatrix} \frac{1}{2}\cos\left(\theta + \frac{\Delta\theta}{2}\right) - \frac{\Delta s}{2b}\sin\left(\theta + \frac{\Delta\theta}{2}\right) & , \frac{1}{2}\cos\left(\theta + \frac{\Delta\theta}{2}\right) + \frac{\Delta s}{2b}\sin\left(\theta + \frac{\Delta\theta}{2}\right) \\ \frac{1}{2}\sin\left(\theta + \frac{\Delta\theta}{2}\right) + \frac{\Delta s}{2b}\cos\left(\theta + \frac{\Delta\theta}{2}\right) & , \frac{1}{2}\sin\left(\theta + \frac{\Delta\theta}{2}\right) - \frac{\Delta s}{2b}\cos\left(\theta + \frac{\Delta\theta}{2}\right) \\ \frac{1}{b} & -\frac{1}{b} \end{bmatrix} \tag{5.11}$$

The details for arriving at equation (5.11) are

$$F_{\Delta_{rl}} = \nabla_{\Delta_{rl}} f = \left[\frac{\partial f}{\partial \Delta s_r} \; \frac{\partial f}{\partial \Delta s_l}\right] = \ldots \tag{5.12}$$

$$\begin{bmatrix} \frac{\partial \Delta s}{\partial \Delta s_r}\cos\left(\theta + \frac{\Delta\theta}{2}\right) + \frac{\Delta s}{2}-\sin\left(\theta + \frac{\Delta\theta}{2}\right)\frac{\partial \Delta\theta}{\partial \Delta s_r} & , \frac{\partial \Delta s}{\partial \Delta s_l}\cos\left(\theta + \frac{\Delta\theta}{2}\right) + \frac{\Delta s}{2}-\sin\left(\theta + \frac{\Delta\theta}{2}\right)\frac{\partial \Delta\theta}{\partial \Delta s_l} \\ \frac{\partial \Delta s}{\partial \Delta s_r}\sin\left(\theta + \frac{\Delta\theta}{2}\right) + \frac{\Delta s}{2}\cos\left(\theta + \frac{\Delta\theta}{2}\right)\frac{\partial \Delta\theta}{\partial \Delta s_r} & , \frac{\partial \Delta s}{\partial \Delta s_l}\sin\left(\theta + \frac{\Delta\theta}{2}\right) + \frac{\Delta s}{2}\cos\left(\theta + \frac{\Delta\theta}{2}\right)\frac{\partial \Delta\theta}{\partial \Delta s_l} \\ \frac{\partial \Delta\theta}{\partial \Delta s_r} & \frac{\partial \Delta\theta}{\partial \Delta s_l} \end{bmatrix} \tag{5.13}$$

and with

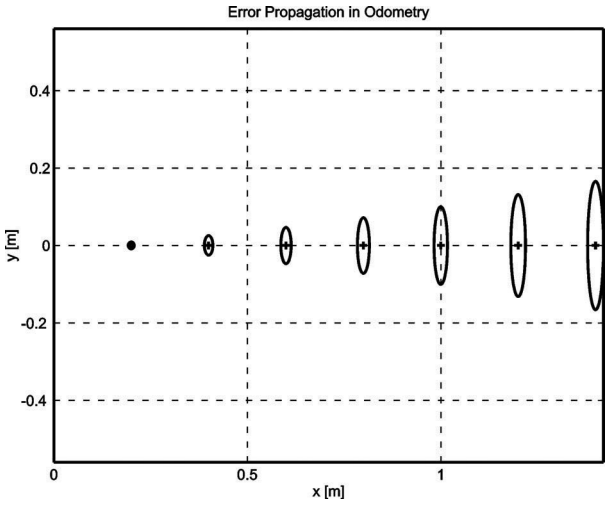


Figure 5.4
Growth of the pose uncertainty for straight-line movement: Note that the uncertainty in y grows much faster than in the direction of movement. This results from the integration of the uncertainty about the robot's orientation. The ellipses drawn around the robot positions represent the uncertainties in the x,y direction (e.g. 3σ). The uncertainty of the orientation θ is not represented in the picture, although its effect can be indirectly observed.

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2} \quad ; \quad \Delta\theta = \frac{\Delta s_r - \Delta s_l}{b} \tag{5.14}$$

$$\frac{\partial \Delta s}{\partial \Delta s_r} = \frac{1}{2} \quad ; \quad \frac{\partial \Delta s}{\partial \Delta s_l} = \frac{1}{2} \quad ; \quad \frac{\partial \Delta\theta}{\partial \Delta s_r} = \frac{1}{b} \quad ; \quad \frac{\partial \Delta\theta}{\partial \Delta s_l} = -\frac{1}{b}, \tag{5.15}$$

we obtain equation (5.11).

Figures 5.4 and 5.5 show typical examples of how the position errors grow with time. The results have been computed using the error model presented earlier.

Once the error model has been established, the error parameters must be specified. One can compensate for deterministic errors properly calibrating the robot. However the error parameters specifying the nondeterministic errors can only be quantified by statistical (repetitive) measurements. A detailed discussion of odometric errors and a method for calibration and quantification of deterministic and nondeterministic errors can be found in [6]. A method for on-the-fly odometry error estimation is presented in [205].

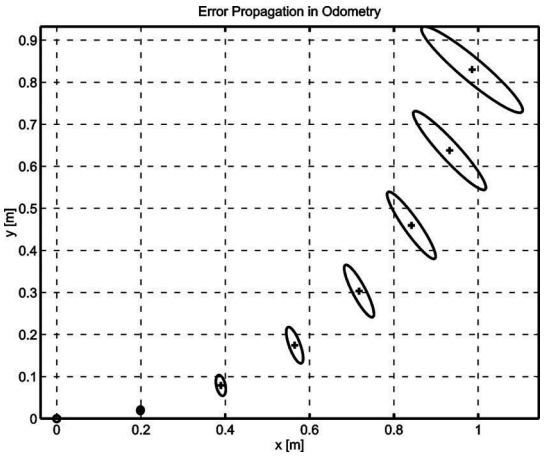


Figure 5.5
Growth of the pose uncertainty for circular movement (r = const): Again, the uncertainty perpendicular to the movement grows much faster than that in the direction of movement. Note that the main axis of the uncertainty ellipse does not remain perpendicular to the direction of movement.

5.3 To Localize or Not to Localize: Localization-Based Navigation Versus Programmed Solutions

Figure 5.6 depicts a standard indoor environment that a mobile robot navigates. Suppose that the mobile robot in question must deliver messages between two specific rooms in this environment: rooms *A* and *B*. In creating a navigation system, it is clear that the mobile robot will need sensors and a motion control system. Sensors are absolutely required to avoid hitting moving obstacles such as humans, and some motion control system is required so that the robot can deliberately move.

It is less evident, however, whether or not this mobile robot will require a *localization system*. Localization may seem mandatory in order to navigate successfully between the two rooms. It is through localizing on a map, after all, that the robot can hope to recover its position and detect when it has arrived at the goal location. It is true that, at the least, the robot must have a way of detecting the goal location. However, explicit localization with reference to a map is not the only strategy that qualifies as a goal detector.

An alternative, espoused by the behavior-based community, suggests that, since sensors and effectors are noisy and information-limited, one should avoid creating a geometric map for localization. Instead, this community suggests designing sets of behaviors that together result in the desired robot motion. Fundamentally, this approach avoids explicit reasoning about localization and position, and thus generally avoids explicit path planning as well.

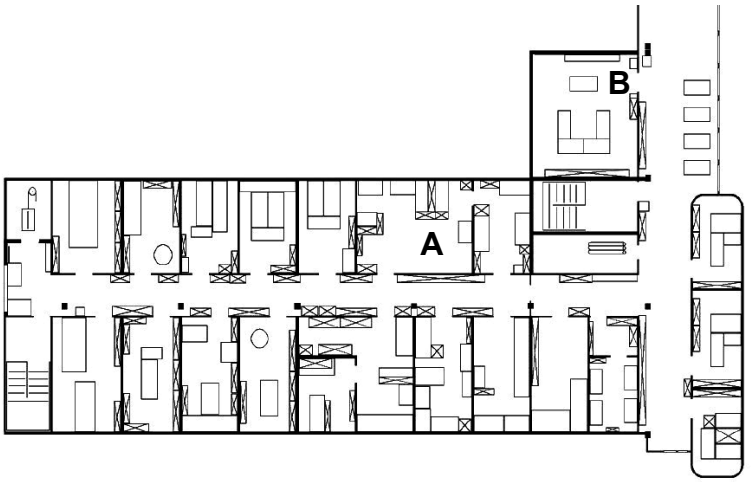


Figure 5.6
A sample environment.

This technique is based on a belief that there exists a procedural solution to the particular navigation problem at hand. For example, in figure 5.6, the behavioralist approach to navigating from room *A* to room *B* might be to design a left-wall following behavior and a detector for room *B* that is triggered by some unique queue in room *B*, such as the color of the carpet. Then the robot can reach room *B* by engaging the left-wall follower with the room *B* detector as the termination condition for the program.

The architecture of this solution to a specific navigation problem is shown in figure 5.7. The key advantage of this method is that, when possible, it may be implemented very quickly for a single environment with a small number of goal positions. It suffers from some disadvantages, however. First, the method does not directly scale to other environments or to larger environments. Often, the navigation code is location-specific, and the same degree of coding and debugging is required to move the robot to a new environment.

Second, the underlying procedures, such as *left-wall-follow,* must be carefully designed to produce the desired behavior. This task may be time-consuming and is heavily dependent on the specific robot hardware and environmental characteristics.

Third, a behavior-based system may have multiple active behaviors at any one time. Even when individual behaviors are tuned to optimize performance, this fusion and rapid switching between multiple behaviors can negate that fine-tuning. Often, the addition of each new incremental behavior forces the robot designer to retune all of the existing behaviors again to ensure that the new interactions with the freshly introduced behavior are all stable.

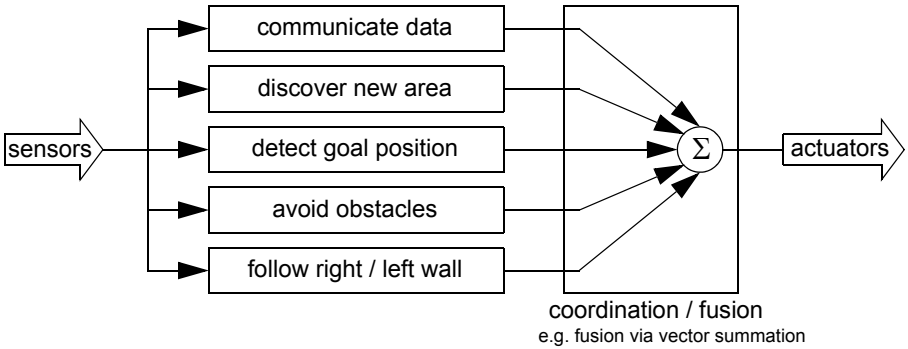


Figure 5.7
An architecture for behavior-based navigation.

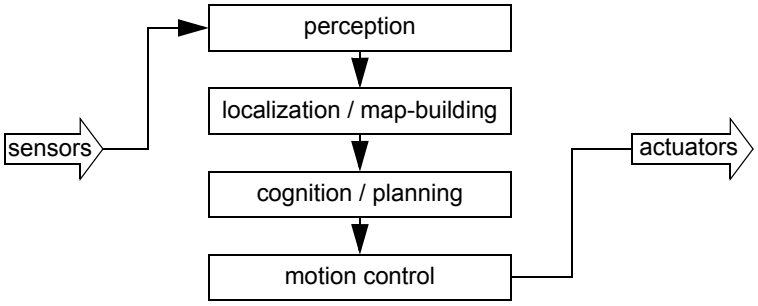


Figure 5.8
An architecture for map-based (or model-based) navigation.

In contrast to the behavior-based approach, the map-based approach includes both *localization* and *cognition* modules (see figure 5.8). In map-based navigation, the robot explicitly attempts to localize by collecting sensor data, then updating some belief about its position with respect to a map of the environment. The key advantages of the map-based approach for navigation are as follows:

- The explicit, map-based concept of position makes the system's belief about position transparently available to the human operators.
- The existence of the map itself represents a medium for communication between human and robot: the human can simply give the robot a new map if the robot goes to a new environment.

- The map, if created by the robot, can be used by humans as well, achieving two uses.

The map-based approach will require more up-front development effort to create a navigating mobile robot. The hope is that the development effort results in an architecture that can successfully map and navigate a variety of environments, thereby amortizing the up-front design cost over time.

Of course the key risk of the map-based approach is that an internal representation, rather than the real world itself, is being constructed and *trusted* by the robot. If that model diverges from reality (i.e., if the map is wrong), then the robot's behavior may be undesirable, even if the raw sensor values of the robot are only transiently incorrect.

In the remainder of this chapter, we focus on a discussion of map-based approaches and, specifically, the localization component of these techniques. These approaches are particularly appropriate for study given their significant recent successes in enabling mobile robots to navigate a variety of environments, from academic research buildings, to factory floors, and to museums around the world.

5.4 Belief Representation

The fundamental issue that differentiates various map-based localization systems is the issue of *representation*. There are two specific concepts that the robot must represent, and each has its own unique possible solutions. The robot must have a representation (a model) of the environment, or a map. What aspects of the environment are contained in this map? At what level of fidelity does the map represent the environment? These are the design questions for *map representation*.

The robot must also have a representation of its belief regarding its position on the map. Does the robot identify a single unique position as its current position, or does it describe its position in terms of a set of possible positions? If multiple possible positions are expressed in a single belief, how are those multiple positions ranked, if at all? These are the design questions for *belief representation*.

Decisions along these two design axes can result in varying levels of architectural complexity, computational complexity, and overall localization accuracy. We begin by discussing belief representation. The first major branch in a taxonomy of belief representation systems differentiates between single-hypothesis and multiple-hypothesis belief systems. The former covers solutions in which the robot postulates its unique position, whereas the latter enables a mobile robot to describe the degree to which it is uncertain about its position. A sampling of different belief and map representations is shown in figure 5.9.

5.4.1 Single-hypothesis belief

The single-hypothesis belief representation is the most direct possible postulation of mobile robot position. Given some environmental map, the robot's belief about position is

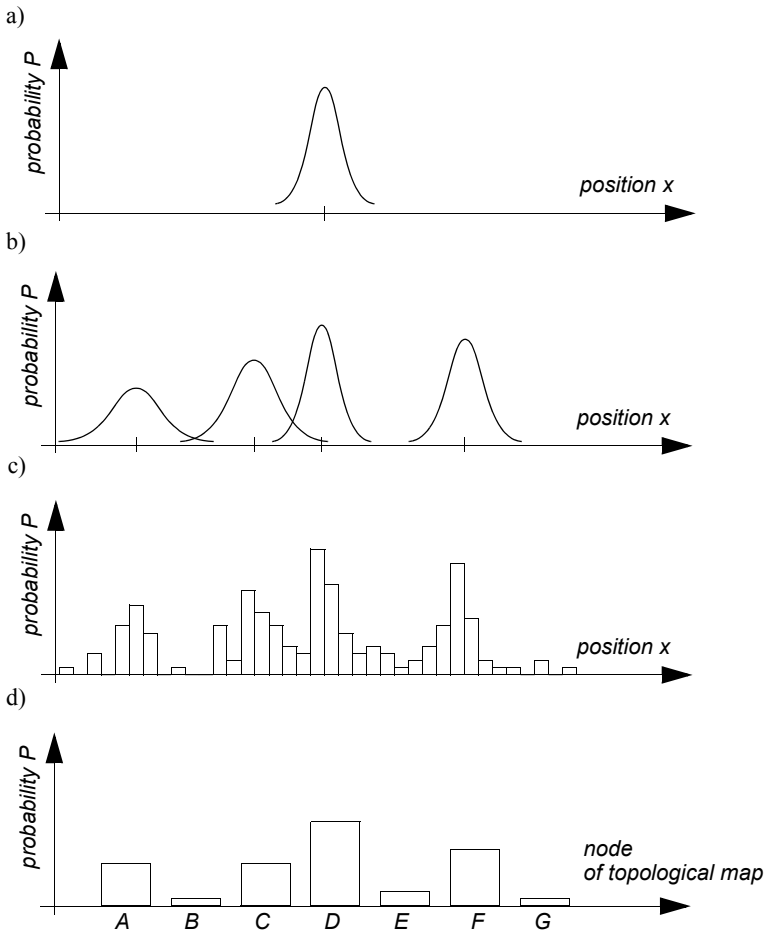


Figure 5.9
Belief representation regarding the robot position (1D) in continuous and discretized (tessellated) maps. (a) Continuous map with single-hypothesis belief, e.g., single Gaussian centered at a single continuous value. (b) Continuous map with multiple-hypothesis belief, e.g;. multiple Gaussians centered at multiple continuous values. (c) Discretized (decomposed) grid map with probability values for all possible robot positions, e.g., Markov approach. (d) Discretized topological map with probability value for all possible nodes (topological robot positions), e.g., Markov approach.

expressed as a single unique point on the map. In figure 5.10, three examples of a single-hypothesis belief are shown using three different map representations of the same actual environment (figure 5.10a). In figure 5.10b, a single point is geometrically annotated as the robot's position in a continuous 2D geometric map. In figure 5.10c, the map is a discrete, tessellated one, and the position is noted at the same level of fidelity as the map cell size. In figure 5.10d, the map is not geometric at all but abstract and topological. In this case, the single hypothesis of position involves identifying a single node i in the topological graph as the robot's position.

The principal advantage of the single-hypothesis representation of position stems from the fact that, given a unique belief, there is no position ambiguity. The unambiguous nature of this representation facilitates decision-making at the robot's cognitive level (e.g., path planning). The robot can simply assume that its belief is correct, and can then select its future actions based on its unique position.

Just as decision making is facilitated by a single-position hypothesis, so updating the robot's belief regarding position is also facilitated, since the single position must be updated by definition to a new, single position. The challenge with this position update approach, which ultimately is the principal disadvantage of single-hypothesis representation, is that robot motion often induces uncertainty due to effector and sensor noise. Therefore, forcing the position update process always to generate a *single* hypothesis of position is challenging and, often, impossible.

5.4.2 Multiple-hypothesis belief

In the case of multiple-hypothesis beliefs regarding position, the robot tracks not just a single possible position but also a possibly infinite set of positions.

In one simple example originating in the work of Jean-Claude Latombe [32, 188], the robot's position is described in terms of a convex polygon positioned in a 2D map of the environment. This multiple-hypothesis representation communicates the set of possible robot positions geometrically, with no preference ordering over the positions. Each point in the map is simply either contained by the polygon and, therefore, in the robot's belief set, or outside the polygon and thereby excluded. Mathematically, the position polygon serves to partition the space of possible robot positions. Such a polygonal representation of the multiple-hypothesis belief can apply to a continuous, geometric map of the environment [57] or, alternatively, to a tessellated, discrete approximation to the continuous environment.

It may be useful, however, to incorporate some ordering on the possible robot positions, capturing the fact that some robot positions are likelier than others. A strategy for representing a continuous multiple-hypothesis belief state along with a preference ordering over possible positions is to model the belief as a mathematical distribution. For example, [87] and [309] notate the robot's position belief using an $\{X, Y\}$ point in the 2D environment

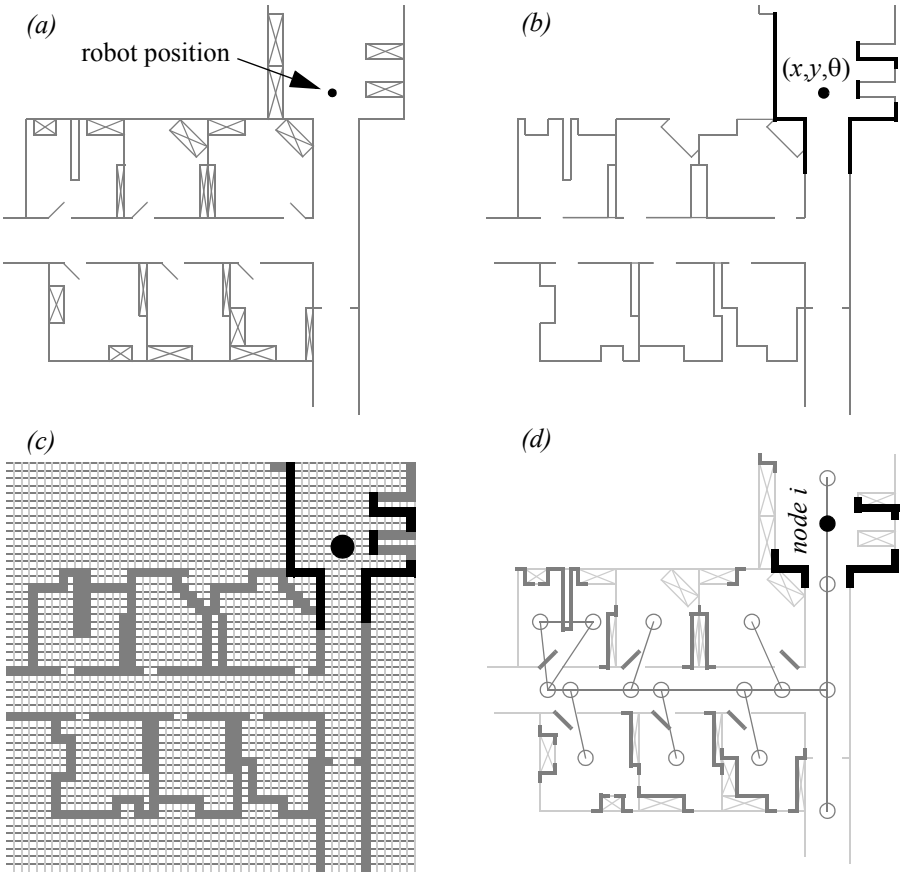


Figure 5.10
Three examples of single hypotheses of position using different map representations: (a) real map with walls, doors and furniture; (b) line-based map → around 100 lines with two parameters; (c) occupancy grid-based map → around 3000 grid cells size 50×50 cm; (d) topological map using line features (Z/S lines) and doors → around 50 features and 18 nodes.

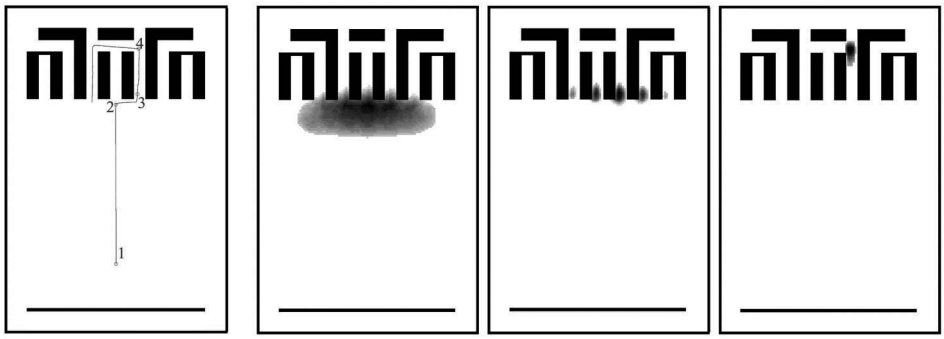


Figure 5.11
Example of multiple-hypothesis tracking (courtesy of W. Burgard [86]). The belief state that is largely distributed becomes very certain after moving to position 4. Note that darker coloring represents higher probability.

as the mean μ plus a standard deviation parameter σ, thereby defining a Gaussian distribution. The intended interpretation is that the distribution at each position represents the probability assigned to the robot being at that location. This representation is particularly amenable to mathematically defined tracking functions, such as the Kalman filter, that are designed to operate efficiently on Gaussian distributions.

An alternative is to represent the set of possible robot positions, not using a single Gaussian probability density function, but using discrete markers for each possible position. In this case, each possible robot position is individually noted along with a confidence or probability parameter (see figure 5.11). In the case of a highly tessellated map this can result in thousands or even tens of thousands of possible robot positions in a single-belief state.

The key advantage of the multiple-hypothesis representation is that the robot can explicitly maintain uncertainty regarding its position. If the robot only acquires partial information regarding position from its sensors and effectors, that information can conceptually be incorporated in an updated belief.

A more subtle advantage of this approach revolves around the robot's ability explicitly to measure its own degree of uncertainty regarding position. This advantage is the key to a class of localization and navigation solutions in which the robot not only reasons about reaching a particular goal but reasons about the future trajectory of its own belief state. For instance, a robot may choose paths that minimize its future position uncertainty. An example of this approach is [306], in which the robot plans a path from point A to point B that takes it near a series of landmarks in order to mitigate localization difficulties. This type of

explicit reasoning about the effect that trajectories will have on the quality of localization requires a multiple-hypothesis representation.

One of the fundamental disadvantages of multiple-hypothesis approaches involves decision making. If the robot represents its position as a region or set of possible positions, then how shall it decide what to do next? Figure 5.11 provides an example. At position 3, the robot's belief state is distributed among five hallways separately. If the goal of the robot is to travel down one particular hallway, then given this belief state, what action should the robot choose?

The challenge occurs because some of the robot's possible positions imply a motion trajectory that is inconsistent with some of its other possible positions. One approach that we will see in the case studies that follow is to assume, for decision-making purposes, that the robot is physically at the most probable location in its belief state, then to choose a path based on that current position. But this approach demands that each possible position have an associated probability.

In general, the right approach to such decision-making problems would be to decide on trajectories that eliminate the ambiguity explicitly. But this leads us to the second major disadvantage of multiple-hypothesis approaches. In the most general case, they can be computationally very expensive. When one reasons in a 3D space of discrete possible positions, the number of possible belief states in the single-hypothesis case is limited to the number of possible positions in the 3D world. Consider this number to be N. When one moves to an arbitrary multiple-hypothesis representation, then the number of possible belief states is the power set of N, which is far larger: 2^N. Thus, explicit reasoning about the possible trajectory of the belief state over time quickly becomes computationally untenable as the size of the environment grows.

There are, however, specific forms of multiple-hypothesis representations that are somewhat more constrained, thereby avoiding the computational explosion while allowing a limited type of multiple-hypothesis belief. For example, if one assumes a Gaussian distribution of probability centered at a single position, then the problem of representation and tracking of belief becomes equivalent to Kalman filtering, a straightforward mathematical process described below. Alternatively, a highly tessellated map representation combined with a limit of ten possible positions in the belief state results in a discrete update cycle that is, at worst, only ten times more computationally expensive than a single-hypothesis belief update. And other ways to cope with the complexity problem, still being precise and computationally cheap, are hybrid metric-topological approaches [314, 317] or multi-Gaussian position estimation [57, 103, 157].

In conclusion, the most critical benefit of the multiple-hypothesis belief state is the ability to maintain a sense of position while explicitly annotating the robot's uncertainty about its own position. This powerful representation has enabled robots with limited sensory

information to navigate robustly in an array of environments, as we shall see in the case studies that follow.

5.5 Map Representation

The problem of representing the environment in which the robot moves is a dual of the problem of representing the robot's possible position or positions. Decisions made regarding the environmental representation can have impact on the choices available for robot position representation. Often the fidelity of the position representation is bounded by the fidelity of the map.

Three fundamental relationships must be understood when choosing a particular map representation:

1. The precision of the map must appropriately match the precision with which the robot needs to achieve its goals.
2. The precision of the map and the type of features represented must match the precision and data types returned by the robot's sensors.
3. The complexity of the map representation has direct impact on the computational complexity of reasoning about mapping, localization, and navigation.

In the following sections, we identify and discuss critical design choices in creating a map representation. Each such choice has great impact on the relationships listed earlier and on the resulting robot localization architecture. As we shall see, the choice of possible map representations is broad. Selecting an appropriate representation requires understanding all of the trade-offs inherent in that choice as well as understanding the specific context in which a particular mobile robot implementation must perform localization. In general, the environmental representation and model can be roughly classified as presented in chapter 4, section 4.4.

5.5.1 Continuous representations

A continuous-valued map is one method for *exact* decomposition of the environment. The position of environmental features can be annotated precisely in continuous space. Mobile robot implementations to date use continuous maps only in 2D representations, as further dimensionality can result in computational explosion.

A common approach is to combine the exactness of a continuous representation with the compactness of the *closed-world assumption*. This means that one assumes that the representation will specify all environmental objects in the map, and that any area in the map that is devoid of objects has no objects in the corresponding portion of the environment. Thus, the total storage needed in the map is proportional to the density of objects in the environment, and a sparse environment can be represented by a low-memory map.

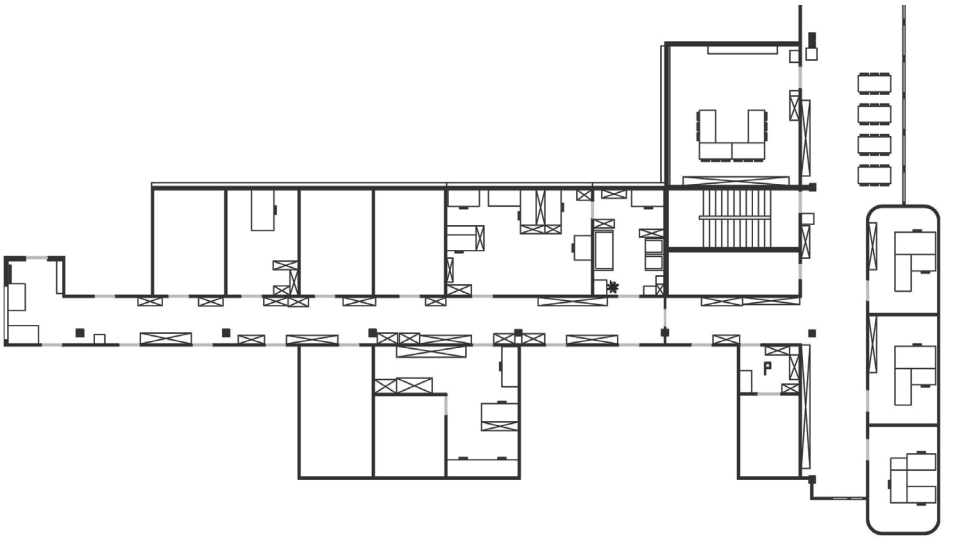


Figure 5.12
A continuous representation using polygons as environmental obstacles.

One example of such a representation, shown in figure 5.12, is a 2D representation in which polygons represent all obstacles in a continuous-valued coordinate space. This is similar to the method used by Latombe [32, 187] and others to represent environments for mobile robot path-planning techniques.

In the case of [32, 187], most of the experiments are in fact simulations run exclusively within the computer's memory. Therefore, no real effort would have been expended to attempt to use sets of polygons to describe a real-world environment, such as a park or office building.

In other work in which real environments must be captured by the maps, one sees a trend toward selectivity and abstraction. The human mapmaker tends to capture on the map, for localization purposes, only objects that can be detected by the robot's sensors and, furthermore, only a subset of the features of real-world objects.

It should be immediately apparent that geometric maps can capably represent the physical locations of objects without referring to their texture, color, elasticity, or any other such secondary features that do not relate directly to position and space. In addition to this level of simplification, a mobile robot map can further reduce memory usage by capturing only aspects of object geometry that are immediately relevant to localization. For example, all objects may be approximated using very simple convex polygons, sacrificing map felicity for the sake of computational speed.

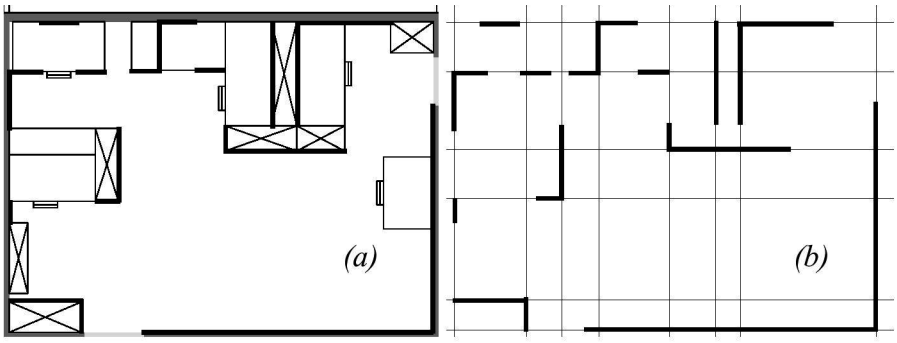


Figure 5.13
Example of a continuous-valued line representation of EPFL. (a) Real map. (b) Representation with a set of infinite lines.

One excellent example involves line extraction. Many indoor mobile robots rely upon laser rangefinding devices to recover distance readings to nearby objects. Such robots can automatically extract best-fit lines from the dense range data provided by thousands of points of laser strikes. Given such a line extraction sensor, an appropriate continuous mapping approach is to populate the map with a set of infinite lines. The continuous nature of the map guarantees that lines can be positioned at arbitrary positions in the plane and at arbitrary angles. The abstraction of real environmental objects such as walls and intersections captures only the information in the map representation that matches the type of information recovered by the mobile robot's rangefinding sensor.

Figure 5.13 shows a map of an indoor environment at EPFL using a continuous line representation. Note that the only environmental features captured by the map are straight lines, such as those found at corners and along walls. This represents not only a sampling of the real world of richer features but also a simplification, for an actual wall may have texture and relief that is not captured by the mapped line.

The impact of continuous map representations on position representation is primarily positive. In the case of single-hypothesis position representation, that position may be specified as any continuous-valued point in the coordinate space, and therefore extremely high accuracy is possible. In the case of multiple-hypothesis position representation, the continuous map enables two types of multiple position representation.

In one case, the possible robot position may be depicted as a geometric shape in the hyperplane, such that the robot is known to be within the bounds of that shape. This is shown in figure 5.33, in which the position of the robot is depicted by an oval bounding area.

Yet, the continuous representation does not disallow representation of position in the form of a discrete set of possible positions. For instance, in [119] the robot position belief state is captured by sampling nine continuous-valued positions from within a region near the robot's best-known position. This algorithm captures, within a continuous space, a discrete sampling of possible robot positions.

In summary, the key advantage of a continuous map representation is the potential for high accuracy and expressiveness with respect to the environmental configuration as well as the robot position within that environment. The danger of a continuous representation is that the map may be computationally costly. But this danger can be tempered by employing abstraction and capturing only the most relevant environmental features. Together with the use of the *closed-world assumption*, these techniques can enable a continuous-valued map to be no more costly, and sometimes even less costly, than a standard discrete representation.

5.5.2 Decomposition strategies

In the previous section, we discussed one method of simplification, in which the continuous map representation contains a set of infinite lines that approximate real-world environmental lines based on a 2D slice of the world. Basically this transformation from the real world to the map representation is a filter that removes all nonstraight data and furthermore extends line segment data into infinite lines that require fewer parameters.

A more dramatic form of simplification is *abstraction*: a general decomposition and selection of environmental features. In this section, we explore decomposition as applied in its more extreme forms to the question of map representation.

Why would one radically decompose the real environment during the design of a map representation? The immediate disadvantage of decomposition and abstraction is the loss of fidelity between the map and the real world. Both qualitatively, in terms of overall structure, and quantitatively, in terms of geometric precision, a highly abstract map does not compare favorably to a high-fidelity map.

Despite this disadvantage, decomposition and abstraction may be useful if the abstraction can be planned carefully so as to capture the relevant, *useful* features of the world while discarding all other features. The advantage of this approach is that the map representation can potentially be minimized. Furthermore, if the decomposition is hierarchical, such as in a pyramid of recursive abstraction, then reasoning and planning with respect to the map representation may be computationally far superior to planning in a fully detailed world model.

A standard, lossless form of *opportunistic decomposition* is termed *exact cell decomposition*. This method, introduced by Latombe [32], achieves decomposition by selecting boundaries between discrete cells based on geometric criticality.

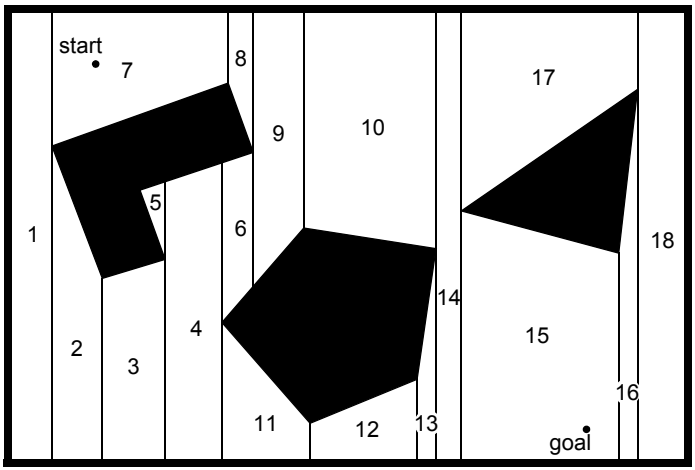


Figure 5.14
Example of exact cell decomposition.

Figure 5.14 depicts an exact decomposition of a planar workspace populated by polygonal obstacles. The map representation tessellates the space into areas of free space. The representation can be extremely compact because each such area is actually stored as a single node, resulting in a total of only eighteen nodes in this example.

The underlying assumption behind this decomposition is that the particular position of a robot within each area of free space does not matter. What matters is the robot's ability to traverse from each area of free space to the adjacent areas. Therefore, as with other representations we will see, the resulting graph captures the adjacency of map locales. If indeed the assumptions are valid and the robot does not care about its precise position within a single area, then this can be an effective representation that nonetheless captures the connectivity of the environment.

Such an exact decomposition is not always appropriate. Exact decomposition is a function of the particular environment obstacles and free space. If this information is expensive to collect or even unknown, then such an approach is not feasible.

An alternative is *fixed decomposition*, in which the world is tessellated, transforming the continuous real environment into a discrete approximation for the map. Such a transformation is demonstrated in figure 5.15, which depicts what happens to obstacle-filled and free areas during this transformation. The key disadvantage of this approach stems from its *inexact* nature. It is possible for narrow passageways to be lost during such a transformation, as shown in figure 5.15. Formally, this means that fixed decomposition is sound but not complete. Yet another approach is adaptive cell decomposition, as presented in figure 5.16.

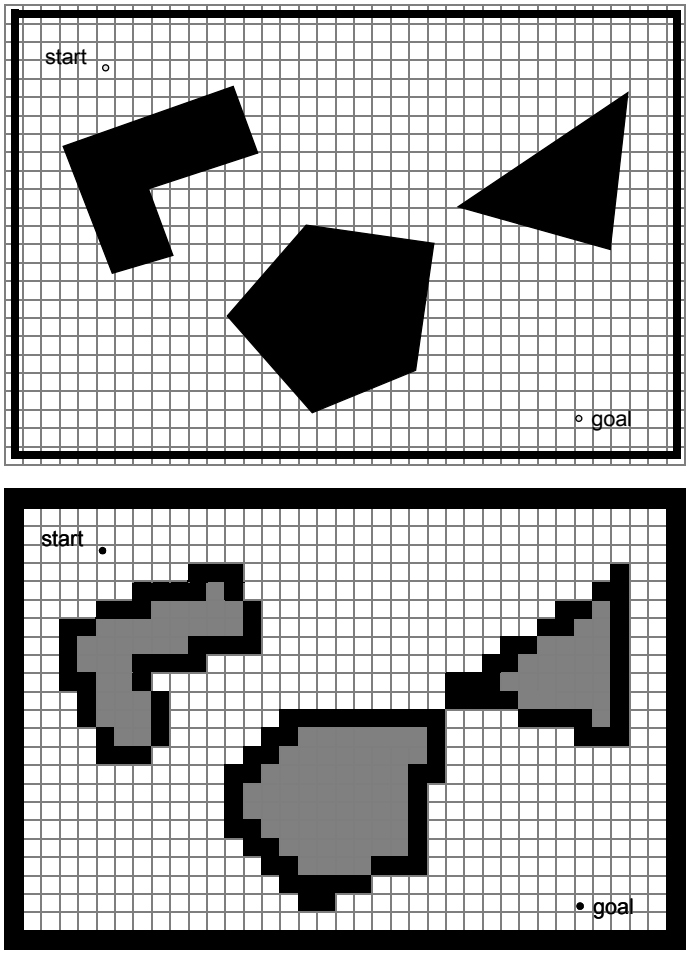


Figure 5.15
Fixed decomposition of the same space (narrow passage disappears).

The concept of fixed decomposition is extremely popular in mobile robotics; it is perhaps the single most common map representation technique currently utilized. One very popular version of fixed decomposition is known as the *occupancy grid* representation [233]. In an occupancy grid, the environment is represented by a discrete grid, where each cell is either filled (part of an obstacle) or empty (part of free space). This method is of particular value when a robot is equipped with range-based sensors because the range values

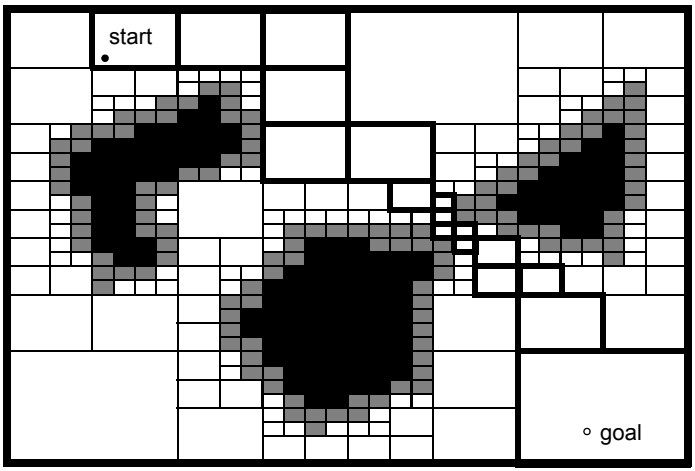


Figure 5.16
Example of adaptive (approximate variable-cell) decomposition of an environment [32]. The rectangle, bounding the free space, is decomposed into four identical rectangles. If the interior of a rectangle lies completely in free space or in the configuration space obstacle, it is not decomposed further. Otherwise, it is recursively decomposed into four rectangles until some predefined resolution is attained. The white cells lie outside the obstacles, the black are inside, and the gray are part of both regions.

of each sensor, combined with the absolute position of the robot, can be used directly to update the filled or empty value of each cell.

In the occupancy grid, each cell may have a counter, whereby the value 0 indicates that the cell has not been "hit" by any ranging measurements and, therefore, it is likely free space. As the number of ranging strikes increases, the cell's value is incremented and, above a certain threshold, the cell is deemed to be an obstacle. The values of cells are commonly discounted when a ranging strike travels *through* the cell, striking a further cell. By also discounting the values of cells over time, both hysteresis and the possibility of transient obstacles can be represented using this occupancy grid approach. Figure 5.17 depicts an occupancy grid representation in which the darkness of each cell is proportional to the value of its counter. One commercial robot that uses a standard occupancy grid for mapping and navigation is the Cye robot [342].

There remain two main disadvantages of the occupancy grid approach. First, the size of the map in robot memory grows with the size of the environment, and if a small cell size is used, this size can quickly become untenable. This occupancy grid approach is not compatible with the *closed-world assumption*, which enabled continuous representations to have potentially very small memory requirements in large, sparse environments. In contrast, the

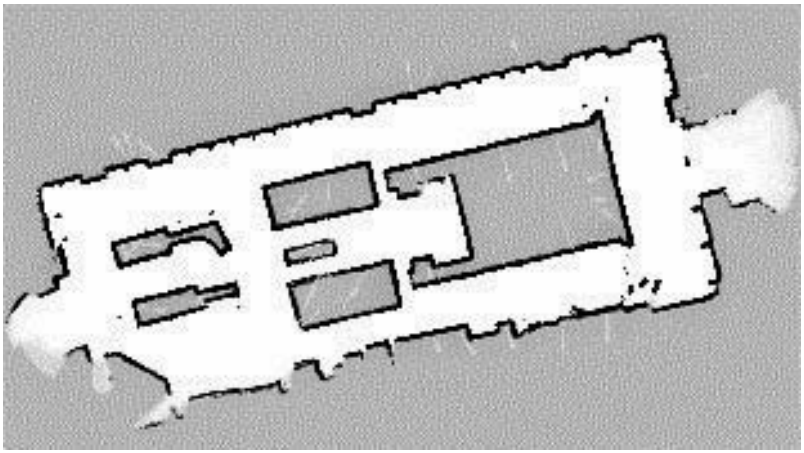


Figure 5.17
Example of an occupancy grid map representation. Courtesy of S. Thrun [314].

occupancy grid must have memory set aside for every cell in the matrix. Furthermore, any fixed decomposition method such as this imposes a geometric grid on the world *a priori*, regardless of the environmental details. This can be inappropriate in cases where geometry is not the most salient feature of the environment.

For these reasons, an alternative, called *topological* decomposition, has been the subject of some exploration in mobile robotics. Topological approaches avoid direct measurement of geometric environmental qualities, instead concentrating on characteristics of the environment that are most relevant to the robot for localization.

Formally, a topological representation is a graph that specifies two things: *nodes* and the *connectivity* between those nodes. Insofar as a topological representation is intended for the use of a mobile robot, nodes are used to denote areas in the world and arcs are used to denote adjacency of pairs of nodes. When an arc connects two nodes, then the robot can traverse from one node to the other without requiring traversal of any other intermediary node.

Adjacency is clearly at the heart of the topological approach, just as adjacency in a cell decomposition representation maps to geometric adjacency in the real world. However, the topological approach diverges in that the nodes are not of fixed size or even specifications of free space. Instead, nodes document an area based on any sensor discriminant such that the robot can recognize entry and exit of the node.

Figure 5.18 depicts a topological representation of a set of hallways and offices in an indoor environment. In this case, the robot is assumed to have an intersection detector, perhaps using sonar and vision to find intersections between halls and between halls and

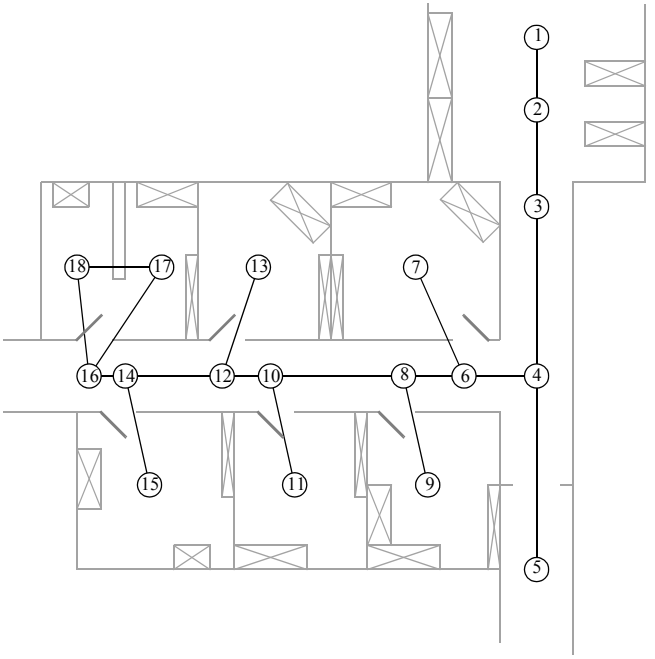


Figure 5.18
A topological representation of an indoor office area.

rooms. Note that nodes capture geometric space, and arcs in this representation simply represent connectivity.

Another example of topological representation is the work of Simhon and Dudek [290], in which the goal is to create a mobile robot that can capture the most interesting aspects of an area for human consumption. The nodes in their representation are visually striking locales rather than route intersections.

In order to navigate using a topological map robustly, a robot must satisfy two constraints. First, it must have a means for detecting its current position in terms of the nodes of the topological graph. Second, it must have a means for traveling between nodes using robot motion. The node sizes and particular dimensions must be optimized to match the sensory discrimination of the mobile robot hardware. This ability to "tune" the representation to the robot's particular sensors can be an important advantage of the topological approach. However, as the map representation drifts further away from true geometry, the expressiveness of the representation for accurately and precisely describing a robot position is lost. Therein lies the compromise between the discrete cell-based map representations and the topological representations. Interestingly, the continuous map representation has

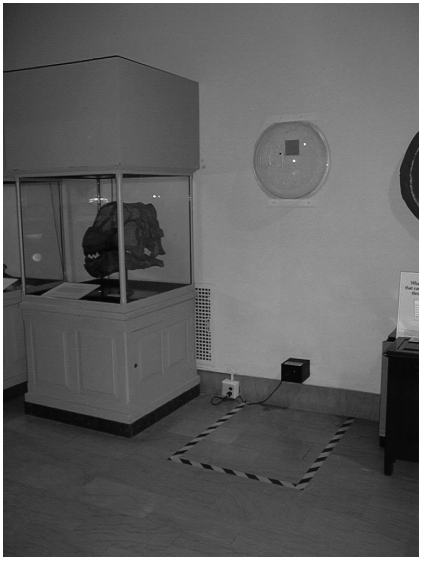


Figure 5.19
An artificial landmark used by Chips during autonomous docking.

the potential to be both compact, like a topological representation, and precise, as with all direct geometric representations.

Yet, a chief motivation of the topological approach is that the environment may contain important nongeometric features—features that have no ranging relevance but are useful for localization. In chapter 4 we described such whole-image vision-based features.

In contrast to these whole-image feature extractors, often spatially localized landmarks are artificially placed in an environment to impose a particular visual-topological connectivity upon the environment. In effect, the artificial landmark can impose artificial structure. Examples of working systems operating with this landmark-based strategy have also demonstrated success. Latombe's landmark-based navigation research [188] has been implemented on real-world indoor mobile robots that employ paper landmarks attached to the ceiling as the locally observable features. Chips, the museum robot, is another robot that uses man-made landmarks to obviate the localization problem. In this case, a bright pink square serves as a landmark with dimensions and color signature that would be hard to accidentally reproduce in a museum environment [251]. One such museum landmark is shown in figure 5.19.

In summary, range is clearly not the only measurable and useful environmental value for a mobile robot. This is particularly true with the advent of color vision, as well as laser

rangefinding, which provides reflectance information in addition to range information. Choosing a map representation for a particular mobile robot requires, first, understanding the sensors available on the mobile robot, and, second, understanding the mobile robot's functional requirements (e.g., required goal precision and accuracy).

5.5.3 State of the art: Current challenges in map representation

The previous sections describe major design decisions in regard to map representation choices. There are, however, fundamental real-world features that mobile robot map representations do not yet represent well. These continue to be the subject of open research, and several such challenges are described here.

The real world is dynamic. As mobile robots come to inhabit the same spaces as humans, they will encounter moving people, cars, strollers, and the transient obstacles placed and moved by humans as they go about their activities. This is particularly true when one considers the home environment with which domestic robots will someday need to contend.

The map representations described earlier do not, in general, have explicit facilities for identifying and distinguishing between permanent obstacles (e.g., walls, doorways, etc.) and transient obstacles (e.g., humans, shipping packages, etc.). The current state of the art in terms of mobile robot sensors is partly to blame for this shortcoming. Although vision research is rapidly advancing, robust sensors that discriminate between moving animals and static structures *from a moving reference frame* are not yet available. Furthermore, estimating the motion vector of transient objects remains a research problem.

Usually, the assumption behind the preceding map representations is that all objects on the map are effectively static. Partial success can be achieved by discounting mapped objects over time. For example, occupancy grid techniques can be more robust to dynamic settings by introducing temporal discounting, effectively treating transient obstacles as noise. The more challenging process of map creation is particularly fragile to environmental dynamics; most mapping techniques generally require that the environment be free of moving objects during the mapping process. One exception to this limitation involves topological representations. Because precise geometry is not important, transient objects have little effect on the mapping or localization process, subject to the critical constraint that the transient objects must not change the topological connectivity of the environment. Still, neither the occupancy grid representation nor a topological approach is actively recognizing and representing transient objects as distinct from both sensor error and permanent map features.

As vision sensing provides more robust and more informative content regarding the transience and motion details of objects in the world, mobile roboticists will in time propose representations that make use of that information. A classic example involves occlusion by human crowds. Museum tour guide robots generally suffer from an extreme amount of occlusion. If the robot's sensing suite is located along the robot's body, then the robot is

effectively blind when a group of human visitors completely surround the robot. This is because its map contains only environmental features that are, at that point, fully hidden from the robot's sensors by the wall of people. In the best case, the robot should recognize its occlusion and make no effort to localize using these invalid sensor readings. In the worst case, the robot will localize with the fully occluded data, and will update its location incorrectly. A vision sensor that can discriminate the local conditions of the robot (e.g,. we are surrounded by people) can help eliminate this error mode.

A second open challenge in mobile robot localization involves the traversal of open spaces. Existing localization techniques generally depend on local measures such as range, thereby demanding environments that are somewhat densely filled with objects that the sensors can detect and measure. Wide-open spaces such as parking lots, fields of grass, and indoor atriums such as those found in convention centers pose a difficulty for such systems because of their relative sparseness. Indeed, when populated with humans, the challenge is exacerbated because any mapped objects are almost certain to be occluded from view by the people.

Once again, more recent technologies provide some hope of overcoming these limitations. Both vision and state-of-the-art laser rangefinding devices offer outdoor performance with ranges of up to a hundred meters and more. Of course, GPS performs even better. Such long-range sensing may be required for robots to localize using distant features.

This trend teases out a hidden assumption underlying most topological map representations. Usually, topological representations make assumptions regarding spatial locality: a node contains objects and features that are themselves within that node. The process of map creation thus involves making nodes that are, in their own self-contained way, recognizable by virtue of the objects contained within the node. Therefore, in an indoor environment, each room can be a separate node, and this is reasonable because each room will have a layout and a set of belongings that are unique to that room.

However, consider the outdoor world of a wide-open park. Where should a single node end and the next node begin? The answer is unclear because objects that are far away from the current node, or position, can yield information for the localization process. For example, the hump of a hill at the horizon, the position of a river in the valley, and the trajectory of the sun all are nonlocal features that have great bearing on one's ability to infer current position. The spatial locality assumption is violated and, instead, replaced by a visibility criterion: the node or cell may need a mechanism for representing objects that are measurable and visible from that cell. Once again, as sensors improve and, in this case, as outdoor locomotion mechanisms improve, there will be greater urgency to solve problems associated with localization in wide-open settings, with and without GPS-type global localization sensors.

We end this section with one final open challenge that represents one of the fundamental academic research questions of robotics: sensor fusion. A variety of measurement types are

possible using off-the-shelf robot sensors, including heat, range, acoustic and light-based reflectivity, color, texture, friction, and so on. Sensor fusion is a research topic closely related to map representation. Just as a map must embody an environment in sufficient detail for a robot to perform localization and reasoning, sensor fusion demands a representation of the world that is sufficiently general and expressive that a variety of sensor types can have their data correlated appropriately, strengthening the resulting percepts well beyond that of any individual sensor's readings.

Perhaps the only general implementation of sensor fusion to date is that of neural network classifier. Using this technique, any number and any type of sensor values may be jointly combined in a network that will use whatever means necessary to optimize its classification accuracy. For the mobile robot that must use a human-readable internal map representation, no equally general sensor fusion scheme has yet been born. It is reasonable to expect that, when the sensor fusion problem is solved, integration of a large number of disparate sensor types may easily result in sufficient discriminatory power for robots to achieve real-world navigation, even in wide-open and dynamic circumstances such as a public square filled with people.

5.6 Probabilistic Map-Based Localization

5.6.1 Introduction

As stated earlier, multiple-hypothesis position representation is advantageous because the robot can explicitly track its own beliefs regarding its possible positions in the environment. Ideally, the robot's *belief state* will change, over time, as is consistent with its motor outputs and perceptual inputs. One geometric approach to multiple-hypothesis representation, mentioned earlier, involves identifying the possible positions of the robot by specifying a polygon in the environmental representation [187]. This method does not provide any indication of the relative chances between various possible robot positions.

Probabilistic map-based localization techniques differ from this because they explicitly identify probabilities with the possible robot positions, and for this reason these methods have been the focus of recent research. The reason probabilistic approaches to mobile robot localization have been developed is that the data coming from the robot sensors are affected by measurement errors, and therefore we can only compute the probability that the robot is in a given configuration. This new area of research is called *probabilistic robotics* [51]. The key idea in probabilistic robotics is to represent uncertainty using probability theory. Stating this in different words, instead of giving a single best estimate of the current robot configuration, probabilistic robotics represents the robot configuration as a probability distribution over the all possible robot poses. By doing so, ambiguity and degree of belief are represented using calculus of probability theory. The success of this theory applied to

the mobile robot localization problem comes from the fact that probabilistic algorithms outperform alternative techniques in many real world applications.

In the following sections, we present two classes of probabilistic map-based localization. The first class, *Markov localization*, uses an explicitly specified probability distribution across all possible robot positions. The second method, *Kalman filter localization*, uses a Gaussian probability density representation of robot position. Unlike Markov localization, Kalman filter localization does not independently consider each possible pose in the robot's configuration space. Interestingly, the Kalman filter localization process results from the Markov localization axioms if the robot's position uncertainty is assumed to have a Gaussian form.

5.6.2 The robot localization problem

Before discussing each method in detail, we present the general robot localization problem and solution strategy. Consider a mobile robot moving in a known environment. As it starts to move, say from a precisely known location, it can keep track of its motion using odometry. Due to odometry uncertainty, after some movement the robot will become very uncertain about its position (see section 5.2.4). To keep position uncertainty from growing unbounded, the robot must localize itself in relation to its environment map. To localize, the robot might use its on-board exteroceptive sensors (e.g., ultrasonic, laser, vision sensors) to make observations of its environment. The information provided by the robot's odometry, plus the information provided by such exteroceptive observations, can be combined to enable the robot to localize as well as possible with respect to its map. The processes of updating based on proprioceptive sensor values and exteroceptive sensor values are often separated logically, leading to a general process for the robot position update that comprises two steps, which are called *prediction* (or *action*) *update* and *perception* (or *measurement*, or *correction*) *update*.

- During the *prediction* (or *action*) *update* the robot uses its proprioceptive sensors to estimate its configuration; for example, the robot estimates its motion using the encoders. In this phase, the uncertainty about the robot configuration increases due to the integration of the odometric error over time. In figure 5.20a, we illustrate this process for a robot moving in a one-dimensional environment.
- During the *perception* (or *measurement*, or *correction*) *update* the robot uses the information from its exteroceptive sensors to correct the position estimated during the prediction phase; for example, the robot uses a rangefinder to measure its current distance from a wall and corrects accordingly the position estimated during the prediction phase. During the perception phase, the uncertainty of the robot configuration shrinks (figure 5.20b).

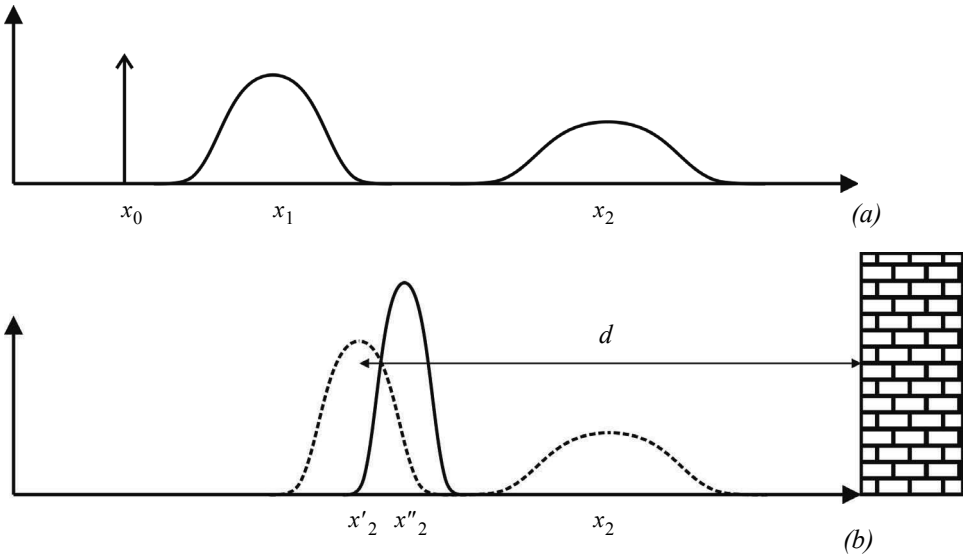


Figure 5.20 In probabilistic robotics, the *beliefs* about the robot configuration are represented as probability density functions. Note, in general the distributions can be any function. Not necessarily Gaussian. Only the Kalman filter assumes Gaussian distributions. (a) Prediction phase: the start position x_0 is assumed to be known, and therefore the probability density function is a *Dirac delta* function. As the robot starts moving, its uncertainty grows due to the odometric error, which accumulates over time. (b) Perception phase: the robot uses its exteroceptive sensor (e.g., a rangefinder) to measure the distance d from the right wall, and computes the position x'_2, which is in conflict with the current position x_2 estimated in the action phase. The perception update corrects the new position to x''_2 and as consequence its uncertainty shrinks (solid line).

In general, the prediction update contributes to increase the uncertainty to the robot's belief about position: encoders have error, and therefore motion is somewhat nondeterministic. By contrast, perception update generally refines the belief state (i.e., the uncertainty shrinks). Sensor measurements, when compared to the robot's environmental model, tend to provide clues regarding the robot's possible position.

In the next sections, we will describe two different methods of probabilistic map-based localization, which are the Markov localization and Kalman filter localization. In the case of Markov localization, the robot's belief state can be represented with any *arbitrary* probability density function. The prediction and perception phases update the probability of every possible robot pose.

In the case of Kalman filter localization, the robot's belief state is conversely represented using a single *Gaussian* probability density function (page 112), and thus it retains just a μ and σ parameterization of the robot's belief about position with respect to the

map. Updating the parameters of the Gaussian distribution is all that is required. This fundamental difference in the representation of belief state leads to the following advantages and disadvantages of the two methods, as presented in [143]:

- Markov localization allows for localization starting from *any unknown* position and can thus recover from ambiguous situations because the robot can track multiple, completely disparate possible positions. However, to update the probability of all positions within the whole state space at any time requires a discrete representation of the space, such as a geometric grid or a topological graph (see section 5.5.2). The required memory and computational power can thus limit precision and map size.
- Kalman filter localization tracks the robot from an *initially known* position and is inherently both precise and efficient. In particular, Kalman filter localization can be used in continuous world representations. However, if the uncertainty of the robot becomes too large (e.g., due to a robot collision with an object) and thus not truly unimodal, the Kalman filter can fail to capture the multitude of possible robot positions and can become irrevocably lost.

In recent research projects, improvements are achieved or proposed by either only updating the state space of interest within the Markov approach [86] or by tracking multiple hypotheses with Kalman filters [57, 231], or by combining both methods to create a hybrid localization system [143, 317]. In the next sections we will review each approach in detail but first we will recall some concept of probability theory. For an in-depth study of probability theory, we refer the reader to [41].

5.6.3 Basic concepts of probability theory

Let X denote a random variable and x a specific value that X might assume. A typical example is dice rolling, where X can take on any value between 1 and 6. We denote with

$$p(X = x) \tag{5.16}$$

the probability that the random variable X has value x. For example, the result of a die roll is characterized by

$$p(X = 1) = p(X = 2) = p(X = 3) = p(X = 4) = p(X = 5) = p(X = 6) = \frac{1}{6}. \tag{5.17}$$

From now on, to simplify the notation we will omit explicit mention of the random variable, and will instead use the simple abbreviation $p(x)$.

In continuous spaces, random variables can take on a continuum of values and in this case we will talk about *probability density functions* (PDFs).

Both discrete and continuous probabilities integrate to one, therefore:

$$\sum_x p(x) = 1 \text{ (for discrete probabilities),} \tag{5.18}$$

$$\int_x p(x)dx = 1 \text{ (for continuous probabilities).} \tag{5.19}$$

Furthermore, probabilities are always non-negative, that is, $p(x) \geq 0$.

Gaussian distribution. As we have seen (page 112), a common probability density function is the Gaussian distribution

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{5.20}$$

also called normal distribution, which is commonly abbreviated with

$$p(x) = N(x, \sigma^2) \tag{5.21}$$

where μ and σ^2 specify the mean and the variance of the random variable x. Note that in this case the random variable x is a scalar. However, when x is a k-dimensional vector, we will have a multivariate normal distribution characterized by a density function of the following form:

$$p(x) = \frac{1}{(2\pi)^{k/2} det(\Sigma)^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right), \tag{5.22}$$

where μ is the mean vector and Σ is a *positive semidefinite* and *symmetric* matrix called *covariance matrix*.

Joint distribution. The *joint distribution* of two random variables X and Y is given by $p(x, y)$, which describes the probability that the random variable X takes on the value x *and* that Y takes on the value y. If X and Y are *independent*, we have

$$p(x, y) = p(x)p(y) \tag{5.23}$$

Conditional probability. The *conditional probability* describes the probability that the random variable X takes on the value x *conditioned* on the knowledge that for sure Y takes on the value y. As an example, to compute the probability that the result of a die roll is 2 conditioned on the fact that we know with a probability of 100% that the result will be an even number (for example, we have a special die). Conditional probability is denoted with $p(x|y)$ and, if $p(y) > 0$, is defined as

$$p(x|y) = \frac{p(x, y)}{p(y)}. \tag{5.24}$$

If X and Y are independent, we have

$$p(x|y) = \frac{p(x)p(y)}{p(y)} = p(x). \tag{5.25}$$

In other words, if X and Y are independent, the knowledge of Y does not provide any useful information about the value of X.

Theorem of total probability. The *theorem of total probability* originates from the axioms of probability theory and is written as:

$$p(x) = \sum_{y} p(x|y)p(y) \text{ (for discrete probabilities),} \tag{5.26}$$

$$p(x) = \int_{y} p(x|y)p(y)dy \text{ (for continuous probabilities).} \tag{5.27}$$

As we will see in the next section, the theorem of total probability is used by both Markov and Kalman-filter localization algorithms during the prediction update.

Bayes rule. The *Bayes rule* relates the conditional probability $p(x|y)$ to its inverse $p(y|x)$. Under the condition that $p(y) > 0$, the Bayes rule is written as:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}. \tag{5.28}$$

As we will see in the next section, the Bayes rule is used by both Markov and Kalman-filter localization algorithms during the measurement update.

Prior and posterior probability. A prior probability distribution of a random variable x is the probability distribution $p(x)$ that we have before incorporating the data y. For example, in mobile robotics the prior could be the probability distribution of the robot location over the whole space *prior* taking into account any sensor measurement. The probability $p(x|y)$, which is computed *after* incorporating the data is referred to as *posterior probability distribution*. As shown by equation (5.28), the Bayes rule provides a convenient way to compute the posterior probability $p(x|y)$ using the "inverse" conditional probability $p(y|x)$ and the prior probability $p(x)$. Using the preceding example from mobile robotics, if we want to know the probability $p(x|y)$ of the robot of occupying a specific position x after reading the sensor data y, we just need to multiply the conditional probability $p(y|x)$ of observing those measurements if the robot was at that position by the probability $p(x)$ of the robot to be there before reading the sensor. The result will then have to be divided by a certain normalization factor $p(y)$. The concept of prior and posterior probability and the benefit of the Bayes rule will be anyway clarified later on.

It is important to notice that the denominator of the Bayes rule, $p(y)$, does not depend on x. For this reason, in the Bayes rule the factor $p(y)^{-1}$ is usually written as a *normalization* factor, generically denoted η, which can be determined straightforwardly by remembering that the integral of a probability distribution is always 1. This way, the Bayes rule can be written as:

$$p(x|y) = \eta p(y|x)p(x) . \tag{5.29}$$

5.6.4 Terminology

Path, input, and observations. Here we introduce the terminology that will be used throughout the next sections. The terminology and the notation are the same as introduced in [51]. Let t denote the time and x_t denote the robot location. For planar motion, $x_t = [x, y, \theta]^T$ is a three-dimensional vector consisting of the robot position and orientation. The robot *path* is given as

$$X_T = \{x_0, x_1, x_2, \ldots, x_T\} , \tag{5.30}$$

where T could also be infinite.

Let u_t denote the proprioceptive sensor readings at time t. This can be, for instance, the reading of the robot's wheel encoders or IMU, or the control input given to the motors (e.g.

the speed).[23] If we assume that the robot receives exactly one data at each point in time, the sequence of proprioceptive data can be written as:

$$U_T = \{u_0, u_1, u_2, \ldots, u_T\} \tag{5.31}$$

In the absence of noise, we could obviously recover all the past robot locations X_T by integrating over U_T provided that the robot initial location x_0 is known. However, because of noise, the integrated path unavoidably drifts from the ground truth and therefore additional means are needed to reduce this drift. As we will see in the next sections, the exteroceptive sensor readings allow us to cancel the drift by keeping it bounded.

Exteroceptive sensors such as cameras, lasers, or ultrasonic rangefinders allow the robot to perceive the environment. The output of these sensors could be directly the 3D coordinates (in meters) of points, lines, or planes in the *sensor reference frame*. But they could also be simply the coordinates (in pixels) of point features (like corners) or lines (like doors). Whatever are the outputs returned by the sensor, they are typically referred to as *observations*, *measurement data*, or *exteroceptive sensor readings*. If we assume that the robot takes exactly one measurement at each point in time, the sequence of measurements is given as

$$Z_T = \{z_0, z_1, z_2, \ldots, z_T\}\ , \tag{5.32}$$

where it is important to remark that the coordinates of each observation are expressed in the *sensor reference frame* attached to the robot.

Finally, let M denote the *true* map of the environment and let us assume that the environment consists of two-dimensional point landmarks (or 2D infinite lines). In this case, the map is a vector of size 2*n*, where *n* is the number of features in the world; therefore,

$$M = \{m_0, m_1, m_2, \ldots, m_{n-1}\}\ , \tag{5.33}$$

where m_i, $i = 0 \ldots n-1$, are vectors representing the 2D coordinates of the landmarks in the *world reference frame* (e.g., the 2D position of the point or the position and orientation of the line). The same discussion can be extended to the 3D case by incorporating the position of points, or location and orientation of lines and planes. Finally, we assume that the world is static, and therefore the environment map M is time-invariant.

23. To facilitate the language, in the remainder we will always refer to u_t as the *control input* but keep in mind that in general it can also represent the proprioceptive sensor readings.

Belief distributions. In section 5.4, we already introduced the concept of *belief representation*. Here we review this concept in terms of probability. In general, a robot cannot measure its *true* state (pose) directly, not even with GPS. What it can know is only the best estimate of its pose (for example $x_t = [3.0, 5.1, 180°]^T$) based on its sensors' data. Therefore, the knowledge the robot has about its state can only be inferred by the data. The best guess about the robot state is called *belief*. In probabilistic robotics, beliefs are represented through conditional probability distributions, and therefore they are posterior probabilities of the state variables conditioned on the available data. If we denote the *belief* over a state variable x_t by $bel(x_t)$, we can write

$$bel(x_t) = p(x_t | z_{1 \to t}, u_{1 \to t}), \tag{5.34}$$

where the posterior $p(x_t | z_{1 \to t}, u_{1 \to t})$ represents the probability of the robot of being at x_t given all its past observations $z_{1 \to t}$ and all its past control inputs $u_{1 \to t}$.

In Markov and Kalman localization, we will also often refer to the belief calculated *before* incorporating the new observation z_t just after the control input u_t. Such a posterior will be denoted as:

$$\overline{bel}(x_t) = p(x_t | z_{1 \to t-1}, u_{1 \to t}). \tag{5.35}$$

This probability distribution $\overline{bel}(x_t)$, which is computed just before including the new observation z_t, is often called *prediction* (or *action*) update, meaning that the current robot pose (belief) is only *predicted* on the basis of the motion control and the previous observations. It is also called *action* because during this phase the robot physically moves. Conversely, the calculation of $bel(x_t)$ from $\overline{bel}(x_t)$ is often called *correction* (or *perception*, or *measurement*) update because the robot pose is corrected after the observation.

5.6.5 The ingredients of probabilistic map-based localization

In order to solve the robot localization problem, the following information is required.

1. The initial probability distribution $bel(x_0)$. In the case the initial robot location is unknown, the initial belief $bel(x_0)$ is a uniform distribution over all poses. Conversely, if the location is perfectly known the initial belief is a *Dirac delta* function. As we will see, the Markov approach allows us to select any arbitrary initial distribution, while in the Kalman filter approach only a Gaussian distribution is allowed.

2. Map of the environment. The environment map $M = \{m_0, m_1, m_2, \ldots, m_n\}$ must be known. If the map is not known *a priori*, then the robot needs to build a map of the environment. Automatic map building will be covered in section 5.8.

3. Data. For localizing, the robot obviously needs to use data from its proprioceptive and exteroceptive sensors. We denote with z_t the current reading from the exteroceptive sensor. z_t is also called the observation. With u_t, we denote instead the reading from the proprioceptive sensor or the control input. For a differential drive robot, u_t could, for example, represent the encoder readings of the right and left wheel, and, therefore, we would write $u_t = [\Delta S_r, \Delta S_l]^T$.

4. The probabilistic motion model. The probabilistic motion model is derived from the kinematics of the robot. In the noise-free case, the robot current location x_t can be computed as a function f of the previous location x_{t-1} and the encoder readings u_t, that is:

$$x_t = f(x_{t-1}, u_t) . \tag{5.36}$$

For example, for a differential drive robot f is simply the odometric-position-update formula (5.7).

To derive the probabilistic motion model, we need to model the error distributions over x_{t-1} and u_t and then use f to compute the error distribution over x_t. As we have seen in section 5.2.4, if we assume that both x_{t-1} and u_t are normally distributed, uncorrelated, and that f can be approximated by its first-order Taylor expansion, then the error distribution over x_t can be modeled as a multivariate Gaussian with mean value $f(x_{t-1}, u_t)$ and covariance matrix specified by the error propagation law (5.9).

5. The probabilistic measurement model. This is derived directly from the exteroceptive sensor model, e.g., the error model of the laser, the sonar, or the camera. As we have seen in chapter 4, laser and sonars provide range measurements, while a single camera provides bearing measurements. Because these measurements are always noisy, in order to characterize the sensor model we have to define the exact, noise-free measurement function. The measurement function h clearly depends on the environment map M and on the robot location x_t, therefore we can write:

$$z_t = h(x_t, M) . \tag{5.37}$$

The measurement function h is typically a change of coordinates from the world frame to the sensor reference frame attached to the robot. For instance, in the example shown in figure 5.20 the robot uses a rangefinder to measure its distance d from the right wall. Here, d is the observation, and therefore $z_t = d$. The map M is represented by a single feature m (i.e., the wall), which we assume is at coordinate $m = 10$ (we assume one coordinate because we suppose that the robot moves in a one-dimensional environment). The measurement function in this simple example is therefore:

$$h(x_t, M) = 10 - x_t . \tag{5.38}$$

In the more general case, h is a change of coordinates from the world frame to the sensor reference frame attached to the robot.

To derive the probabilistic measurement model, we just need to add a noise term to the measurement function such that the probability distribution $p(z_t|x_t, M)$ peaks at the noise-free value $h(x_t, M)$. For example, if we assume Gaussian noise we can write

$$p(z_t|x_t, M) = N(h(x_t, M), R_t) , \tag{5.39}$$

where, more generally, N denotes a multivariate normal distribution with mean value $h(x_t, M)$ and noise covariance matrix R_t.

5.6.6 Classification of localization problems

Before proceeding with Markov and Kalman filter localization, we have to understand the difference among three types of localization problems, which are: position tracking, global localization, and the kidnapped robot problem.

Position tracking. In position tracking, the robot current location is updated based on the knowledge of its previous position (tracking). This implies that the robot initial location is supposed to be known. Additionally, the uncertainty on the robot pose has to be small. If the uncertainty is too large, position tracking might fail to localize the robot. This concept will be investigated more in detail in Kalman filter localization, as in position tracking the robot belief is usually modeled with a unimodal distribution, such as the normal distribution.

Global localization. Global localization, conversely, assumes that the robot initial location is unknown. This means that the robot can be placed anywhere in the environment—without knowledge about it—and is able to localize globally within it. In global localization, the robot initial belief is usually a uniform distribution.

Kidnapped robot problem. The kidnapped robot problem tackles the case the robot gets kidnapped and moved to another location. The kidnapped robot problem is similar to the global localization problem only if the robot realizes having been kidnapped. The difficulty arises when the robot does not know that it has been moved to another location and it believes it knows where it is but in fact does not. The ability to recover from kidnapping is a necessary condition for the operation of any autonomous robot and even more for commercial robots.

5.6.7 Markov localization

Markov localization tracks the robot's belief state using an arbitrary probability density function to represent the robot's position (see also [87, 169, 249, 252]). In practice, all known Markov localization systems implement this generic belief representation by first tessellating the robot configuration space (x, y, θ) into a finite, discrete number of possible robot poses in the map. In actual applications, the number of possible poses can range from several hundred to millions of positions and orientations.

Markov localization addresses the *global localization problem*, the *position tracking problem*, and the *kidnapped robot problem*.

As we mentioned in section 5.6.2, the probabilistic robot localization process consists in the iteration of *prediction* and *measurement* updates. They compute the belief state that results when new information (e.g., encoder values and measurement data) is incorporated into a prior belief state with arbitrary probability density. As we will see, in both Markov and Kalman-filter localization the prediction update is based on the *theorem of total probability* (page 301), while the perception update is based on the *Bayes rule* (page 301).

In the next sections, we will explain separately these two steps for the case of Markov localization. We will first illustrate Markov localization in the continuous case and then in the discrete case (geometric grid-based). Finally, we will show an example of Markov localization using a topological map.

5.6.7.1 Prediction and measurement updates

Prediction (action) update. Let us recall that in this phase the robot estimates its current position (i.e. belief) based on the knowledge of the previous position (i.e., belief) and the odometric input. The *theorem of total probability* (page 301) is used to compute the robot's current belief $\overline{bel}(x_t)$ as a function of the previous belief $bel(x_{t-1})$ and the proprioceptive data (e.g., the encoder measurement or the control input) u_t:

$$\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1} \quad \text{(continuous case)}, \tag{5.40}$$

$$\overline{bel}(x_t) = \sum_{x_{t-1}} p(x_t | u_t, x_{t-1}) bel(x_{t-1}) \quad \text{(discrete case)}. \tag{5.41}$$

As observed, the belief $\overline{bel}(x_t)$ that the robot assigns to the state x_t is obtained by the integral (or sum) of the product of two distributions: the prior assigned to x_{t-1}, and the probability that the control u_t induces a transition from x_{t-1} to x_t.

Let us try to clarify the reason of this integral (sum). In order to compute the probability of position x_t in the new belief state, one must integrate over all the possible ways in which the robot may have reached x_t according to the potential positions expressed in the former

belief state. This is subtle but fundamentally important. The same location x_t can be reached from multiple source locations with the same encoder measurement u_t because the encoder measurement is uncertain.

Also observe that the integral (sum) in (5.40) and (5.41) must be computed over all possible robot positions x_t. This means that in real situations, where the number of cells used to represent the robot poses is several million, the computation (5.40) and (5.41) can become impractical and, hence, impede the real-time operation.

Finally, observe that (5.40) and (5.41) can be seen as a *convolution* (page 197, equation [4.102]) between the previous belief $bel(x_{t-1})$ and the probabilistic motion model $p(x_t|u_t, x_{t-1})$.[24] By thinking in terms of convolution, the reader should now have clear the reason why the prediction update causes the uncertainty of the robot location to grow (figure 5.20).

Perception (measurement) update. Let us recall that in this phase the robot corrects its previous position (i.e. its former belief) by opportunely combining it with the information from its exteroceptive sensors (figure 5.20). The *Bayes rule* (page 301) is used to compute the robot's new belief state $bel(x_t)$ as a function of its measurement data z_t and its former belief state $\overline{bel}(x_t)$:

$$bel(x_t) = \eta p(z_t|x_t, M)\overline{bel}(x_t), \tag{5.42}$$

where $p(z_t|x_t, M)$ is the probabilistic measurement model (page 305), that is, the probability of observing the measurement data z_t given the knowledge of the map M and the robot pose x_t. Therefore, the new belief state is simply the product between the probabilistic measurement model and the previous belief state. Observe that (5.42) does not update only one pose but all possible robot poses x_t.

The Markov localization algorithm. Figure 5.21 depicts the Markov localization algorithm in pseudo-algorithmic form.[25]

The critical challenge in Markov localization is the calculation of $p(z_t|x_t, M)$. The sensor model must calculate the probability of a specific perceptual measurement given the location of the robot and the map of the environment. Three key assumptions are used to construct this sensor model:

24. Note that (5.40) is not a real convolution because here the sign of none of the two functions is inverted.

25. Note that, since both Markov and Kalman localization make use of the Bayes rule, they are also called Bayes filters.

```
for all x_t do
   bel̄(x_t) = ∫ p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}   (prediction update)
   bel(x_t) = η p(z_t | x_t, M) bel̄(x_t)                      (measurement update)
endfor
return bel(x_t)
```

Figure 5.21 The general algorithm for Markov localization.

1. If an object in the map is detected by, for instance, a range sensor, the measurement error can be described with a distribution that has a mean at the correct reading. The adopted distribution is usually a Gaussian.
2. There should always be a nonzero chance that a range sensor will read any measurement value, even if this measurement disagrees sharply with the environmental geometry. This means that the adopted distribution should always be nonzero in the range of all possible values returnable by the sensors. The peak should be centered at the correct sensor reading and the probability should be set to small values elsewhere. Again, a Gaussian distribution implicitly solves this problem.
3. In contrast to the generic error described in at the previous point, there is a specific failure mode in ranging sensors whereby the signal is absorbed or coherently reflected, causing the sensor's range measurement to be maximal. Therefore, there is a local peak in the probability density distribution at the maximal reading of a range sensor.

5.6.7.2 The Markov assumption

Equations (5.40) and (5.42) form the basis of Markov localization and incorporate the *Markov assumption*. Formally, this means that their output x_t is a function only of the robot's previous state x_{t-1} and its most recent actions (odometry) u_t and perception z_t. In a general, *non-Markovian* situation, the state of a system depends upon all of its history. After all, the values of a robot's sensors at time t do not really depend only on its position at time t. They depend to some degree on the trajectory of the robot over time, indeed, on the entire history of the robot. For example, the robot could have experienced a serious collision recently that has biased the sensor's behavior. Similarly, the position of the robot at time t does not really depend only on its position at time $t-1$ and its odometric measurements. Due to its history of motion, one wheel may have worn more than the other, causing a left-turning bias over time that affects its current position. Additionally, there might also be unmodeled dynamics of the environment such as moving people (which have effect on sensor measurements), inaccuracies in the probabilistic motion and measurement models,

errors in the map used for a localizing robot, and software variables that influence multiple controls.

So the Markov assumption is, of course, not a valid assumption. However, the Markov assumption greatly simplifies tracking, reasoning, and planning, and so it is an approximation that continues to be extremely popular in mobile robotics. Indeed, Markov localization has been found to be surprisingly robust to such violations.

5.6.7.3 Illustration of Markov localization

In figure 5.22, we illustrate the working principle of the Markov localization in the continuous case. For simplicity, our environment is a one-dimensional hallway with three identical pillars.

In this example, we assume that at the beginning the robot does not know its initial location and has therefore to localize from scratch. Clearly, this is a global localization problem. According to the probabilistic framework described before, the robot initial belief $bel(x_0)$ is a uniform distribution over all locations as illustrated in figure 5.22a.

Now suppose that the robot uses its exteroceptive sensors and senses that it is next to a pillar. This is clearly the *perception update* of Markov localization. Then, according to the Bayes rule its belief $bel(x_0)$ has to be multiplied by $p(z_t|x_t, M)$ as stated in equation (5.42). How do we characterize $p(z_t|x_t, M)$? Because the three pillars are exactly identical, the robot does not know which one of the pillars is facing. Therefore, the probability $p(z_t|x_t, M)$ of observing a pillar is characterized by three peaks, each corresponding to one of the indistinguishable pillars in the environment. The upper plot in (b) visualizes $p(z_t|x_t, M)$. After this perception update the robot still does not know where it is. Indeed, it now has three, distinct hypotheses which are all equally plausible. Also notice that the probability in the regions not next to a pillar is nonzero. In probabilistic robot localization we can never be 100% sure that the robot is not somewhere. Therefore, it is important to maintain low-probability hypotheses. This is essential for achieving robustness, for example, if the robot gets lost or kidnapped. The lower plot in (b) visualizes the result of the multiplication. Because it results from a multiplication with a constant function, the result is still characterized by the three exactly identical peaks.

Now suppose that the robot moves to the right. We are now in the *action update* of Markov localization. Figure 5.22c shows the effect on the robot's belief. As a result of the *convolution* of the robot's previous belief with the motion model $p(x_t|u_t, x_{t-1})$, the new belief has been shifted in the direction of motion and also flattened out. The three peaks are now larger, which reflects the uncertainty that is introduced by the robot motion.

Figure 5.22d depicts the belief after observing another pillar. We are again in the *perception update*. Here, Markov localization algorithm multiples again the current belief with the perceptual probability $p(z_t|x_t, M)$. As observed, this time the result of the multiplication

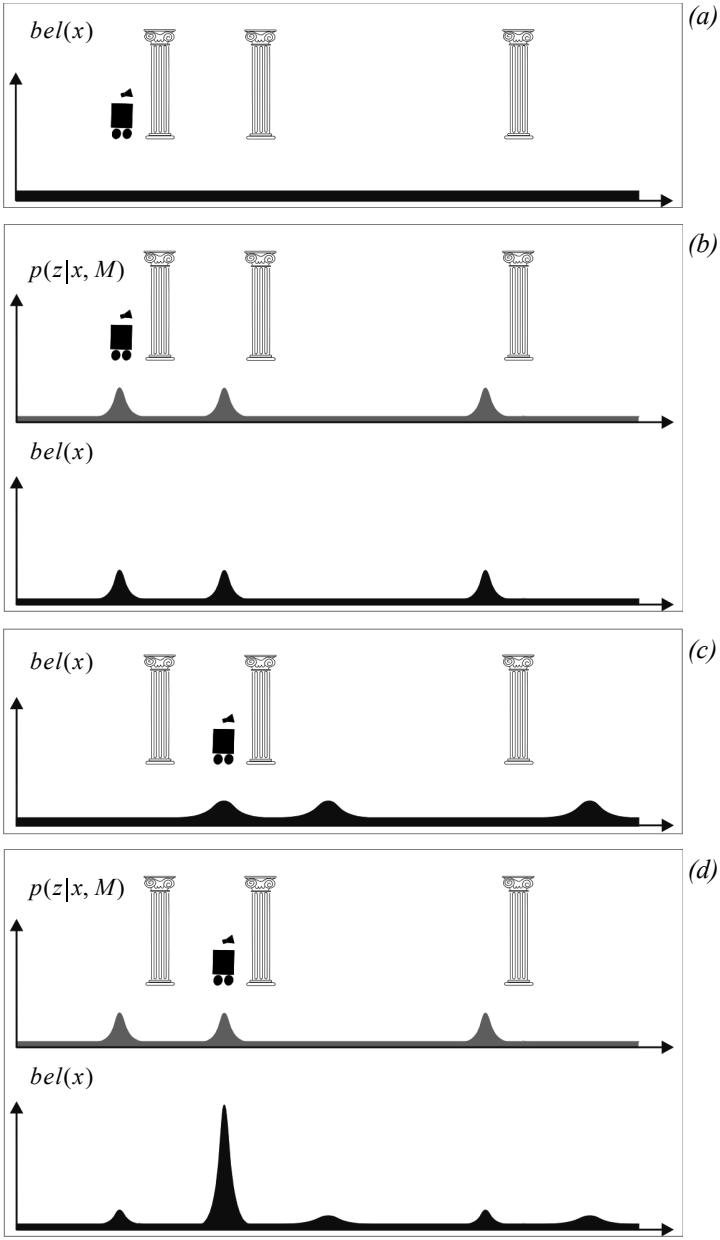


Figure 5.22 Illustration of the Markov localization algorithm.

is a single distinguishable peak near one of the pillars, and the robot is now quite confident of where it is.

5.6.7.4 Case study 1: Markov localization using a grid map

Markov localization is implemented in practice using a grid-space representation of the environment. Usually, a fixed decomposition is used, which consists in tessellating the state-space into fine-grained cells of uniform size (section 5.5.2).

For planar motion, the robot configuration is expressed by three parameters (x, y, θ). This means that also the space of all possible robot orientations must be discretized. The final state-space is therefore stored in the memory of the robot as a three-dimensional array (figure 5.24).

In this section, we illustrate the working principle of Markov localization using a grid map. For simplicity, we will assume again that the environment is one-dimensional. Considerations for the more general case of 2D environments will be done at the end of this section.

Let us tessellate our environment into ten equally spaced cells (figure 5.23). Suppose that the robot's initial belief $bel(x_0)$ is a uniform distribution from 0 to 3 as shown in figure 5.23a. Observe that all the elements were normalized so that their sum is 1.

Prediction update. Let us recall that in this phase, the robot moves and updates its belief using the motion model of the control input. Therefore, we need the probabilistic motion model (i.e. the odometric error model). Let us assume that the probabilistic motion model of the odometry $p(x_1 | u_1, x_0)$ is the one represented in figure 5.23b. This model must be interpreted in this way: between time $t = 0$ and time $t = 1$, the robot may have moved either two or three units to the right. In this example, both movements have the same probability to occur. What will the robot belief be after this movement? The answer is again in the prediction-phase equation (5.41), that is, the final belief $\overline{bel}(x_1)$ is given by the theorem of total probability, which convolves the initial belief $bel(x_0)$ with the motion model $p(x_1 | u_1, x_0)$. Using (5.41), we obtain:

$$\overline{bel}(x_1) = \sum_{x_0 = 0}^{3} p(x_1 | u_1, x_0) bel(x_0) . \tag{5.43}$$

Here, we would like to explain where this formula actually comes from. In order to compute the probability of position x_1 in the new belief state, one must sum over all the possible ways in which the robot may reach x_1 according to the potential positions expressed in the former belief state x_0 and the potential input expressed by u_1. Observe that because x_0 is limited between 0 and 3, the robot can only reach the states between 2 to 6, therefore:

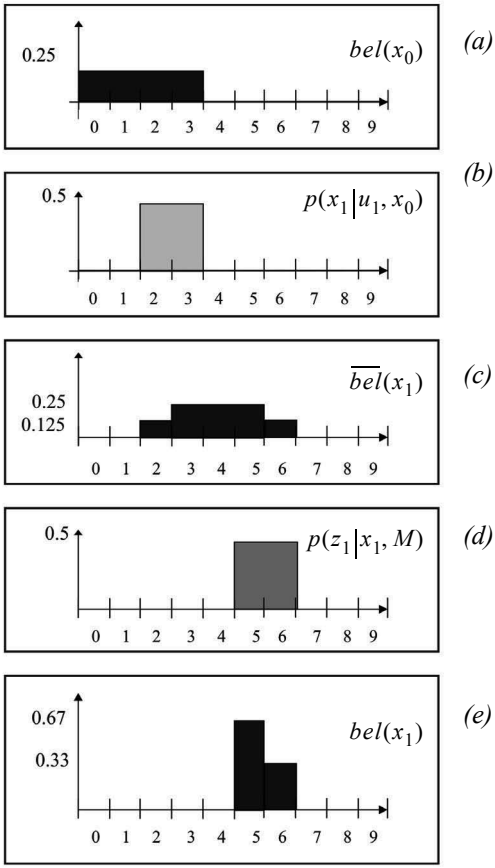


Figure 5.23 Markov localization using a grid-map.

$$p(x_1 = 2) = p(x_0 = 0)p(u_1 = 2) = 0.125\,, \tag{5.44}$$

$$p(x_1 = 3) = p(x_0 = 0)p(u_1 = 3) + p(x_0 = 1)p(u_1 = 2) = 0.25 \tag{5.45}$$

$$p(x_1 = 4) = p(x_0 = 1)p(u_1 = 3) + p(x_0 = 2)p(u_1 = 2) = 0.25 \tag{5.46}$$

$$p(x_1 = 5) = p(x_0 = 2)p(u_1 = 3) + p(x_0 = 3)p(u_1 = 2) = 0.25 \tag{5.47}$$

$$p(x_1 = 6) = p(x_0 = 3)p(u_1 = 3) = 0.125 \tag{5.48}$$

Expression (5.44) results from the fact that the state $x_1 = 2$ can only be reached with the combination $(x_0 = 0, u_1 = 2)$. Expression (5.45) comes from the fact that the state $x_1 = 3$ can only be reached with the combinations $(x_0 = 0, u_1 = 3)$ *or* $(x_0 = 1, u_1 = 2)$. The other expressions follow in a similar way. The reader can now verify that equations (5.44)–(5.48) are implementing nothing but the theorem of total probability (or convolution)[26] enunciated in (5.43). The result of the application of this theorem is shown in figure 5.23c.

Measurement update. Let us now assume that the robot uses its onboard rangefinder and measures the distance z from the origin. Assume that the statistical error model of the range sensor is the one shown in figure 5.23d. This plot tells us that the distance of the robot from the origin can be equally 5 or 6 units. What will the final robot belief be after this measurement? The answer is again in the measurement-update equation (5.42). The final belief $bel(x_1)$ is computed accordingly to the Bayes rule. It is the product between the robot current belief $\overline{bel}(x_1)$ and the measurement error model $p(z_1 | x_1, M)$, where, in this case, the map M is simply the origin of the axes. Therefore:

$$bel(x_1) = \eta p(z_1 | x_1, M) \overline{bel}(x_1) . \tag{5.49}$$

The reader can verify that we need $\eta = 1/0.1875 \cong 5.33$ to make the final result $bel(x_1)$ normalized to one. The final belief is shown in figure 5.23e.

3D grid maps. As we said at the beginning of the section, in the more general planar motion case the grid-map is a three-dimensional array where each cell contains the probability of the robot to be in that cell (figure 5.24). In this case, the cell size must be chosen carefully. During each prediction and measurement steps, all the cells are updated. If the number of cells in the map is too large, the computation can become too heavy for real-time operations. The convolution in a 3D space is clearly the computationally most expensive step. As an example, consider a $30 \text{ m} \times 30 \text{ m}$ environment and a cell size of $0.1 \text{ m} \times 0.1 \text{ m} \times 1°$. In this case, the number of cells that need to be updated at each step would be $30 \times 30 \times 100 \times 360 = 32.4$ million cells!

One possible solution would then be to increase the cell size at the expense of localization accuracy. Another solution is to use an *adaptive* cell decomposition instead of a *fixed* cell decomposition as proposed by Burgard et al. [86, 87]. In this work, they overcame the problem of the huge state space by dynamically adapting the size of the cells according to the robot's certainty in its position, that is, smaller cells where the robot is more certain to

26. As mentioned earlier, notice that we are using improperly the term convolution. The only difference with convolution is that the theorem of total probability does not invert the sign of neither of the two argument functions.

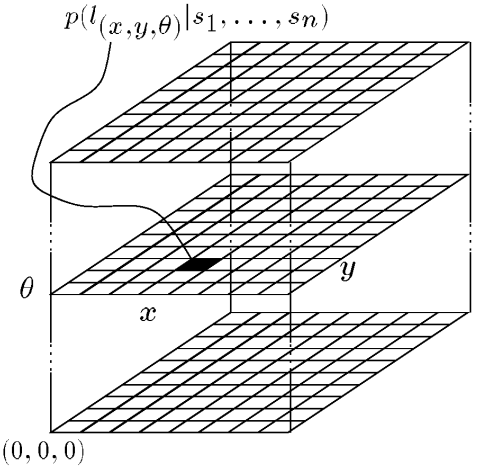


Figure 5.24
The belief state representation 3D array used in Markov localization (courtesy of W. Burgard and S. Thrun).

be and bigger cells elsewhere. This way, they were able to localize a robot in a $30\ \text{m} \times 30\ \text{m}$ with an error smaller than 4 cm using a number of cells varying only between 400 and 3600.

The resulting robot localization system of Burgard and his colleagues has been part of a navigation system that has demonstrated great success both at the University of Bonn (Germany) and at a public museum in Bonn. This is a challenging application because of the dynamic nature of the environment, as the robot's sensors are frequently subject to occlusion due to humans gathering around the robot. The robot ability to function well in this setting is a demonstration of the power of the Markov localization approach.

Reducing computational complexity: Randomized sampling. One class of techniques deserves mention because it can significantly reduce the computational overhead of techniques that employ fixed-cell decomposition representations. The basic idea, which we call *randomized sampling,* is known alternatively as *particle filter* algorithms, *condensation algorithms*, and *Monte Carlo* algorithms [129, 311].

Irrespective of the specific technique, the basic algorithm is the same in all these cases. Instead of representing *every* possible robot position by representing the complete and correct belief state, an approximate belief state is constructed by representing only a *subset* of the complete set of possible locations that should be considered.

For example, consider a robot with a complete belief state of 10,000 possible locations at time t. Instead of tracking and updating all 10,000 possible locations based on a new

sensor measurement, the robot can select only 10% of the stored locations and update only those locations. By weighting this sampling process with the probability values of the locations, one can bias the system to generate more samples at local peaks in the probability density function. So the resulting 1000 locations will be concentrated primarily at the highest probability locations. This biasing is desirable, but only to a point.

We also wish to ensure that *some* less likely locations are tracked, as otherwise, if the robot does indeed receive unlikely sensor measurements, it will fail to localize. This *randomization* of the sampling process can be performed by adding additional samples from a flat distribution, for example. Further enhancements of these randomized methods enable the number of statistical samples to be varied on the fly, based, for instance, on the ongoing localization confidence of the system. This further reduces the number of samples required on average while guaranteeing that a large number of samples will be used when necessary [129].

These sampling techniques have resulted in robots that function indistinguishably as compared to their full belief state set ancestors, yet use computationally a fraction of the resources. Of course, such sampling has a penalty: completeness. The probabilistically complete nature of Markov localization is violated by these sampling approaches because the robot is failing to update *all* the nonzero probability locations, and thus there is a danger that the robot, due to an unlikely but correct sensor reading, could become truly lost. Of course, recovery from a lost state is feasible just as with all Markov localization techniques.

5.6.7.5 Case study 2: Markov localization using a topological map

A straightforward application of Markov localization is possible when the robot's environment representation already provides an appropriate decomposition. This is the case when the environmental representation is purely topological.

Consider a contest in which each robot is to receive a topological description of the environment. The description would include only the connectivity of hallways and rooms, with no mention of geometric distance. In addition, this supplied *map* would be imperfect, containing several false arcs (e.g., a closed door). Such was the case for the 1994 American Association for Artificial Intelligence (AAAI) National Robot Contest, at which each robot's mission was to use the supplied map and its own sensors to navigate from a chosen starting position to a target room.

Dervish, the winner of this contest, employed probabilistic Markov localization and used a multiple-hypothesis belief state over a topological environmental representation. We now describe Dervish as an example of a robot with a discrete, topological representation and a probabilistic localization algorithm.

Dervish, shown in figure 5.25, includes a sonar arrangement custom-designed for the 1994 AAAI National Robot Contest. The environment in this contest consisted of a rectilinear indoor office space filled with real office furniture as obstacles. Traditional sonars

Figure 5.25
Dervish exploring its environment.

were arranged radially around the robot in a ring. Robots with such sensor configurations are subject to both tripping over short objects below the ring and to decapitation by tall objects (such as ledges, shelves, and tables) that are above the ring.

Dervish's answer to this challenge was to arrange one pair of sonars diagonally upward to detect ledges and other overhangs. In addition, the diagonal sonar pair also proved to ably detect tables, enabling the robot to avoid wandering underneath tall tables. The remaining sonars were clustered in sets of sonars, such that each individual transducer in the set would be at a slightly varied angle to minimize specularity. Finally, two sonars near the robot's base were positioned to detect low obstacles, such as paper cups, on the floor.

We have already noted that the representation provided by the contest organizers was purely topological, noting the connectivity of hallways and rooms in the office environment. Thus, it would be appropriate to design Dervish's perceptual system to detect matching perceptual events: the detection and passage of connections between hallways and offices.

This *abstract* perceptual system was implemented by viewing the trajectory of sonar strikes to the left and right sides of Dervish over time. Interestingly, this perceptual system would use time alone and no concept of encoder value to trigger perceptual events. Thus, for instance, when the robot detects a 7–17 cm indentation in the width of the hallway for

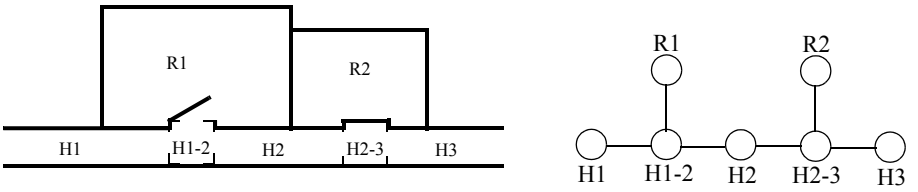


Figure 5.26
A geometric office environment (left) and its topological analog (right).

more than 1 second continuously, a *closed door* sensory event is triggered. If the sonar strikes jump well beyond 17 cm for more than 1 second, an *open door* sensory event triggers.

To reduce coherent reflection sensor noise (see section 4.1.9) associated with Dervish's sonars, the robot would track its angle relative to the hallway center line and completely suppress sensor events when its angle to the hallway exceeded 9 degrees. Interestingly, this would result in a conservative perceptual system that frequently misses features, particularly when the hallway is crowded with obstacles that Dervish must negotiate. Once again, the conservative nature of the perceptual system, and in particular its tendency to issue false negatives, would point to a probabilistic solution to the localization problem so that a complete trajectory of perceptual inputs could be considered.

Dervish's environmental representation was a discrete topological map, identical in abstraction and information to the map provided by the contest organizers. Figure 5.26 depicts a geometric representation of a typical office environment overlaid with the topological map for the same office environment. Recall that for a topological representation the key decision involves assignment of nodes and connectivity between nodes (see section 5.5.2). As shown on the left in figure 5.26, Dervish uses a topology in which node boundaries are marked primarily by doorways (and hallways and foyers). The topological graph shown on the right depicts the information captured in the example shown.

Note that in this particular topological model arcs are zero-length while nodes have spatial expansiveness and together cover the entire space. This particular topological representation is particularly apt for Dervish, given its task of navigating through hallways into a specific room and its perceptual capability of recognizing discontinuities in hallway walls.

In order to represent a specific belief state, Dervish associated with each topological node n a probability that the robot is at a physical position within the boundaries of n: $p(x_t = n)$. As will become clear, the probabilistic update used by Dervish was approxi-

mate, therefore technically one should refer to the resulting values as *likelihoods* rather than probabilities.

Table 5.1
Dervish's certainty matrix.

	Wall	Closed door	Open door	Open hallway	Foyer
Nothing detected	0.70	0.40	0.05	0.001	0.30
Closed door detected	0.30	0.60	0	0	0.05
Open door detected	0	0	0.90	0.10	0.15
Open hallway detected	0	0	0.001	0.90	0.50

The perception update process for Dervish functions precisely as in equation (5.42). Perceptual events are generated asynchronously, each time the feature extractor is able to recognize a large scale feature (e.g., doorway, intersection) based on recent ultrasonic values. Each perceptual event consists of a percept-pair (a feature on one side of the robot or two features on both sides).

Given a specific percept-pair z, equation (5.42) enables the likelihood of each possible position n to be updated using the formula:

$$p(n|z) = \eta p(z|n)p(n). \tag{5.50}$$

The value of $p(n)$ is already available from the current belief state of Dervish, and so the challenge lies in computing $p(z|n)$. The key simplification for Dervish is based upon the realization that, because the feature extraction system only extracts four total features and because a node contains (on a single side) one of five total features, every possible combination of node type and extracted feature can be represented in a 4×5 table.

Dervish's *certainty matrix* (shown in table 5.1) is just this lookup table. Dervish makes the simplifying assumption that the performance of the feature detector (i.e., the probability that it is correct) is only a function of the feature extracted and the actual feature in the node. With this assumption in hand, we can populate the *certainty matrix* with confidence estimates for each possible pairing of perception and node type. For each of the five world features that the robot can encounter (wall, closed door, open door, open hallway, and foyer) this matrix assigns a likelihood for each of the three one-sided percepts that the sensory system can issue. In addition, this matrix assigns a likelihood that the sensory system will fail to issue a perceptual event altogether (*nothing detected*).

For example, using the specific values in table 5.1, if Dervish is next to an open hallway, the likelihood of mistakenly recognizing it as an open door is 0.10. This means that for any node n that is of type *open hallway* and for the sensor value $z = \textit{open door}$, $p(z|n) = 0.10$. Together with a specific topological map, the certainty matrix enables straightforward computation of $p(z|n)$ during the perception update process.

For Dervish's particular sensory suite and for any specific environment it intends to navigate, humans generate a specific certainty matrix that loosely represents its perceptual confidence, along with a global measure for the probability that any given door will be closed versus opened in the real world.

Recall that Dervish has no encoders and that perceptual events are triggered asynchronously by the feature extraction processes. Therefore, Dervish has no prediction update step as depicted by equation (5.41). When the robot does detect a perceptual event, multiple perception update steps will need to be performed to update the likelihood of every possible robot position given Dervish's former belief state. This is because there is a chance that the robot has traveled *multiple* topological nodes since its previous perceptual event (i.e., false-negative errors). Formally, the perception update formula for Dervish is in reality a combination of the general form of prediction update and measurement update. The likelihood of position n given perceptual event i is calculated as in equation (5.41):

$$p(n_t|z_t) = \sum p(n_t|n'_{t-i}, z_t)p(n'_{t-i}) . \tag{5.51}$$

The value of $p(n'_{t-i})$ denotes the likelihood of Dervish being at position n' as represented by Dervish's former belief state. The temporal subscript $t-i$ is used in lieu of $t-1$ because for each possible position n' the discrete topological distance from n' to n can vary depending on the specific topological map. The calculation of $p(n_t|n'_{t-i}, z_t)$ is performed by multiplying the probability of generating perceptual event z at position n by the probability of having failed to generate perceptual events at all nodes between n' and n:

$$p(n_t|n'_{t-i}, z_t) = p(z_t, n_t) \cdot p(\varnothing, n_{t-1}) \cdot p(\varnothing, n_{t-2}) \cdot \ldots \cdot p(\varnothing, n_{t-i+1}) . \tag{5.52}$$

For example (figure 5.27), suppose that the robot has only two nonzero nodes in its belief state, {1-2, 2-3}, with likelihoods associated with each possible position: $p(1-2) = 1.0$ and $p(2-3) = 0.2$. For simplicity assume the robot is facing east with certainty. Note that the likelihoods for nodes 1–2 and 2–3 do not sum to 1.0. These values are not formal probabilities, and so computational effort is minimized in Dervish by avoiding normalization altogether. Now suppose that a perceptual event is generated: the robot detects an open hallway on its left and an open door on its right simultaneously.

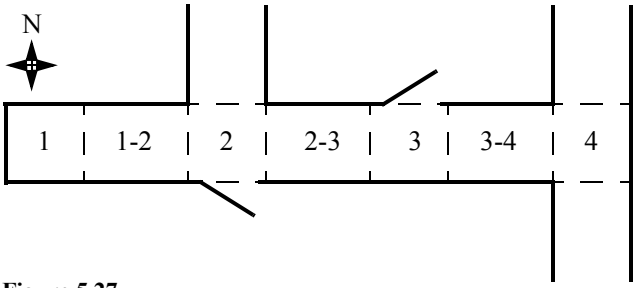


Figure 5.27
A realistic indoor topological environment.

State 2–3 will progress potentially to states 3, 3–4, and 4. But states 3 and 3–4 can be eliminated because the likelihood of detecting an open door when there is only wall is zero. The likelihood of reaching state 4 is the product of the initial likelihood for state 2–3, 0.2, the likelihood of not detecting anything at node 3, (a), and the likelihood of detecting a hallway on the left and a door on the right at node 4, (b). Note that we assume the likelihood of detecting nothing at node 3–4 is 1.0 (a simplifying approximation).

(a) occurs only if Dervish fails to detect the door on its left at node 3 (either closed or open), $[0.6 \cdot 0.4 + (1 - 0.6) \cdot 0.05]$, and correctly detects nothing on its right, 0.7.

(b) occurs if Dervish correctly identifies the open hallway on its left at node 4, 0.90, and mistakes the right hallway for an open door, 0.10.

The final formula, $0.2 \cdot [0.6 \cdot 0.4 + 0.4 \cdot 0.05] \cdot 0.7 \cdot [0.9 \cdot 0.1]$, yields a likelihood of 0.003 for state 4. This is a partial result for $p(4)$ following from the prior belief state node 2-3.

Turning to the other node in Dervish's prior belief state, 1–2 will potentially progress to states 2, 2–3, 3, 3–4, and 4. Again, states 2–3, 3, and 3–4 can all be eliminated since the likelihood of detecting an open door when a wall is present is zero. The likelihood of state 2 is the product of the prior likelihood for state 1–2, (1.0), the likelihood of detecting the door on the right as an open door, $[0.6 \cdot 0 + 0.4 \cdot 0.9]$, and the likelihood of correctly detecting an open hallway to the left, 0.9. The likelihood for being at state 2 is then $1.0 \cdot 0.4 \cdot 0.9 \cdot 0.9 = 0.3$. In addition, 1–2 progresses to state 4 with a certainty factor of $4.3 \cdot 10^{-6}$, which is added to the certainty factor above to bring the total for state 4 to 0.00328. Dervish would therefore track the new belief state to be {2, 4}, assigning a very high likelihood to position 2 and a low likelihood to position 4.

Empirically, Dervish's map representation and localization system have proved to be sufficient for navigation of four indoor office environments: the artificial office environment created explicitly for the 1994 National Conference on Artificial Intelligence; and the psychology, history, and computer science departments at Stanford University. All of these

experiments were run while providing Dervish with no notion of the distance between adjacent nodes in its topological map. It is a demonstration of the power of probabilistic localization that, in spite of the tremendous lack of action and encoder information, the robot is able to navigate several real-world office buildings successfully.

One open question remains with respect to Dervish's localization system. Dervish was not just a localizer but also a navigator. As with all multiple hypothesis systems, one must ask the question, how does the robot decide how to move, given that it has multiple possible robot positions in its representation? The technique employed by Dervish is a common technique in the mobile robotics field: plan the robot's actions by assuming that the robot's actual position is its most likely node in the belief state. Generally, the most likely position is a good measure of the robot's actual world position. However, this technique has shortcomings when the highest and second highest most likely positions have similar values. In the case of Dervish, it nonetheless goes with the highest-likelihood position at all times, save at one critical juncture. The robot's goal is to enter a target room and remain there. Therefore, from the point of view of its goal, it is critical that Dervish finish navigating only when the robot has strong confidence in being at the correct final location. In this particular case, Dervish's execution module refuses to enter a room if the gap between the most likely position and the second likeliest position is below a preset threshold. In such a case, Dervish will actively plan a path that causes it to move farther down the hallway in an attempt to collect more sensor data and thereby increase the relative likelihood of one position in the multiple-hypothesis belief state.

Although computationally unattractive, one can go farther, imagining a planning system for robots such as Dervish for which one specifies a *goal belief state* rather than a goal position. The robot can then reason and plan in order to achieve a goal confidence level, thus explicitly taking into account not only robot position but also the measured likelihood of each position. An example of just such a procedure is the sensory uncertainty field of Latombe [306], in which the robot must find a trajectory that reaches its goal while maximizing its localization confidence on-line.

The major weakness of a purely topological decomposition of the environment is the resolution limitation imposed by such a granular representation. The position of the robot is usually limited to the resolution of a single node in such cases, and this may be undesirable for certain applications.

5.6.8 Kalman filter localization

5.6.8.1 Introduction

The Markov localization model can represent any arbitrary probability density function over the robot position. This approach is very general but, due to its generality, inefficient. Consider instead the key demands on a robot localization system. One can argue that it is not the exact replication of a probability density curve but the *sensor fusion* problem that is

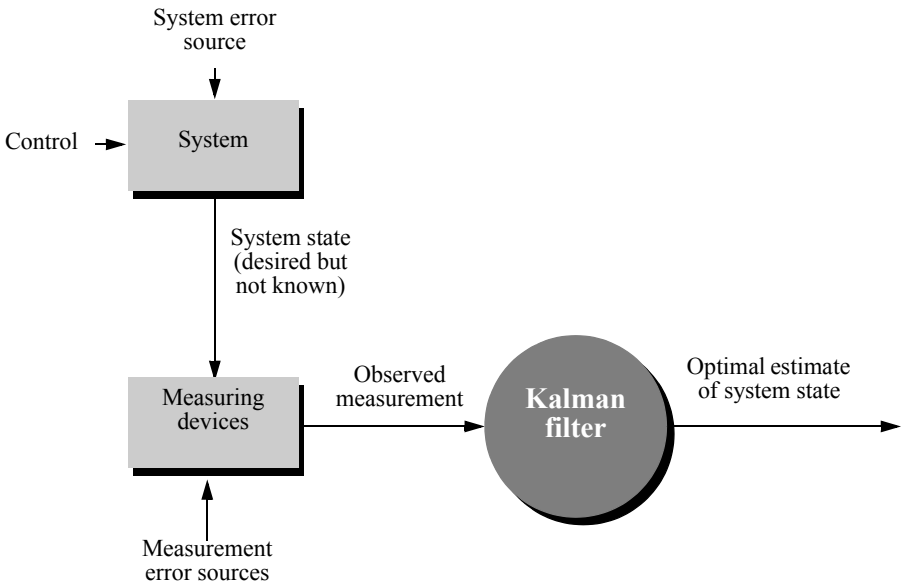


Figure 5.28
Typical Kalman filter application [209].

key to robust localization. Robots usually include a large number of heterogeneous sensors, each providing clues as to robot position and, critically, each suffering from its own failure modes. Optimal localization should take into account the information provided by all of these sensors. In this section we describe a powerful technique for achieving this sensor fusion, called the Kalman filter. This mechanism is in fact more efficient than Markov localization because of key simplifications when representing the probability density function of the robot's belief state and even its individual sensor readings, as described below. But the benefit of this simplification is a resulting *optimal recursive data-processing algorithm*. It incorporates all information, regardless of precision, to estimate the current value of the variable of interest (i.e., the robot's position). A general introduction to Kalman filters can be found in [209], and a more detailed treatment is presented in [3].

Figure 5.28 depicts the general scheme of Kalman filter estimation, where a system has a control signal and system error sources as inputs. A measuring device enables measuring some system states with errors. The Kalman filter is a mathematical mechanism for producing an optimal estimate of the system state based on the knowledge of the *system* and the *measuring device*, the description of the system noise and measurement errors and the uncertainty in the dynamics models. Thus the Kalman filter *fuses* sensor signals and system

knowledge in an optimal way. Optimality depends on the criteria chosen to evaluate the performance and on the assumptions. Within the Kalman filter theory the system is assumed to be *linear* and with *white Gaussian* noise. For most mobile robot applications, the system is nonlinear. In this cases, the Kalman filter is usually applied after linearizing the system. The extension of Kalman filter to nonlinear systems is known as the *Extended Kalman Filter* (*EKF*), but its optimality cannot be guaranteed. As we have discussed earlier, the assumption of Gaussian error is invalid for our mobile robot applications, but nevertheless the results are extremely useful. In other engineering disciplines, the Gaussian error assumption has in some cases been shown to be quite accurate [209].

We begin with a section that illustrates the Kalman filter localization (section 5.6.8.2). Then, we introduce the Kalman filter theory (section 5.6.8.3), and present an application of that theory to the problem of mobile robot localization (5.6.8.4). Finally, section 5.6.8.5 presents a case study of a mobile robot that navigates indoor spaces by virtue of Kalman filter localization.

5.6.8.2 Illustration of Kalman filter localization

The *Kalman filter localization* algorithm, or *KF localization*, is a special case of Markov localization. Instead of using an arbitrary density function, the Kalman filter uses Gaussians to represent the robot belief $bel(x_t)$, the motion model, and the measurement model. Because a Gaussian is simply defined by its mean μ_t and covariance Σ_t, only these two parameters are updated during the prediction and measurement phase, resulting in a very efficient algorithm in comparison to Markov localization algorithm. However, the assumptions made by the Kalman filter limit the choice of the initial belief $bel(x_0)$ also to a Gaussian, which means that the robot initial location must be known with a certain approximation. Hence, the robot cannot recover its position if it gets lost. This is in contrast with the Markov localization. The Kalman filter therefore addresses the position-tracking problem but not the global localization or the kidnapped robot problem.

Figure 5.29 illustrates the Kalman filter localization algorithm using again our example of a mobile robot in a one-dimensional environment. As mentioned, the robot initial belief $bel(x_0)$ is represented by a Gaussian distribution. As shown in figure 5.29a, we assume that at the beginning the robot is near the first pillar. As the robot moves to the right (we are in the *action phase*), its uncertainty increases as a result of the convolution with the motion model (i.e., application of the theorem of total probability). The resulting belief is therefore a shifted Gaussian of increased width, figure 5.29b. Now, suppose that the robot uses its exteroceptive sensors (we are in the *perception phase*) and senses that it is near the second pillar. The posterior probability $p(z_t|x_t, M)$ of the observation is shown in figure 5.29c. This probability density is again a Gaussian. In order to compute the robot current belief, we must fuse this measurement probability with the robot's belief before the observation using the Bayes rule. The result of this fusion is again a Gaussian shown at the bottom of figure

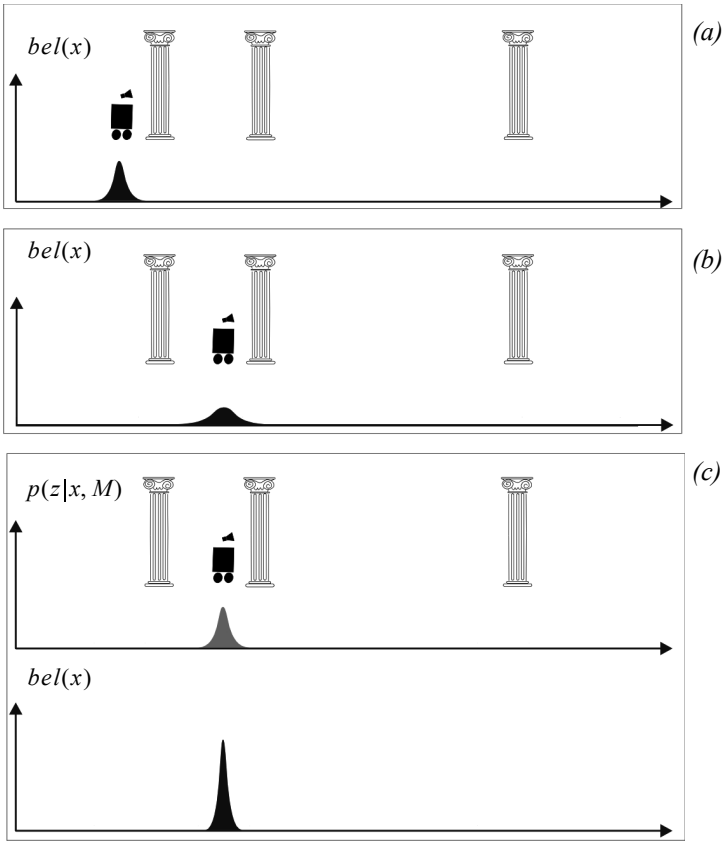


Figure 5.29 Application of the Kalman filter algorithm to mobile robot localization.

5.29c. Note that the variance of the resulting belief is smaller than the variances of both the measurement probability and the robot's previous belief. This result is obvious, since the fusion of two independent estimates should make the robot more certain than each individual estimate.

5.6.8.3 Introduction to Kalman filter theory

As said in the previous section, the Kalman filter theory is based on the assumptions that the system is linear and that overall the robot configuration, the odometric error model, and the measurement error model are affected by white Gaussian noise.

A Gaussian distribution is represented only by its first and second moments, which are the mean μ_t and the variance σ_t^2 (see equation 5.20). When the robot configuration is a vector (which is the case in practical applications) the distribution is a multivariate Gaussian represented by a mean vector μ_t and a covariance matrix Σ_t (see equation 5.22).

During the *prediction* and *measurement* updates only mean μ_t and covariance Σ_t are updated. Therefore, the Kalman filter is based on four equations: two for updating μ_t and Σ_t in the prediction update, and another two in the measurement update. In Kalman filtering the measurement update is also commonly called *correction update*.

As for Markov localization, the *prediction* and *measurement* update equations of Kalman filter are also based respectively on the *theorem of total probability* and on the *Bayes rule*. In this section, we will review these properties applied to special case of Gaussian distributions. For an in-depth study of these properties, we refer the reader to [41].

Applying the theorem of total probability. Let x_1, x_2 be two random variables that are independent and normally distributed:

$$x_1 = N(\mu_1, \sigma_1^2) \tag{5.53}$$

$$x_2 = N(\mu_2, \sigma_2^2)\,. \tag{5.54}$$

Let also y be a function of x_1 and x_2, that is,

$$y = f(x_1, x_2)\,. \tag{5.55}$$

What will the distribution of y be? The answer is much simpler when f is a linear function of the inputs, that is, when

$$y = Ax_1 + Bx_2\,. \tag{5.56}$$

In this case, if the inputs are independent, it can be shown that y is also normally distributed with mean and variance given by the following expressions:

$$\langle y \rangle = A\mu_1 + B\mu_2 \tag{5.57}$$

$$\sigma_y^2 = A^2\sigma_1^2 + B^2\sigma_2^2 \tag{5.58}$$

If x_1 and x_2 are vectors with covariances Σ_1 and Σ_2 respectively, then

$$\langle y \rangle = A\mu_1 + B\mu_2\,. \tag{5.59}$$

$$\Sigma_y = A\Sigma_1 A^T + B\Sigma_2 B^T\,. \tag{5.60}$$

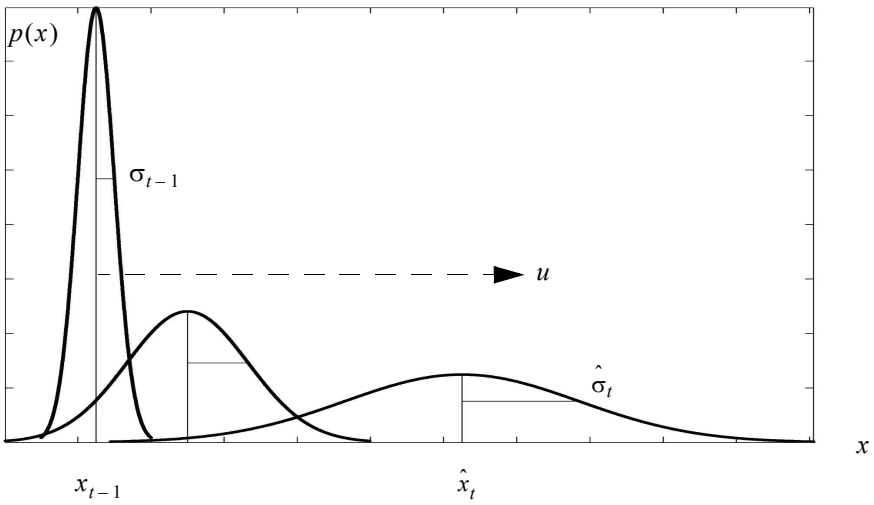


Figure 5.30
Propagation of probability density of a moving robot [209].

This result follows directly from the application of the total probability theorem. We can also look at this problem in terms of convolution, by reminding that the probability density function of the sum of two independent random variables is the convolution of each of their density functions [41]. It can also be shown that the convolution of two Gaussian random variables is another Gaussian [41].

In the case f is nonlinear, y is *not* normally distributed. However, it is a common practice to consider a first-order approximation by linearizing f about (μ_1, μ_2):

$$y \cong f(\mu_1, \mu_2) + F_{x_1}(x_1 - \mu_1) + F_{x_2}(x_2 - \mu_2) . \tag{5.61}$$

where F_{x_1} and F_{x_2} are the jacobians of f. This way, we obtain:

$$\langle y \rangle = f(\mu_1, \mu_2) . \tag{5.62}$$

$$\Sigma_y = F_{x_1} \Sigma_1 F_{x_1}{}^T + F_{x_2} \Sigma_2 F_{x_2}{}^T . \tag{5.63}$$

Equations (5.62) and (5.63) will be used in section 5.6.8.4 to implement the *prediction update* of Extended Kalman Filter (EKF)[27] localization. As we will show, in Kalman localization f is used to represent the odometric position update and its inputs are the robot pre-

vious position x_{t-1} and the control input u. In the simple case of a one-dimensional environment, the odometric position update is described by a simple sum, therefore, $f(x_{t-1}, u) = x_{t-1} + u$ and the update of the uncertainty over the time expressed in (5.63) is illustrated in figure 5.30. Observe that the uncertainty of the robot position after the application of (5.63) is larger.

Applying the Bayes rule. Let q denote the robot position, $p_1(q)$ the robot's belief resulting from the prediction update, and $p_2(q)$ the robot's belief resulting from some exteroceptive sensor measurement (for example, a rangefinder that returns the position of the robot directly in the global reference frame). The Bayes rule tells us how to compute the final distribution of the robot's belief $p(q)$ after the measurement has been taken. As usual, probability densities in Kalman filtering are assumed to be normally distributed; therefore

$$p_1(q) = N(\hat{q}_1, \sigma_1^2), \tag{5.64}$$

$$p_2(q) = N(\hat{q}_2, \sigma_1^2). \tag{5.65}$$

According to the Bayes rule, the final distribution $p(q)$, after the measurement, is proportional to the product $p_1(q) \cdot p_2(q)$ (figure 5.31). From the product of the two density functions (5.64) and (5.65), we obtain:

$$\frac{1}{\sigma_1\sqrt{2\pi}}\exp\left(-\frac{(q-\hat{q}_1)^2}{2{\sigma_1}^2}\right)\cdot\frac{1}{\sigma_2\sqrt{2\pi}}\exp\left(-\frac{(q-\hat{q}_2)^2}{2{\sigma_2}^2}\right) = \frac{1}{\sigma_1\sigma_2 2\pi}\exp\left(-\frac{(q-\hat{q}_1)^2}{2{\sigma_1}^2}-\frac{(q-\hat{q}_2)^2}{2{\sigma_2}^2}\right). \tag{5.66}$$

As we can see, the argument of this exponential is quadratic in q, hence $p(q)$ is a Gaussian. We now need to determine its mean value $\hat{q}$ and variance σ that allow us to rewrite (5.66) in the form

$$\Omega\exp\left(-\frac{(q-\hat{q})^2}{2\sigma^2}\right). \tag{5.67}$$

27. The extended Kalman filter is the extension of the standard Kalman filter to nonlinear systems, by considering the first order approximation of the state transition function f and the observation model h.

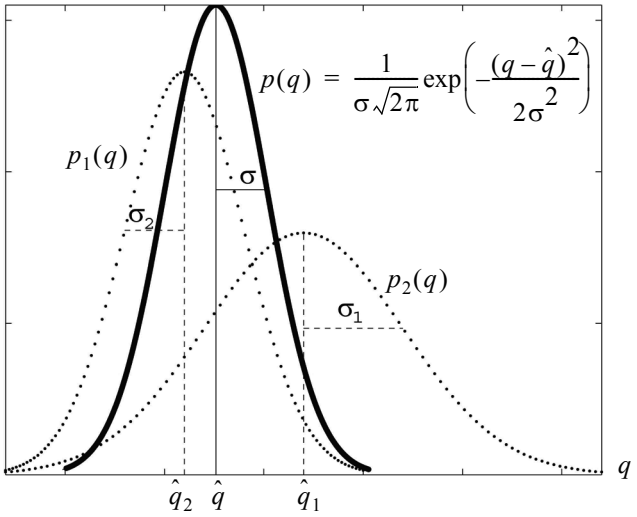


Figure 5.31
Fusing the probability density of two estimates [209]: the result of the product of two Gaussian function is another Gaussian. The result is then rescaled to make its area equal to 1.

By rearranging the exponential in (5.66), we get

$$\exp\left(-\frac{(q-\hat{q}_1)^2}{2\sigma_1^2}-\frac{(q-\hat{q}_2)^2}{2\sigma_2^2}\right) =$$

$$= \exp\left(-\frac{1}{2}\left(\frac{q^2(\sigma_1^2+\sigma_2^2)-2q(\hat{q}_1\sigma_2^2+\hat{q}_2\sigma_1^2)+(\hat{q}_1^2\sigma_2^2+\hat{q}_2^2\sigma_1^2)}{\sigma_1^2\sigma_2^2}\right)\right) =$$

$$= \exp\left(-\frac{1}{2}\left(\frac{q^2-\frac{2q(\hat{q}_1\sigma_2^2+\hat{q}_2\sigma_1^2)}{\sigma_1^2+\sigma_2^2}+\frac{\hat{q}_1^2\sigma_2^2+\hat{q}_2^2\sigma_1^2}{\sigma_1^2+\sigma_2^2}}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}}\right)\right) =$$

$$= \exp\left(-\frac{1}{2}\frac{\left(q-\frac{\hat{q}_1\sigma_2^2+\hat{q}_2\sigma_1^2}{\sigma_1^2+\sigma_2^2}\right)^2}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}}\right)\cdot\exp\left(-\frac{1}{2}\frac{\frac{\hat{q}_1^2\sigma_2^2+\hat{q}_2^2\sigma_1^2}{\sigma_1^2+\sigma_2^2}-\left(\frac{\hat{q}_1\sigma_2^2+\hat{q}_2\sigma_1^2}{\sigma_1^2+\sigma_2^2}\right)^2}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}}\right). \quad (5.68)$$

We can notice that the second term of this product depends only on $\hat{q}_1$ and $\hat{q}_2$ and, therefore, is constant. Hence, we can rewrite (5.68) as

$$\Omega \exp\left(-\frac{1}{2} \frac{\left(q - \frac{\hat{q}_1 \sigma_2^2 + \hat{q}_2 \sigma_1^2}{\sigma_1^2 + \sigma_2^2} \right)^2}{\frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 + \sigma_2^2}} \right) = \Omega \exp\left(-\frac{(q - \hat{q})^2}{2\sigma^2} \right), \tag{5.69}$$

where

$$\hat{q} = \frac{\hat{q}_1 \sigma_2^2 + \hat{q}_2 \sigma_1^2}{\sigma_1^2 + \sigma_2^2}, \text{ or alternatively } \hat{q} = \frac{\frac{1}{\sigma_1^2}\hat{q}_1 + \frac{1}{\sigma_2^2}\hat{q}_2}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}}, \tag{5.70}$$

and

$$\sigma^2 = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 + \sigma_2^2}, \text{ or alternatively } \frac{1}{\sigma^2} = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} = \frac{\sigma_1^2 + \sigma_2^2}{\sigma_1^2 \sigma_2^2}. \tag{5.71}$$

Notice that (5.70) and (5.71) can also be written as

$$\hat{q} = \hat{q}_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}(\hat{q}_2 - \hat{q}_1), \tag{5.72}$$

$$\sigma^2 = \sigma_1^2 - \frac{\sigma_1^4}{\sigma_1^2 + \sigma_2^2}. \tag{5.73}$$

These last two expressions will be valuable in the Kalman filter implementation. In Kalman filtering the factor $\sigma_1^2/(\sigma_1^2 + \sigma_2^2)$ is commonly called *Kalman gain*.

From equation (5.73) we can clearly see that the resulting variance σ^2 is smaller than both σ_1^2 and σ_2^2. Thus, the uncertainty of the position estimate has been decreased by combining the two measurements, that is, the previous robot's belief and the measurement from the exteroceptive sensor. Thus, even poor measurements will only increase the precision of an estimate. This is a result that we expect based on information theory. The solid proba-

bility density curve in figure 5.31 represents the result of the fusion operated by the Kalman filter.

Note that equations (5.72) and (5.73) are only valid for the one-dimensional case. For *n*-dimensional vectors, the final mean $\hat{q}$ and covariance $\hat{P}$ after fusion can be written respectively as:

$$\hat{q} = q_1 + P(P+R)^{-1}(q_2 - q_1) \tag{5.74}$$

$$\hat{P} = P - P(P+R)^{-1}P, \tag{5.75}$$

where P and R are the covariances of q_1 and q_2 respectively.

Equations (5.74) and (5.75) will be used in section 5.6.8.4 to implement the *measurement update* of EKF localization. In Kalman filtering these equations are usually written as

$$\hat{q} = q_1 + K(q_2 - q_1), \tag{5.76}$$

$$\hat{P} = P - K \cdot \Sigma_{IN} \cdot K^T, \tag{5.77}$$

where $K = P(P+R)^{-1}$ is the *Kalman gain*, $(q_2 - q_1)$ is the *innovation*, and $\Sigma_{IN} = (P+R)$ is the *innovation covariance*.

5.6.8.4 Application to mobile robots: Kalman filter localization

The Kalman filter is an optimal and efficient sensor fusion technique. Application of the Kalman filter to localization requires posing the robot localization problem as a sensor fusion problem. Recall that the basic probabilistic update of robot belief state can be segmented into two phases, *prediction update* and *measurement update*.

The key difference between the Kalman filter approach and our earlier Markov localization approach lies in the measurement update process. In Markov localization, the entire perception, that is, the robot's set of instantaneous sensor measurements, is used to update each possible robot position in the belief state individually. By contrast, the measurement update using a Kalman filter is a multistep process. The robot's total sensory input is treated not as a monolithic whole but as a set of extracted features that each relate to objects in the environment. Given a set of possible features, the Kalman filter is used to fuse the distance estimate from each feature to a matching object in the map. Instead of carrying out this matching process for many possible robot locations individually as in the Markov approach, the Kalman filter accomplishes the same probabilistic update by treating the whole, unimodal, and Gaussian belief state at once.

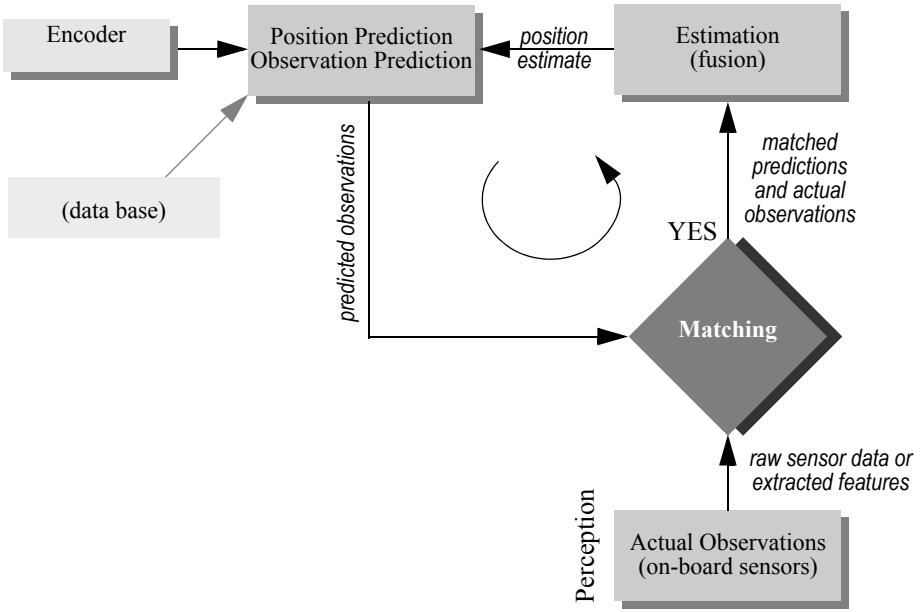


Figure 5.32
Schematic for Kalman filter mobile robot localization (see [35]).

Figure 5.32 depicts the particular schematic for Kalman filter localization. The first step is the *prediction update*, the straightforward application of a Gaussian error motion model to the robot's measured encoder travel. The *measurement update*, as just mentioned, is composed of multiple steps that are here summarized:

1. In the *observation step*, the robot collects actual sensor data and extracts appropriate features (e.g., lines, doors, or even the value of a specific sensor).

2. At the same time, based on its predicted position in the map, the robot generates a *measurement prediction* that consists in the features that the robot expects to observe from the position where it thinks it is (e.g. the position estimated in the prediction step).

3. In the *matching step* the robot computes the best matching between the features extracted during observation and the expected features selected during the measurement prediction.

4. Finally, the Kalman filter fuses the information provided by all of these matches to update the robot belief state in the *estimation step*.

In the following sections these steps are described in greater detail. The presentation is based on the work of Leonard and Durrant-Whyte [35, pages 61–65] and on that of Thrun, Burgard, and Fox [51].

Prediction update: Applying the theorem of total probability. The robot's position $\hat{x}_t$ at timestep t is predicted based on its old location at timestep $t-1$ and its movement due to the control input u_t:

$$\hat{x}_t = f(x_{t-1}, u_t) . \tag{5.78}$$

For a differential-drive robot, $f(x_{t-1}, u_t)$ is given by (5.7), which describes the odometric position estimation.

Knowing the plant and the error model, we can also compute the variance P_{t-1} associated with this prediction using the equation derived from the total probability theorem applied to Gaussian distributions (see equation [5.63]):

$$\hat{P}_t = F_x \cdot P_{t-1} \cdot F_x^T + F_u \cdot Q_t \cdot F_u^T , \tag{5.79}$$

where P_{t-1} is the covariance of the previous robot state x_{t-1} and Q_t is the covariance of the noise associated to the motion model. This equation should not surprise the reader. This is in fact nothing but the application of the error propagation law (section 4.1.3.2).

Equations (5.78) and (5.79) are the two key equations of the prediction update in EKF localization. They allow us to predict the robot's position and its uncertainty after a movement specified by the control input u_t.

Again, note that because the belief state is assumed to be Gaussian, we are just updating two values: the mean value and the covariance of the distribution. Conversely, notice that in Markov localization *all* the robot's possible states (i.e., all the cells) are updated!

Measurement update. As we said before, this phase consists of four steps:

1. Observation. The first step is to obtain sensor measurements z_t from the robot at time t. In general, the observation z_t consists of a set of n single observations z_t^i (i=0...n) extracted from the sensor. Formally, each single observation can represent an extracted feature like a point landmark, a line, or even a single, raw sensor value.

The parameters of the features are usually specified in the sensor frame and therefore in a local reference frame of the robot. However, for matching we need to represent the observations and measurement predictions in the same frame $\{S\}$. In our presentation we will transform the measurement predictions from the global world coordinate frame $\{W\}$ to the

sensor frame $\{S\}$. This transformation is specified by the function h, as discussed in section 5.6.5 when we talked about the *probabilistic measurement model*.

2. Measurement prediction. We use the robot predicted position $\hat{x}_t$ and the map M to generate multiple predicted feature observations $\hat{z}_t^j$.[28] The predicted observations are what the robot *expects* to see if it was at that particular position. Assume, for example, that, based only on the motion estimated by the odometry, the robot expects to be in front of a door. Assume now that the robot checks this hypothesis using its sensors and detects that it is actually facing a wall. Then, in this case the *door* is the predicted observation $\hat{z}_t$, while the *wall* is the actual observation z_t.

In order to compute the predicted observation, the robot must transform all the features m^j in the map M into the *local* sensor coordinate frame. If we define the transformation for the feature j through the function h^j, then we can write:

$$\hat{z}_t^j = h^j(\hat{x}_t, m^j), \tag{5.80}$$

which obviously depends on the position of each feature in the map (represented by m^j) and the current robot position $\hat{x}_t$.

3. Matching. At this point we have a set of actual observations, positioned in the sensor space, and we also have a set of predicted features, also positioned in the sensor space. The matching step has the purpose of identifying all of the single observations that match specific predicted features well enough to be used during the estimation process. In other words, we will, for a subset of the observations and a subset of the predicted features, find pairings that intuitively say "this observation is the robot's measurement of this predicted feature based on the map."

Formally, the goal of the matching procedure is to produce an assignment from the observation z_t^i to the predicted observations $\hat{z}_t^j$. For each measurement prediction for which a corresponding observation is found, we calculate the innovation v_t^{ij}. The *innovation* is a measure of the difference between the predicted and observed measurements:

$$v_t^{ij} = [z_t^i - \hat{z}_t^j] = [z_t^i - h^j(\hat{x}_t, m^j)]. \tag{5.81}$$

The *innovation covariance* $\Sigma_{IN_t}^{ij}$ can be found by applying the error propagation law (section 4.1.3.2, equation [4.15]):

28. Note that we use index j because observed and predicted features are not yet matched, that is, the observed feature i might not correspond with the feature j in the map.

$$\Sigma_{IN_t}^{ij} = H^j \cdot \hat{P}_t \cdot H^{jT} + R_t^i, \tag{5.82}$$

where H^j is the jacobian of h^j and R_t^i represents the covariance (noise) of the actual observation z_t^i.

To determine the validity of the correspondence between measurement prediction and observation, a *validation gate g* has to be specified. A possible choice for the validation gate is the Mahalanobis distance:

$$v_t^{ijT} \cdot (\Sigma_{IN_t}^{ij})^{-1} \cdot v_t^{ij} \leq g^2. \tag{5.83}$$

However, depending on the application, the sensors, and the environment models, more sophisticated validation gates might be employed.

The validation equation is used to test the observation z_t^i for membership in the validation gate for each predicted measurement. When a single observation falls in the validation gate, we get a successful match. If one observation falls in multiple validation gates, the best matching candidate is selected or multiple hypotheses are tracked. Observations that do not fall in the validation gate are simply ignored for localization. Such observations could have resulted from objects not in the map, such as new objects (e.g., someone places a large box in the hallway) or transient objects (e.g., humans standing next to the robot may form a line feature). One approach is to take advantage of such unmatched observations to populate the robot's map.

4. Estimation: Applying the Bayes rule. In this step, we compute the best estimate x_t of the robot's position based on the position prediction $\hat{x}_t$ and all the observations z_t^i at time t. To do this position update, we first stack the validated observations z_t^i into a single vector to form z_t and designate the composite innovation v_t. Then, we stack the measurement Jacobians H^j for each validated measurement together to form the composite Jacobian H and the measurement error (noise) vector $R_t = diag[R_t^i]$. From these, we can then compute the composite innovation covariance Σ_{IN_t} using equation (5.82). Finally, by using the results from the application of the Bayes rule to Gaussian distributions, equations (5.74) and (5.75), we can update the robot's position estimate x_t and its associated covariance P_t as

$$x_t = \hat{x}_t + K_t v_t, \tag{5.84}$$

$$P_t = \hat{P}_t - K_t \cdot \Sigma_{IN_t} \cdot K_t^T, \tag{5.85}$$

where

$$K_t = \hat{P}_t \cdot H_t^T \cdot (\Sigma_{IN_t})^{-1} \tag{5.86}$$

is the Kalman gain.

As an exercise, the reader can verify that when the h is an identity (5.84) and (5.85) reduce exactly to equations (5.74) and (5.75). Indeed, by imposing H equal to the identity matrix, equation (5.84) simplifies to

$$x_t = \hat{x}_t + \hat{P}_t(\hat{P}_t + R_t)^{-1}(z_t - \hat{x}_t) \tag{5.87}$$

$$P_t = \hat{P}_t - \hat{P}_t(\hat{P}_t + R_t)^{-1}\hat{P}_t, \tag{5.88}$$

which correspond respectively to (5.74) and (5.75).

Equation (5.84) says that the best estimate x_t of the robot state at time t is equal to the best prediction of the value $\hat{x}_t$ before the new measurement z_t is taken, plus a correction term of an optimal weighting value K_t times the difference between z_t and the best prediction $\hat{z}_t$ at time t.

The new, fused estimate of the robot position is again subject to a Gaussian probability density curve. Its mean and covariance are simply functions of two inputs, mean and covariance. Thus the Kalman filter provides both a compact, simplified representation of uncertainty and an extremely efficient technique for combining heterogeneous estimates to yield a new estimate for our robot's position.

In the next section, we will implement a Kalman filter localization algorithm for a differential drive robot.

5.6.8.5 Case study: Kalman filter localization with line feature extraction

The Pygmalion robot at the EPFL is a differential-drive robot that uses a laser rangefinder as its primary sensor [59, 60]. In contrast to Dervish, the environmental representation of Pygmalion is continuous and abstract: the map consists of a set of infinite lines describing the environment. Pygmalion's belief state is, of course, represented as a Gaussian distribution since this robot uses the Kalman filter localization algorithm. The value of its mean position x_t is represented to a high level of precision, enabling Pygmalion to localize with very high precision when desired. We next present details for Pygmalion's implementation of the Kalman filter localization steps. For simplicity we assume that the sensor frame $\{S\}$ is equal to the robot frame $\{R\}$. If not specified, all the vectors are represented in the world coordinate system $\{W\}$.

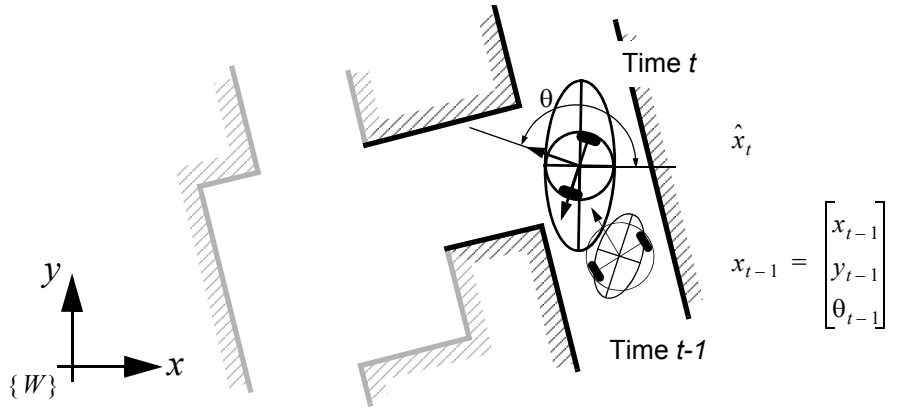


Figure 5.33
Prediction of the robot's position (thick) based on its former position (thin) and the executed movement. The ellipses drawn around the robot positions represent the uncertainties in the *x,y* direction (e.g., 3σ). The uncertainty of the orientation θ is not represented in the picture.

1. Robot position prediction. Suppose that at time $t-1$ the robot best position estimate is $x_{t-1} = [x_{t-1}, y_{t-1}, \theta_{t-1}]^T$. The control input u_t drives the robot to the position $\hat{x}_t$ (figure 5.33).

The robot position prediction $\hat{x}_t$ at time t can be computed from the previous estimate x_{t-1} and the odometric integration of the movement. For the differential drive that Pygmalion has, we can use the odometry model developed in section 5.2.4:

$$\hat{x}_t = f(x_{t-1}, u_t) = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} \cos\left(\theta_{t-1} + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r + \Delta s_l}{2} \sin\left(\theta_{t-1} + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r - \Delta s_l}{b} \end{bmatrix}, \tag{5.89}$$

where Δs_l, Δs_r characterize the displacement of the left and right wheel. Therefore, the control input is exactly $u_t = [\Delta s_l, \Delta s_r]^T$.

The updated covariance matrix is

$$\hat{P}_t = F_x \cdot P_{t-1} \cdot F_x^T + F_u \cdot Q_t \cdot F_u^T, \tag{5.90}$$

where P_{t-1} is the covariance of the previous robot state x_{t-1} and Q_t is the covariance of the noise associated to the motion model (see equation [5.8]), that is,

$$Q_t = \begin{bmatrix} k_r|\Delta s_r| & 0 \\ 0 & k_l|\Delta s_l| \end{bmatrix}. \tag{5.91}$$

2. Observation. For line-based localization, each single observation (i.e., a line feature) is extracted from the raw laser rangefinder data and consists of the two line parameters α_t^i, r_t^i (figure 4.88), because for a rotating laser rangefinder a representation in the polar coordinate frame is more appropriate:

$$z_t^i = \begin{bmatrix} \alpha_t^i \\ r_t^i \end{bmatrix}. \tag{5.92}$$

After acquiring the raw data at time *t*, lines and their uncertainties are extracted (figure 5.34a–b. This leads to n observed lines with $2n$ line parameters (figure 5.34c) and a covariance matrix R_t^i for each line that can be calculated from the uncertainties of all the measurement points contributing to each line as developed for line extraction in section 4.7.1:

$$R_t^i = \begin{bmatrix} \sigma_{\alpha\alpha} & \sigma_{\alpha r} \\ \sigma_{r\alpha} & \sigma_{rr} \end{bmatrix}_t^i. \tag{5.93}$$

3. Measurement prediction. Based on the stored map and the predicted robot position $\hat{x}_t$, the measurement predictions $\hat{z}_t^j$ of the expected features are generated (figure 5.35).[29] These features are stored in the map *M* and specified in the world coordinate system $\{W\}$. In order to compute the predicted observation, the robot must transform all the line features m^j in the map M into its local robot coordinate frame $\{R\}$. According to figure 5.35, the transformation is given by

29. To reduce the required calculation power, there is often an additional step that first selects the possible features, in this case lines, from the whole set of features in the map.

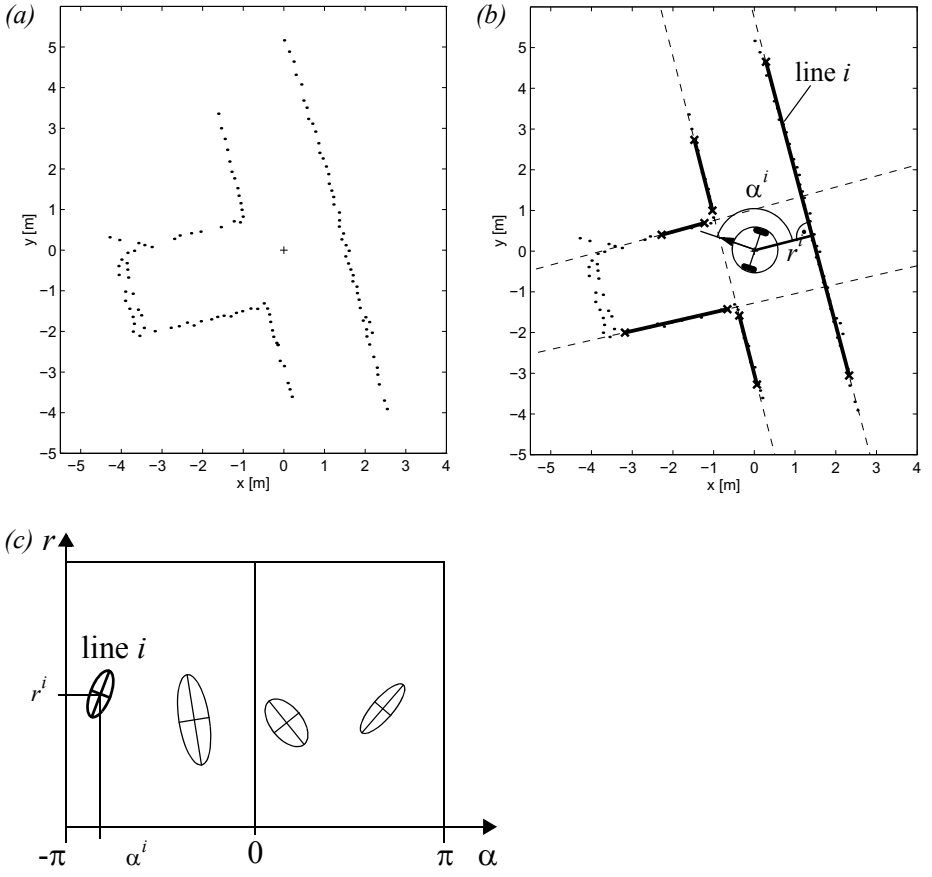


Figure 5.34
Observation: From the raw data (a) acquired by the laser scanner at time t, lines are extracted (b). The line parameters α^i and r^i and its uncertainties can be represented in the model space (c).

$$\hat{z}_t^j = \begin{bmatrix} \hat{\alpha}_t^j \\ \hat{r}_t^j \end{bmatrix} = h^j(\hat{x}_t, m^j) = \begin{bmatrix} {}^{\{W\}}\alpha_t^j - \hat{\theta}_t \\ {}^{\{W\}}r_t^j - (\hat{x}_t \cos({}^{\{W\}}\alpha_t^j) + \hat{y}_t \sin({}^{\{W\}}\alpha_t^j)) \end{bmatrix}, \tag{5.94}$$

and its Jacobian H^j by

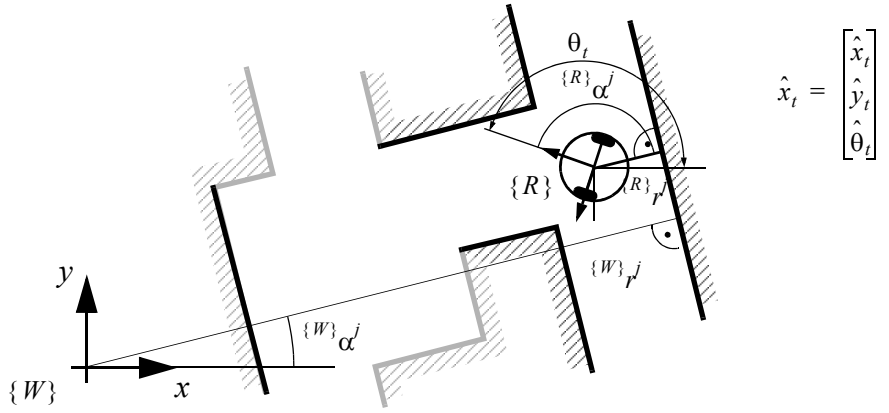


Figure 5.35
Representation of the target position in the world coordinate frame $\{W\}$ and robot coordinate frame $\{R\}$.

$$H^j = \begin{bmatrix} \frac{\partial \alpha_t^j}{\partial \hat{x}} & \frac{\partial \alpha_t^j}{\partial \hat{y}} & \frac{\partial \alpha_t^j}{\partial \hat{\theta}} \\ \frac{\partial r_t^j}{\partial \hat{x}} & \frac{\partial r_t^j}{\partial \hat{y}} & \frac{\partial r_t^j}{\partial \hat{\theta}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 \\ -\cos(^{\{W\}}\alpha_t^j) & -\sin(^{\{W\}}\alpha_t^j) & 0 \end{bmatrix}, \tag{5.95}$$

where we used

$$m^j = {}^{\{W\}}\begin{bmatrix} \alpha_t^j \\ r_t^j \end{bmatrix}. \tag{5.96}$$

The measurement prediction results in predicted lines represented in the robot coordinate frame (figure 5.36). They are uncertain, because the prediction of robot position is uncertain.

4. Matching. For matching, we must find correspondence (or a pairing) between predicted and observed features (figure 5.37). In our case we take the Mahalanobis distance

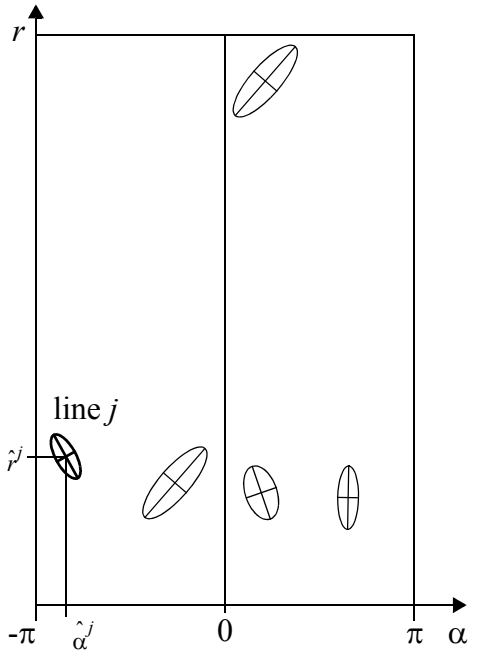


Figure 5.36
Measurement predictions: Based on the and the estimated robot position the targets (visible lines) are predicted. They are represented in the model space similar to the observations.

$$v_t^{ijT} \cdot (\Sigma_{IN_t}^{ij})^{-1} \cdot v_t^{ij} \leq g^2 \tag{5.97}$$

with

$$\begin{aligned} v_t^{ij} &= [z_t^i - \hat{z}_t^j] = [z_t^i - h^j(\hat{x}_t, m^j)] = \\ &= \begin{bmatrix} \alpha_t^i \\ r_t^i \end{bmatrix} - \begin{bmatrix} {}^{\{W\}}\alpha_t^j - \hat{\theta}_t \\ {}^{\{W\}}r_t^j - (\hat{x}_t \cos({}^{\{W\}}\alpha_t^j) + \hat{y}_t \sin({}^{\{W\}}\alpha_t^j)) \end{bmatrix}, \end{aligned} \tag{5.98}$$

$$\Sigma_{IN_t}^{ij} = H^j \cdot \hat{P}_t \cdot H^{jT} + R_t^i \tag{5.99}$$

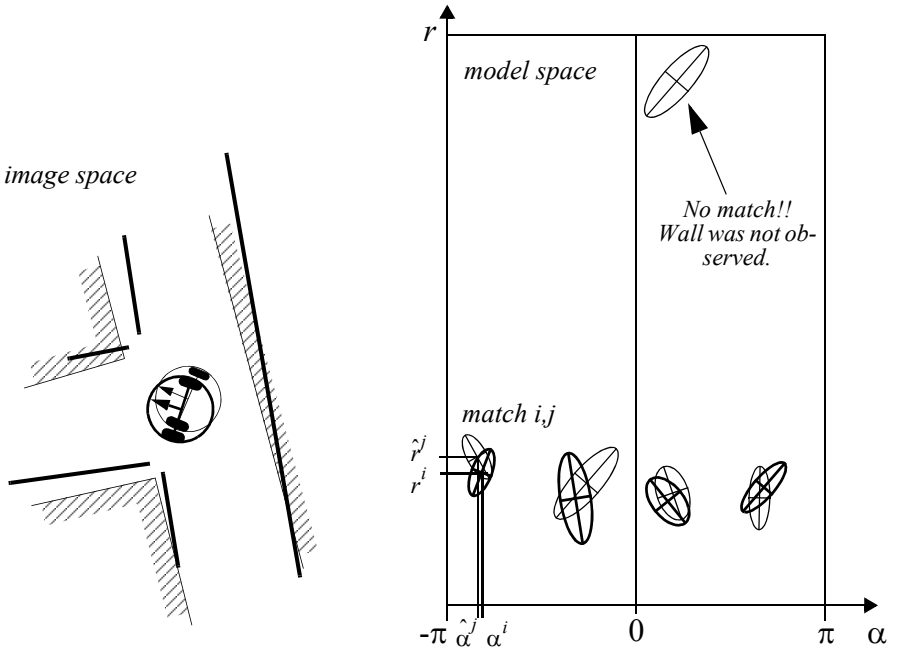


Figure 5.37
Matching: The observations (thick) and the predicted observation (thin) are matched and the innovation and its uncertainties are calculated.

to enable finding the best matches while eliminating all other remaining observed and predicted unmatched features.

5. Estimation. Applying the Kalman filter results in a final pose estimate corresponding to the weighted sum of (figure 5.38);

- the pose estimates of each matched pairing of observed and predicted features;
- the robot position estimation based on odometry and observation positions.

5.7 Other Examples of Localization Systems

Markov localization and Kalman filter localization have been two extremely popular strategies for research mobile robot systems navigating indoor environments. They have strong formal bases and therefore well-defined behavior. But there are other probabilistic localiza-

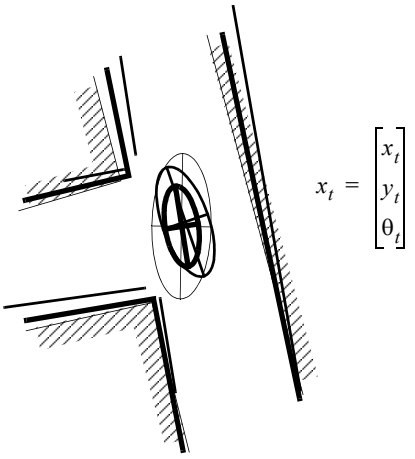


Figure 5.38
Kalman filter estimation of the new robot position: By fusing the prediction of robot position (thin) with the innovation gained by the measurements (thick) we get the updated estimate x_t of the robot position (very thick).

tion techniques that have been used with varying degrees of success on commercial and research mobile robot platforms. Some techniques that deserve mention are Unscented Kalman Filter (UKF) localization, grid localization, and Monte Carlo localization. UKF is similar to EKF in that it also assumes Gaussian distributions but relies on a different way to linearize the motion and measurements models, which is called the *unscented transform*. Conversely, grid localization and Monte Carlo localization are not limited to unimodal distributions. While grid localization uses the so-called histogram filter to represent the robot belief, Monte Carlo localization uses particle filters. The latter is probably the most popular localization algorithm (this was already introduced on page 315). Because a description of these techniques goes beyond the scope of this book, we refer the reader to [51] for such information.

There are, however, several categories of localization techniques that deserve mention. Not surprisingly, many implementations of these techniques in commercial robotics employ modifications of the robot's environment, something that the Markov localization and Kalman filter localization communities eschew. In the following sections, we briefly identify the general strategy incorporated by each category and reference example systems, including, as appropriate, those that modify the environment and those that function without environmental modification.

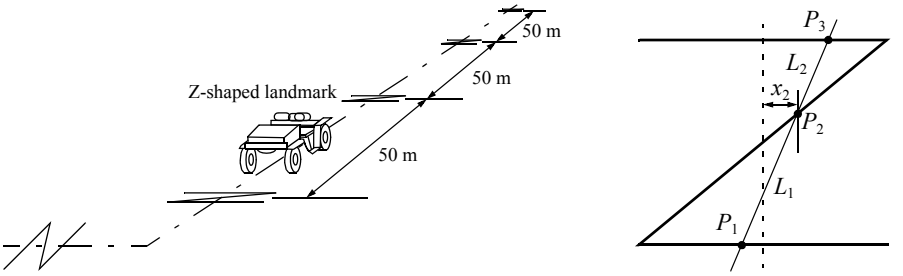


Figure 5.39
Z-shaped landmarks on the ground. Komatsu Ltd., Japan [6, pages 179-180].

5.7.1 Landmark-based navigation

Landmarks are generally defined as passive objects in the environment that provide a high degree of localization accuracy when they are within the robot's field of view. Mobile robots that make use of landmarks for localization generally use artificial markers that have been placed by the robot's designers to make localization easy.

The control system for a landmark-based navigator consists of two discrete phases. When a landmark is in view, the robot localizes frequently and accurately, using action update and perception update to track its position without cumulative error. But when the robot is in a no-landmark "zone," then only action update occurs, and the robot accumulates position uncertainty until the next landmark enters the robot's field of view.

The robot is thus effectively *dead-reckoning* from landmark zone to landmark zone. This in turn means the robot must consult its map carefully, ensuring that each motion between landmarks is sufficiently short, given its motion model, that it will be able to localize successfully upon reaching the next landmark.

Figure 5.39 shows one instantiation of landmark-based localization. The particular shape of the landmarks enables reliable and accurate pose estimation by the robot, which must travel using *dead reckoning* between the landmarks.

One key advantage of the landmark-based navigation approach is that a strong formal theory has been developed for this general system architecture [187]. In this work, the authors have shown precise assumptions and conditions which, when satisfied, guarantee that the robot will always be able to localize successfully. This work also led to a real-world demonstration of landmark-based localization. Standard sheets of paper were placed on the ceiling of the Robotics Laboratory at Stanford University, each with a unique checkerboard pattern. A Nomadics 200 mobile robot was fitted with a monochrome CCD camera aimed vertically up at the ceiling. By recognizing the paper landmarks, which were placed approx-

imately 2 m apart, the robot was able to localize to within several centimeters, then move, using dead reckoning, to another landmark zone.

The primary disadvantage of landmark-based navigation is that in general it requires significant environmental modification. Landmarks are local, and therefore a large number are usually required to cover a large factory area or research laboratory. For example, the Robotics Laboratory at Stanford made use of approximately thirty discrete landmarks, all affixed individually to the ceiling.

5.7.2 Globally unique localization

The landmark-based navigation approach makes a strong general assumption: when the landmark is in the robot's field of view, localization is essentially perfect. One way to reach the Holy Grail of mobile robotic localization is effectively to enable such an assumption to be valid no matter *where* the robot is located. It would be revolutionary if a look at the robot's sensors immediately identified its particular location, uniquely and repeatedly.

Such a strategy for localization is surely aggressive, but the question of whether it can be done is primarily a question of sensor technology and sensing software. Clearly, such a localization system would need to use a sensor that collects a very large amount of information. Since vision does indeed collect far more information than previous sensors, it has been used as the sensor of choice in research toward globally unique localization. If humans were able to look at an individual picture and identify the robot's location in a well-known environment, then one could argue that the information for globally unique localization does exist within the picture; it must simply be teased out. As described in section 4.6, an important milestone toward this direction has been achieved with "bag of features" approaches, where the current image is first converted into a "bag" of distinctive local features (section 4.5) which are then used to find the most similar images in a dataset of million of pictures in less than one second. This approach successfully demonstrated robust localization on a more than 1,000 km trajectory using purely images collected from vehicle-mounted camera [108,109].

If one would like to use laser scans instead of camera images, then the angular histogram depicted in figure 4.95 of the previous chapter is another example in which the robot's laser sensor values are transformed into an identifier of location. In this case, the identifier is a histogram instead of a bag of features. However, due to the limited information content of laser scans, it is likely that two *places* in the robot's environment may have angular histograms that are too similar to be differentiated successfully. Therefore, image-based localization should be preferred for large-scale environments, since images provide better globally unique localization than laser-based strategies.

The key advantage of globally unique localization is that, when these systems function correctly, they greatly simplify robot navigation. The robot can move to any point and will always be assured of localizing by collecting a sensor scan.

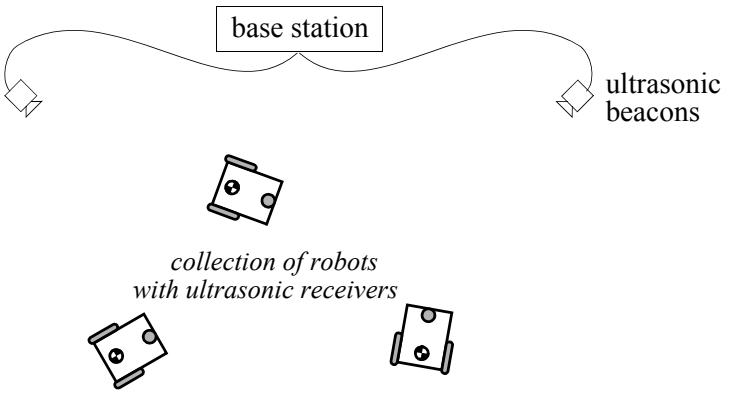


Figure 5.40
Active ultrasonic beacons.

But the main disadvantage of globally unique localization is that it is likely that this method will *never* offer a complete solution to the localization problem. There will always be cases where local sensory information is truly ambiguous, and globally unique localization using only current sensor information is therefore unlikely to succeed (e.g., in a forest!). Humans often have excellent *local positioning systems*, particularly in nonrepeating and well-known environments such as their homes. However, there are a number of environments in which such immediate localization is challenging even for humans: consider hedge mazes and large new office buildings with repeating halls that are identical.

5.7.3 Positioning beacon systems

One of the most reliable solutions to the localization problem is to design and deploy an active beacon system specifically for the target environment. This is the preferred technique used by both industry and military applications as a way of ensuring the highest possible reliability of localization. The GPS system can be considered as just such a system (see section 4.1.8.1).

Figure 5.40 depicts one such beacon arrangement for a collection of robots. Just as with GPS, by designing a system whereby the robots localize passively while the beacons are active, any number of robots can simultaneously take advantage of a single beacon system. As with most beacon systems, the design depicted depends foremost upon geometric principles to effect localization. In this case the robots must know the positions of the two active ultrasonic beacons in the global coordinate frame in order to localize themselves to the global coordinate frame.

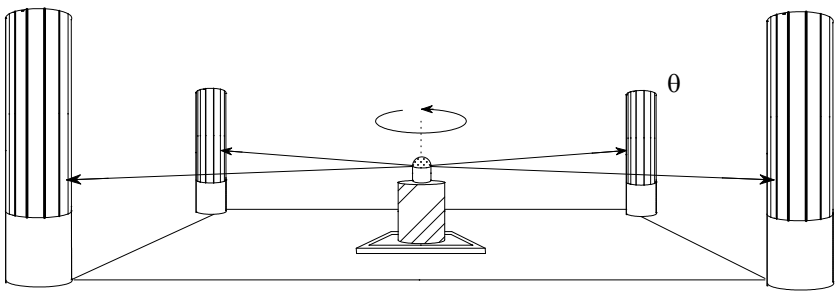


Figure 5.41
Passive optical beacons.

A popular type of beacon system in industrial robotic applications is depicted in figure 5.41. In this case, beacons are retroreflective markers that can be easily detected by a mobile robot based on their reflection of energy back to the robot. Given known positions for the optical retroreflectors, a mobile robot can identify its position whenever it has three such beacons in sight simultaneously. Of course, a robot with encoders can localize over time as well, and it does not need to measure its angle to all three beacons at the same instant.

The advantage of such beacon-based systems is usually extremely high engineered reliability. By the same token, significant engineering usually surrounds the installation of such a system in a specific commercial setting. Therefore, moving the robot to a different factory floor will be both time-consuming and expensive. Usually, even changing the routes used by the robot will require serious reengineering.

5.7.4 Route-based localization

Even more reliable than beacon-based systems are route-based localization strategies. In this case, the route of the robot is explicitly marked so that it can determine its position, not relative to some global coordinate frame but relative to the specific path it is allowed to travel. There are many techniques for marking such a route and the subsequent intersections. In all cases, one is effectively creating a railway system, except that the railway system is somewhat more flexible and certainly more human-friendly than a physical rail. For example, high ultraviolet-reflective, optically transparent paint can mark the route such that only the robot, using a specialized sensor, easily detects it. Alternatively, a guidewire buried underneath the hall can be detected using inductive coils located on the robot chassis.

In all such cases, the robot localization problem is effectively trivialized by forcing the robot to always follow a prescribed path. To be fair, there are new industrial *unmanned*

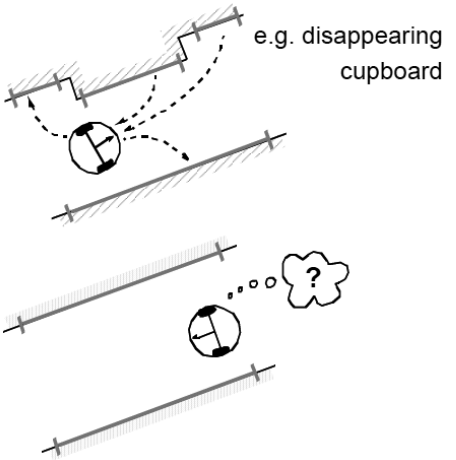


Figure 5.42 An autonomous robot should be able to track changes in the environments for localizing.

guided vehicles that do deviate briefly from their route in order to avoid obstacles. Nevertheless, the cost of this extreme reliability is obvious: the robot is much more inflexible given such localization means, and therefore any change to the robot's behavior requires significant engineering and time.

5.8 Autonomous Map Building

5.8.1 Introduction

All the localization strategies that we have discussed so far require the existence of a map of the environment. The map is normally built by hand. This means that, for accurate localization, the position of the landmarks (e.g., walls, artificial beacons, etc.) that the robot uses for self-localizing must be accurately measured and included in the map. Unfortunately, this approach can be hard, costly, and very time-consuming when the size of the environment is large or when the environment changes due to artificial modifications or dynamic objects. Assume, for instance, a domestic robot that is supposed to work in home environments. In this case the robot should be able to detect changes in the map due to the rearrangement of the furniture (figure 5.42). Another drawback of handmade maps is that the look of the map can be different depending on the different perception of who makes the map.

The alternative to handmade map building is therefore "automatic map building." Indeed, a robot that localizes successfully has the right sensors for detecting the environment, and so the robot ought to build its own map. This ambition goes to the heart of autonomous mobile robotics. In prose, we can express our eventual goal as follows: *starting from*

an arbitrary initial point, a mobile robot should be able to explore autonomously the environment with its on-board sensors, gain knowledge about it, interpret the scene, build an appropriate map, and localize itself relative to this map.

The recent advances in both robotics and computer vision have made this goal somewhat achieved. An important subgoal has been the invention of techniques for place recognition and for autonomous creation and modification of an environmental map. Of course a mobile robot's sensors have only a limited range, and so the robot must physically explore its environment to build such a map. So, the robot must not only create a map, but it must also do this while moving and localizing to explore the environment. In the robotics community, this is often called the Simultaneous Localization and Mapping (SLAM) problem. The relevance of the SLAM problem for the robotics community owes to the fact that the solution to this problem would make a robot truly autonomous.

After a short introduction to the simultaneous localization and mapping problem (section 5.8.2), we will review three major algorithms from which a large number of published methods have been derived. First, we will review the traditional approach, which is based on the extended Kalman filter (section 5.8.4). As an application of one of these methods, we will present the Visual-SLAM algorithm (section 5.8.5), which uses a single camera as only sensor. Second, we will review the Graph-SLAM algorithm (section 5.8.7), which is born from the intuition that the SLAM problem can be interpreted as a sparse graph of constraints. Third, we will review the particle-filter SLAM (section 5.8.8). Finally, we will discuss open problems in SLAM (section 5.8.9).

It is important to point out that none of these method is the favorite solution to the SLAM problem. The choice of the right method will depend on the number and type of features in the environments, the resolution of the desired map, the computational time, and so on.

For an in-depth study of SLAM algorithms, we refer the reader to [51]. Up-to-date references and on-line software can instead be found in these tutorials [62, 120, 313]. In the following sections, we will keep the same notation as in the section on localization, which is also the same as in [51].

5.8.2 SLAM: The simultaneous localization and mapping problem

As we have seen in section 5.6.2, localization is the problem of estimating the robot position (and therefore its path) given a known map of the environment. Conversely, mapping is the construction of the map of the environment knowing the true path of the robot. The aim of SLAM is to recover both the robot path and the environment map using only the data gathered by its proprioceptive and exteroceptive sensors. These data are typically the robot displacement estimated from the odometry and features (e.g., corners, lines, planes) extracted from laser, ultrasonic, or camera images.

SLAM is difficult because both the estimated path and the extracted features are corrupted by noise. The problem is illustrated in figure 5.43. Let us assume that the robot

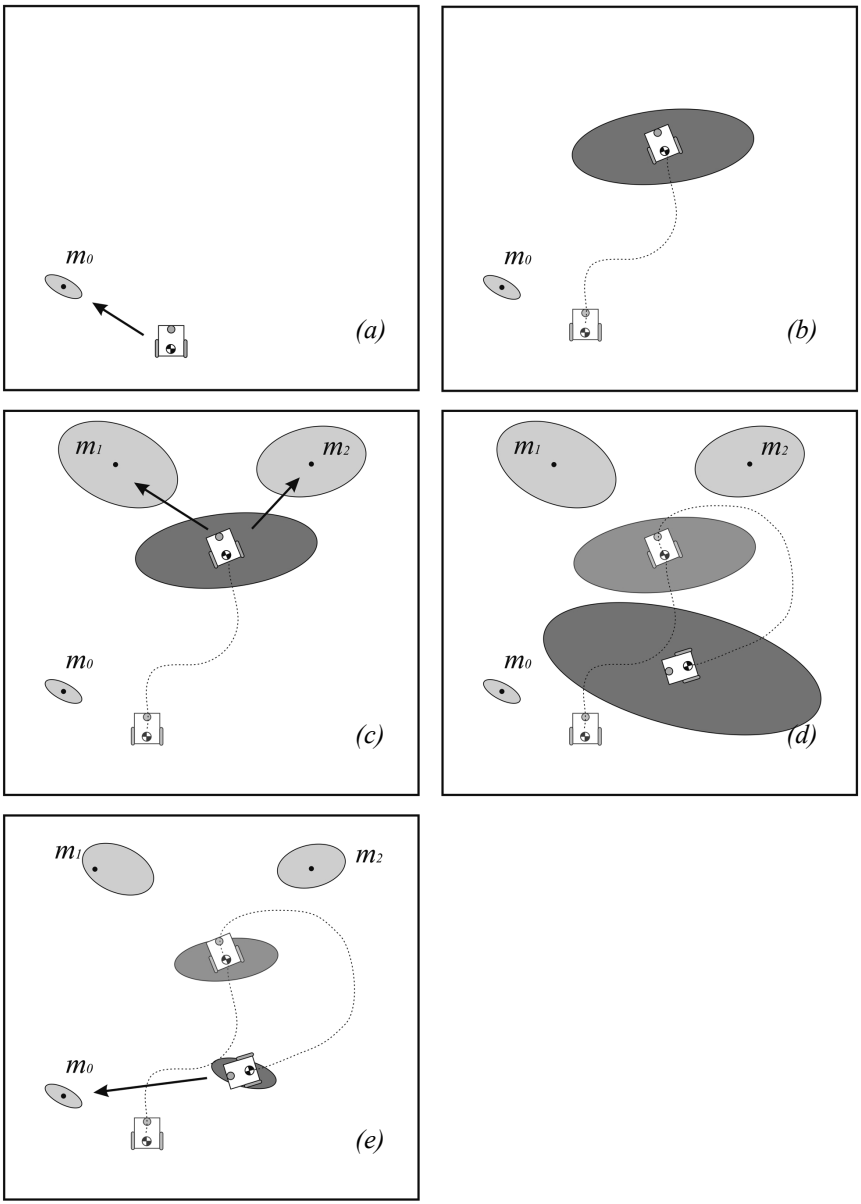


Figure 5.43 Illustration of the SLAM problem.

uncertainty at its initial location is zero. From this position, the robot observes a feature which is mapped with an uncertainty related to the exteroceptive sensor error model (a). As the robot moves, its pose uncertainty increases under the effect of the errors introduced by the odometry (b). At this point, the robot observes two features and maps them with an uncertainty that results from the combination of the measurement error with the robot pose uncertainty (c). From this, we can notice that the map becomes correlated with the robot position estimate. Similarly, if the robot updates its position based on an observation of an imprecisely known feature in the map, the resulting position estimate becomes correlated with the feature location estimate. In order to reduce its uncertainty, the robot must observe features whose location is relatively well known. These features can, for instance, be landmarks that the robot has already observed before. In this case, the observation is called *loop closure detection*. When a loop closure is detected, the robot pose uncertainty shrinks. At the same time, the map is updated and the uncertainty of other observed features and all previous robot poses also reduce (figure 5.43e).

The general problem of map-building is thus an example of the chicken-and-egg problem. For localization the robot needs to know where the features are, whereas for map-building the robot needs to know where it is on the map.

5.8.3 Mathematical definition of SLAM

As we already did for map-based localization, we also describe SLAM in probabilistic terminology. The terminology used in the section is the same introduced in sections 5.6.4 and 5.6.5 for probabilistic map-based localization. The reader may take a moment to review those sections before proceeding.

Let us recall that we define the robot pose at time t by x_t. The robot path is given as

$$X_T = \{x_0, x_1, x_2, \ldots, x_T\}, \tag{5.100}$$

where T might also be infinite. In SLAM, the robot initial location x_0 is assumed to be known, while the others locations are not.

Let u_t denote the robot motion between time $t-1$ and time t. Let us recall that these data can be the proprioceptive sensor readings (e.g., from the robot's wheel encoders) or the control inputs given to the motors. The sequence of the robot relative motions can then be written as:

$$U_T = \{u_0, u_1, u_2, \ldots, u_T\}. \tag{5.101}$$

Let M denote the *true* map of the environment

$$M = \{m_0, m_1, m_2, \ldots, m_{n-1}\}, \tag{5.102}$$

then m_i, $i = 0 \ldots n-1$, are vectors representing the positions of the landmarks that, again, might be points, lines, planes, or any sort of high-level feature (e.g., doors). Observe that, for simplicity, we assume that the map is static.

Finally, if we assume that the robot takes one measurement at each time, we can denote by

$$Z_T = \{z_0, z_1, z_2, \ldots, z_T\} \tag{5.103}$$

the sequence of landmark observations in the sensor reference frame attached to the robot. For example, if the robot is equipped with an on-board camera, the observation z_i can be a vector representing the coordinates of a corner or those of a line in the image. If, instead, the robot is equipped with a laser rangefinder, such a vector can represent the position of a corner or a line in the laser sensor frame.

According to this terminology, we can now define SLAM as the problem of recovering a model of the map M and the robot path X_T from the odometry U_T and observations Z_T. In the literature, we distinguish between the *full* SLAM problem and the *online* SLAM problem. The full SLAM problem consists in estimating the joint posterior probability over X_T and M from the data, that is

$$p(X_T, M | Z_T, U_T) . \tag{5.104}$$

The online SLAM problem, conversely, consists in estimating the joint posterior over x_t and M from the data, that is

$$p(x_t, M | Z_T, U_T) . \tag{5.105}$$

Therefore, the full SLAM problem tries to recover the entire robot path X_T, while the online SLAM problem tries to estimates only the current robot pose x_t.

In order to solve the SLAM problem, we need to know the probabilistic motion model and probabilistic measurement model. These models have been introduced in section 5.6.5. In particular, let us recall that

$$p(x_t | x_{t-1}, u_t) \tag{5.106}$$

represents the probability that the robot pose is x_t given the robot previous pose x_{t-1} and proprioceptive data (or control input) u_t. Similarly, let us recall that

$$p(z_t | x_t, M) \tag{5.107}$$

is the probability of measuring z_t given the known map M and assuming that the robot takes the observation at location x_t.

We encourage the reader to take a moment to review these concepts in section 5.6.5.

In the next sections, we will describe the three main paradigms developed over the last two decades to solve the SLAM problem, which are EKF SLAM, graph-based SLAM, and particle filter SLAM. From these paradigms, many other algorithms have been derived. For an in-depth study of these algorithms, we refer the reader to [51].

5.8.4 Extended Kalman Filter (EKF) SLAM

In this section, we will see the application of the EKF to the online SLAM problem. EKF-based SLAM is historically the first formulation proposed and was introduced in several papers [100, 294, 295, 228, 229].

The EKF SLAM proceeds exactly like the standard EKF that we have seen for robot localization (section 5.6.8), with the only difference that it uses an extended state vector y_t which comprises both the robot pose x_t and the position of all the features m_i in the map, that is:

$$y_t = [x_t, m_0, \ldots, m_{n-1}]^T. \tag{5.108}$$

In our localization example based on line features (section 5.6.8.5), the dimension of y_t would be 3+2n, since we need three variables to represent the robot pose (x, y, θ) and 2n variables for the n line-landmarks having vector components (α^i, r^i). Therefore, the state vector would be written as

$$y_t = [x_t, y_t, \theta_t, \alpha_0, r_0, \alpha_1, r_1, \ldots, \alpha_{n-1}, r_{n-1}]^T. \tag{5.109}$$

As the robot moves and takes measurements, the state vector and covariance matrix are updated using the standard equations of the extended Kalman filter. Clearly, the state vector in EKF SLAM is much larger that the state vector in EKF localization where only the robot pose was being updated. This makes EKF SLAM computationally much more expensive.

Notice that, because of its formulation, maps in EKF SLAM are supposed to be feature-based (i.e., points, lines, planes). As new features are observed, they are added to the state vector. Thus, the noise covariance matrix grows quadratically, with size $(3 + 2n) \times (3 + 2n)$. For computational reasons, the size of the map is therefore usually limited to less than a thousand features. However, numerous approaches have been developed to cope with a larger number of features, which decompose the map into smaller submaps, for which covariances are updated separately [63].

As we mentioned, the implementation of the EKF SLAM is nothing but the straightforward application of the EKF equations to the online SLAM problem, that is, equations

(5.78)–(5.79) and (5.84)–(5.85). In order to do this, we need to specify the functions that characterize the prediction and measurement model. If we use again our line-based localization example of section 5.6.8.5, the measurement model is then the same as in equation (5.94). The prediction model, conversely, has to take into account that the motion will only update the robot pose according to (5.89), while the features will remain unchanged. Therefore, we can write the prediction model of the EKF SLAM as

$$\hat{y}_t = y_t + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} \cos\left(\theta_{t-1} + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r + \Delta s_l}{2} \sin\left(\theta_{t-1} + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r - \Delta s_l}{b} \\ 0 \\ 0 \\ \cdots \\ 0 \\ 0 \end{bmatrix}. \tag{5.110}$$

At the start, when the robot takes the first measurements, the covariance matrix is populated by assuming that these (initial) features are uncorrelated, which implies that the off diagonal elements are set to zero. However, when the robot starts moving and takes new measurements, both the robot pose and features start becoming correlated. Accordingly, the covariance matrix becomes *nonsparse.*[30] The existence of this correlation can be explained by recalling that the uncertainty of the features in the map depends on the uncertainty associated to the robot pose. But it also depends on the uncertainty of other features that have been used to update the robot pose. This means that when a new feature is observed, this contributes to correct not only the estimate of the robot pose but also that of the other features as well. The more observations are made, the more the correlations between the features will grow. Therefore, the correlations between the features—and so the fact that the covariance matrix is nonsparse—are of significant important in SLAM [105]: the bigger these correlations, the better the solution of the SLAM.

Figure 5.43 illustrates the working principle of EKF SLAM in a simple environment with three features. The robot initial location is assumed as the origin of the system reference frame and therefore the initial uncertainty of the robot pose is zero. From this position, the robot observes a feature and maps it with an uncertainty related to the sensor error

30. In numerical analysis, a sparse matrix is a matrix populated primarily with zeros.

model (a). As the robot moves, its pose uncertainty increases under the effect of the errors introduced by the odometry (b). At some point, the robot observes two features and maps them with an uncertainty which results from the combination of the measurement error with the robot pose uncertainty (c). From this, we can notice that the map becomes correlated with the robot position estimate. Now, the robot drives back toward its starting position and its pose uncertainty increases again (d). At this point, it reobserves the first feature, whose location is relatively well known compared to the other features. This makes the robot more sure about its current location and therefore its pose uncertainty shrinks (e). Notice that so far we only considered the online SLAM problem. Therefore, only the robot current position was being updated. The full SLAM problem, conversely, updates the entire robot path and thus all its previous poses. In this case, after reobserving the first feature, also the robot previous pose uncertainty will shrink and so will also the uncertainties associated to the other features. The position of these features is in fact correlated with the robot previous poses.

EKF SLAM has been successfully applied in many different domains, including airborne, underwater, outdoor, and indoor environments. Figure 5.44 shows results of a 6DoF SLAM using a 3D laser rangefinder. The robot starts at the center and makes three rounds. Figure 5.44a shows the resulting map using only odometry. As you can see, the map is inconsistent (the scans are not aligned) due to the accumulated odometry drift. In (b), the accumulated odometry error is drastically reduced by using scan matching and alignment techniques.[31] Finally, in (c), the accumulated drift and the offset error are no longer present after application of EKF SLAM. Notice that in this particular application, horizontal and vertical planes have been used as features.

The basic formulation of EKF SLAM assumes that the position of the features is fully measurable from a single robot location. This is because most SLAM applications have been realized using rangefinders (i.e., lasers, sonars, or stereo cameras) that provide both range and bearing information about the features. However, there are situations where either the range [190] or the bearing information (the angle) is available. The latter occurs, for example, when using a single camera. As seen in section 4.2.3, a calibrated camera is a bearing sensor (figure 4.31). In this case, the SLAM problem is usually called *monocular Visual SLAM* or *bearing-only SLAM* [110, 227]. In this case, the standard EKF can still be applied, as we will see in the next section.

31. One of the most popular techniques for aligning two different laser scans is the Iterative Closest Point (ICP) algorithm [72]. This, however, works well only if the relative motion between the two scans is known with good approximation (for instance from the odometry) otherwise other global optimization techniques are required [97,204].

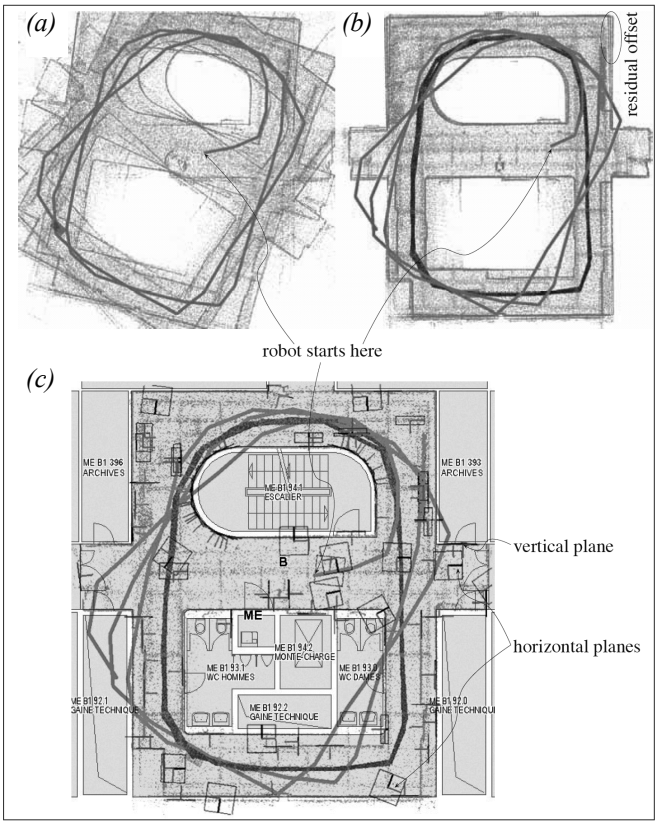


Figure 5.44 EKF SLAM using 3D laser scanner. (a) The robot starts at the center and makes three rounds. (a) Aligned 3D scans using odometry only leading to an inconsistent map. (b) Aligned 3D scans after using scan matching. The accumulated odometry error can be drastically reduced, but a small residual error remains (see offset).(c) The result of the EKF SLAM. The map built has been superimposed on a building plan for visual comparison. Notice that the offset is no longer present. Image courtesy of J. Weingarten [331].

5.8.5 Visual SLAM with a single camera

The term Visual SLAM (V-SLAM) was coined in 2003 by Davison [110, 112], who presented the first real-time EKF SLAM system with a single hand-held camera. No odometry, range finder, or GPS was used but just a single perspective camera. V-SLAM can be seen as a multiview Structure-from-Motion (SfM) (which we introduced in section 4.2.6). Indeed, both attempt to recover simultaneously both the camera motion and the structure (feature positions) of the environment from a single camera by tracking interest points in

the images. The main difference between V-SLAM and SfM is in that V-SLAM takes into account the feature uncertainty using a probabilistic framework. Another difference is that V-SLAM needs to process the images chronologically while SfM works also for unordered datasets. The original V-SLAM implementation by Davison used an extended Kalman filter. As we mentioned at the end of section 5.8.4, V-SLAM is also called bearing-only SLAM, to emphasize the fact that it uses only angle observations. This, again, is in contrast to laser-based or ultrasound-based SLAM, which need instead both angle and range information. Because of this, bearing-only SLAM is more challenging than range-bearing SLAM. In laser-based SLAM, the position of the features in the robot frame can be estimated from a single robot position. In V-SLAM, conversely, we need to move the camera to recover the position of the features, as we know from structure-from-motion.

The first problem in monocular V-SLAM is the estimation of the position of the features at the time the system starts. Using a rangefinder, this is obviously not a problem, but for V-SLAM this is not possible. To overcome this problem, in his original implementation Davison used a planar pattern of known geometry where the relative position of at least four boundary corners is known. From four corners of known position, the 6DoF camera pose with respect to these points can be determined uniquely.[32] As long as the camera is moved in front of the pattern, the camera pose can be estimated from single images. As the camera starts moving away from the pattern, new features must be triangulated and added to the map. At this point, the EKF V-SLAM process starts.

The implementation of the EKF V-SLAM is again the vanilla EKF applied to the SLAM problem. To implement it, we need to know the motion and measurement update functions.

As we did for the standard EKF SLAM, the state vector y contains both the camera pose and the feature position but this time also the camera velocity. Also, observe that Davison chose to parametrize the camera orientation with *quaternions* in order to avoid singularities, and thus the camera orientation is represented with four variables. The dimension of the state vector in the EKF V-SLAM is therefore $13 + 3n$; in fact, we need three parameters for the position r, four for the orientation-quaternion q, three for the translational velocity v, another three for the angular velocity ω, and $3n$ for the feature positions m_i. Also observe that in his implementation, the observed features are not lines but image points and therefore the feature position is represented by three cartesian coordinates. The vector state at time t can therefore be written as

$$y_t = [x_t, m_0, m_1, \ldots, m_{n-1}]^T, \tag{5.111}$$

where

32. The problem of determining the camera position and orientation from a set of 2D-3D point correspondences is known as *camera pose estimation* [29].

$$x_t = [r_t, q_t, v_t, \omega_t]^T . \tag{5.112}$$

Prediction step. Notice that in V-SLAM we do not use odometry to predict the next camera position. To overcome this problem, Davison proposed the use of a constant velocity model. This means that between consecutive frames the velocity is assumed to be constant, and therefore the position of the camera at time t is computed by integrating the motion starting at time *t-1*, assuming that the initial velocity is the one estimated at time *t-1*. By keeping this in mind, we can actually write the motion prediction function f as:

$$\hat{x}_t = f(x_{t-1}, u_t) = \begin{bmatrix} r + (v + V)\Delta t \\ q \times q((\omega + \Omega)\Delta t) \\ v + V \\ \omega + \Omega \end{bmatrix}, \tag{5.113}$$

where the unknown intentions (in terms of velocity and acceleration) of the carrier of the camera are taken into account in the constant velocity model by V and Ω, which are computed as:

$$V = a\Delta t \text{ and } \Omega = \alpha\Delta t , \tag{5.114}$$

where a and α are the unknown translational and angular accelerations that are modeled as zero mean Gaussian distributions. The prediction update equation of the EKF can therefore be written as

$$\begin{bmatrix} \hat{x}_t \\ \hat{\alpha}_t^0 \\ \hat{r}_t^0 \\ \cdots \\ \hat{\alpha}_t^{n-1} \\ \hat{r}_t^{n-1} \end{bmatrix} = \begin{bmatrix} f(x_{t-1}, u_t) \\ \alpha_{t-1}^0 \\ r_{t-1}^0 \\ \cdots \\ \alpha_{t-1}^{n-1} \\ r_{t-1}^{n-1} \end{bmatrix}, \tag{5.115}$$

where $(\hat{\alpha}_t^i, \hat{r}_t^i)$ and $(\alpha_{t-1}^i, r_{t-1}^i)$ denote the positions of the i-th feature at times t and *t-1* respectively.

Measurement update. In the measurement update, the camera pose is corrected based on the reobservation of features. In addition, new features are initialized and added to the map.

Figure 5.45 (Left) Feature image patches. (Right) Search regions predicted from the previous frame using a constant velocity motion model. Image courtesy of Andrew Davison [110].

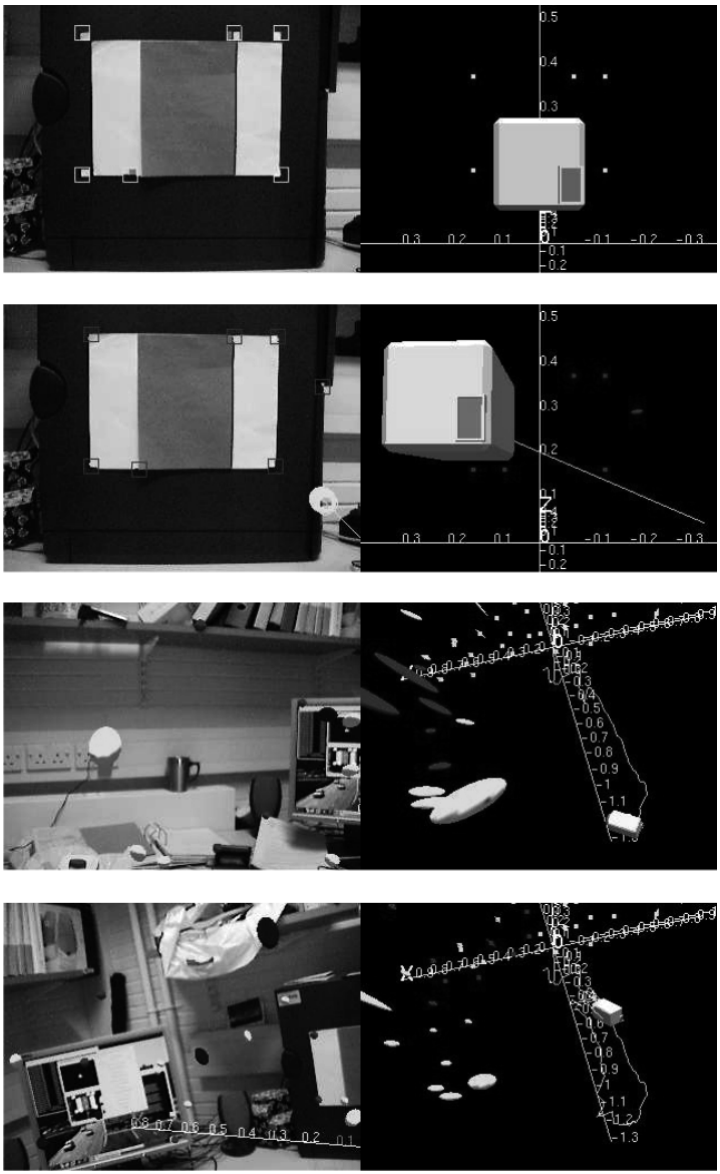
In V-SLAM, the features are interest points (figure 5.45) extracted using one of the interest point detectors described in section 4.5. Therefore, the features are expressed in image pixel coordinates.

In this phase, we also have to define the measurement function h. This function is used to compute the predicted observations, that is, to predict where the features are going to appear after the motion update. To determine h, we need to take into account the transformation from the world coordinate frame to the local camera frame and, in addition, the perspective transformation from the camera frame onto the image plane (figure 4.32, equation [4.44]). Therefore, function h is given exactly by equation (4.44). Finally, after computing the uncertainty of each predicted observations (which is drawn as an ellipse in figure 5.45), we can update the state vector and its covariance using the standard EKF measurement update equations. The main steps of Davison's V-SLAM are illustrated in figure 5.46.

5.8.6 Discussion on EKF SLAM

As we mentioned earlier, EKF SLAM is nothing but the application of the vanilla extended Kalman filter with a joint state composed of the robot pose and the feature locations. At every iteration both the state and the joint covariance matrix are updated, which means that the computation grows quadratically with the number of features. In order to overcome these limitations, efficient real-time implementations of EKF SLAM have been proposed in the last years, which can cope with thousand of features. The main idea is to decompose the map into smaller submaps for which covariances are updated separately.

Another problem of EKF SLAM is in the linearization made by the extended Kalman filter, which is reflected by the use of the Jacobians in the motion and measurement updates. Unfortunately, both the motion and the measurement model are typically nonlin-



(a)

(b)

(c)

(d)

Figure 5.46 (a) The camera starts moving with six known features on a pattern. (b) Nearby unknown features are initialized and added to the map. (c) As the camera moves, the uncertainty of the estimated features in the map increases. (d) As the camera revisits some of the features seen at the beginning, its uncertainty shrinks. Image courtesy of Andrew Davison [110,112].

ear, therefore their linearization can sometimes lead to inconsistency or divergence of the solution.

Another issue in EKF SLAM is its sensitiveness to incorrect data associations of the features, which happens when the robot incorrectly matches feature m_i with features m_j. This problem becomes even more important at the loop closure, that is, when the robot returns to reobserve features after a long traverse. Incorrect data association can occur frequently with 2D laser rangefinders due to the difficulty of identifying distinctive features in their point clouds. This task is, however, facilitated with cameras thanks to the huge availability of feature detectors (section 4.5). Some of these detectors, like SIFT (section 4.5.5.1), have recently demonstrated very successful results in loop closure detection over very long traverses (1000 km) [109] by employing the "bag of features" approach (see section 4.6 on location recognition).

We have also seen that the correlations between features is of significant importance in SLAM. The more observations are made, the more the correlations between features grow, the better the solution of the SLAM. Eliminating or ignoring the correlations between features (like it was done at the beginning of the research in EKF SLAM) is exactly contrary to the nature of the SLAM problem. As the robot moves and observes some features, these become more and more correlated. In the limit, they become fully correlated, that is, given the exact position of any feature, the location of any other feature can be determined with absolute precision.[33]

Regarding the convergence of the map, as the robot moves making observations, the determinant of the map covariance matrix and of all covariance submatrices converges monotonically toward zero. This means that the error in the relative position between the features decreases to the point where the map is known with absolute certainty or, alternatively, it reaches a lower bound that depends on the error introduced when the first observation was taken.

5.8.7 Graph-based SLAM

Graph-based SLAM was introduced for the first time in [200], which influenced many other implementations. Most of the graph-based SLAM techniques attempt to solve the full SLAM problem, but several approaches can also be found in the literature to solve the online SLAM problem.

Graph-based SLAM is born from the intuition that the SLAM problem can be interpreted as a sparse graph of nodes and constraints between nodes. The nodes of the graph are the robot locations $x_0, x_1, \ldots, x_T$ and the n features in the map $m_0, m_1, \ldots, m_{n-1}$. The constraints are the relative position between consecutive robot poses x_{t-1}, x_t (given by the

33. Note, this is only possible in principle. In real scenarios, there is always some uncertainty left (e.g., measurement uncertainty).

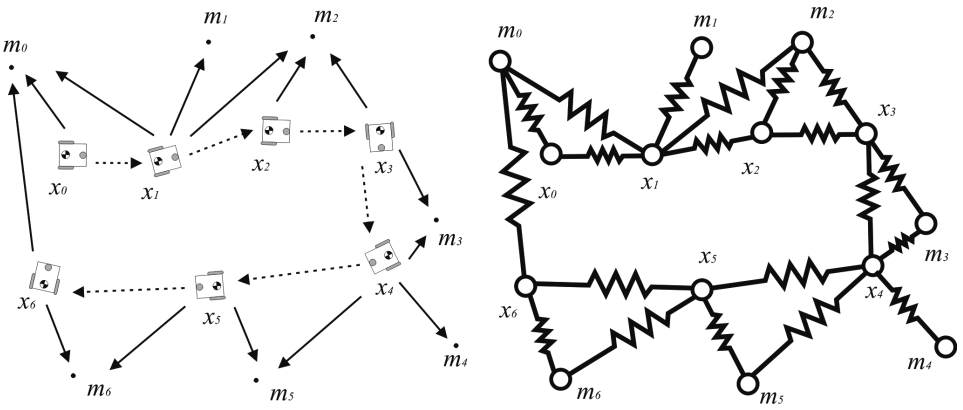
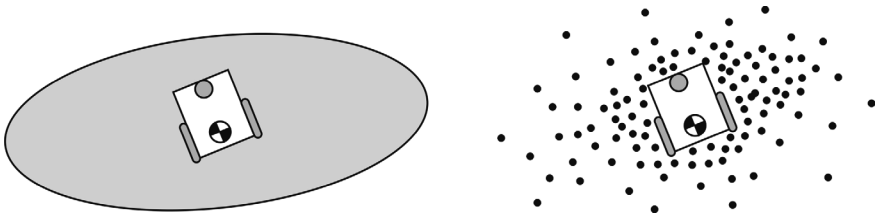


Figure 5.47 Evocative illustration of the graph construction. Constraints between nodes in the graph are represented as "soft" constraints (like springs). The solution of the SLAM problem can then be computed as the configuration with minimal energy.

odometry input u_t) and the relative position between the robot locations and the features observed from those locations.

The key property to remember about graph-based SLAM is that the constraints are not to be thought as rigid constraints but as soft constraints (figure 5.47). It is by relaxing these constraints that we can compute the solution to the full SLAM problem, that is, the best estimate of the robot path and the environment map. In other words, graph-based SLAM represents robot locations and features as the nodes of an elastic net. The SLAM solution can then be found by computing the state of minimal energy of this net [139]. Common optimization techniques to find the solution are based on gradient descent, and similar ones. A very efficient minimization procedure, along with open source code, was proposed in [141].

There is a significant advantage of graph-based SLAM techniques over EKF SLAM. As we have seen, in EKF SLAM the amount of computation and memory requirement to update and store the covariance matrix grows quadratically in the number of features. Conversely, in graph-based SLAM the update time of the graph is constant and the required memory is linear in the number of features. However, the final graph optimization can become computationally costly if the robot path is long. Nevertheless, graph-based SLAM algorithms have shown impressing and very successful results with even hundred million features [79, 116, 117, 173, 315]. However, these algorithms attempt to optimize over the entire robot path and were therefore implemented to work offline. Some of the online implementations used submap approaches.



(a) *(b)*

Figure 5.48 The standard EKF SLAM represent the probability distribution of the robot location is a parametric form, which is a two-dimensional Gaussian (a). Conversely, particle filter SLAM represent this the probability distribution as a set of particles drawn randomly from the parametric distribution (b). For the specific example of a Gaussian distribution, (b) the density of particles is higher toward the center of the Gaussian and decreases with the distance.

5.8.8 Particle filter SLAM

This particular solution to the SLAM problem is based on the randomized sampling of the belief distribution that we introduced already on page 315. The term *particle* filter is born from the fact that it represents the robot belief distribution not in a parametric form (like a Gaussian) but rather as a set of samples (i.e., *particles*) drawn randomly from this distribution. This concept is pictorially illustrated in figure 5.48. The power of this representation is in its ability to model any sort of distribution (e.g., non-Gaussian) and also nonlinear transformations.

Particle filters find their origin in Monte Carlo methods [215], but a step that makes them practically applicable to the SLAM problem is based on the work of Rao and Blackwell [75, 263], from which these filters inherited the name of *Rao-Blackwellized* particle filters. Finally, the Rao-Blackwellized particle filter was applied for the first time to the SLAM problem by Murphy and Russel [239] and found a very efficient implementation in the work of Montemerlo et al. [231], who also coined the name of FastSLAM.

Now we will give a general overview of the particle filter SLAM. For a detailed explanation of the solution to this problem, we refer the reader to the original paper on FastSLAM [231].

At every time step, the particle filter maintains always the same number K of particles (e.g., $K = 1000$). Each particle contains an estimate of the robot path $X_t^{[k]}$ and estimates of the position of each feature in the map, which are represented as two-dimensional Gaussians with mean values $\mu_{t,i}^{[k]}$ and covariance matrices $\Sigma_{t,i}^{[k]}$. Therefore, a particle is characterized by

$$X_t^{[k]};\ (\mu_{t,0}^{[k]}, \Sigma_{t,0}^{[k]});\ (\mu_{t,1}^{[k]}, \Sigma_{t,1}^{[k]});\ \ldots;\ (\mu_{t,1n-1}^{[k]}, \Sigma_{t,n-1}^{[k]}) \tag{5.116}$$

where k denotes the index of the particle and n the number of features in the map. Note that in particle filter SLAM, the mean and covariance of each feature are updated using distinct Kalman filters, one for each feature in the map.

When the robot moves, the motion model specified by the odometry reading u_t is applied to each particle $x_{t-1}^{[k]}$ to generate the new location $x_t^{[k]}$.

When the robot makes an observation z_t, we compute for each particle the so-called *importance factor* $w_t^{[k]}$, which is determined as the probability of observing z_t given the particle $x_t^{[k]}$ and all previous observations $z_{0 \to t-1}$, that is,

$$w_t^{[k]} = p(z_t | x_t^{[k]}, z_{0 \to t-1}) . \tag{5.117}$$

Notice that computing the importance factor for each particle is like sampling the probability distribution $p(z_t | x_t, z_{0 \to t-1})$.

The final step in particle filter SLAM is called *resampling*. This step replaces the current set of particles with another set according to the importance factor determined above. Finally, the mean and covariance of each feature are updated according to the standard EKF update rule.

Although this description of the algorithm may appear rather complex, the FastSLAM algorithm can be readily implemented and is one of the easiest-to-implement SLAM algorithms. Furthermore, FastSLAM has the big advantage over EKF SLAM that its complexity grows logarithmically in the number of features (and thus not quadratically as in EKF SLAM). This is mainly because instead of using a single covariance matrix for both the robot pose and the map (as in EKF SLAM), it uses separate Kalman filters, one for each feature. A very efficient implementation of FastSLAM is FastSLAM 2.0, which was proposed by the same authors in [232].

Finally, another great advantage over EKF SLAM is that due to the use of randomized sampling it does not require the linearization of the motion model and can also represent non-Gaussian distributions.

5.8.9 Open challenges in SLAM

One of the first assumptions we made is that the map is time-invariant, that is, static. However, real-world environments present moving objects such as vehicles, animals, and humans. A good SLAM algorithm should then be robust against dynamic objects. One way to tackle this problem is by treating them as outliers. However, the ability to recognize these objects, or even to predict where they are moving to, would improve the efficiency and the quality of the final map.

Another current topic of research is multiple robot mapping [312], that is, how to combine the individual readings of a team of multiple robots exploring the environment.

Another issue in SLAM is its sensitiveness to incorrect data associations. This problem is particularly important at the loop closure, that is, when the robot returns to previously visited locations after a long traverse. 2D laser rangefinders are more prone than cameras to incorrect data associations due to the difficulty of identifying unique and distinctive features in laser point clouds. The task of recognizing loop closures is, however, facilitated with cameras. Cameras provide much richer information than lasers. Furthermore, the development of distinctive feature detectors such as SIFT and SURF (section 4.5.5), which are also robust under large changes in the camera viewpoint and scale, has allowed researchers to cope with very challenging and large-scale environments (even 1000 km [109] without GPS). This is made possible by the use of the "bag of features" approach, which we described in section 4.6.

As we have seen in section 5.8.5, visual SLAM is a very recent active field of research that is fascinating more and more researchers around the world. Although laser scanners are still the most used sensors for SLAM, cameras are more appealing because they are cheaper and provide much richer information. Furthermore, they are lighter than laser, which enables the use on-board micro lightweight helicopters [76]. However, monocular cameras have the disadvantage that they provide only bearing information rather than depth; therefore, the solution to the SLAM will always be up to a scale. The absolute scale can, however, be recovered using some prior information such as the knowledge of the size of one element in the scene (a window, a table, etc.) or using other sensors such as GPS, odometry, or IMU. A recent solution even demonstrated the ability to recover the absolute scale by exploiting nonholonomic constraints of wheeled vehicles [277]. Stereo cameras conversely provide measurements directly in the absolute scale, but their resolution degrades with the measured distance.

5.8.10 Open source SLAM software and other resources

Here is a list of some of open source SLAM software and datasets available online.

1. http://www.openslam.org contains one of the most comprehensive lists of SLAM software currently available. There you can find up-to-date resources for both C/C++ and Matlab. Furthermore, here you also can to upload your own SLAM algorithm.

2. http://www.doc.ic.ac.uk/~ajd/software.html contains both C/C++ and Matlab implementations of Davison's real-time monocular visual SLAM.

3. http://www.robots.ox.ac.uk/~gk/PTAM/ is an alternative real-time monocular visual SLAM algorithm known as PTAM and implemented by Klein and Murray [167].

4. http://webdiis.unizar.es/~neira/software/slam/slamsim.htm is a Matlab EKF SLAM simulator

5. http://www.rawseeds.org/home provides a large collection of benchmarked datasets for SLAM. Several sensors were used to acquire the data, among them laser rangefinder, multiple cameras, IMUs, and GPS. Furthermore, these datasets come with a ground truth that can be used to evaluate the performance of your SLAM algorithm.

6. Additional software, datasets, and lectures about SLAM can be found on the websites of the past SLAM summer schools: google SLAM summer school.

5.9 Problems

1. Consider a differential-drive robot that uses wheel encoders only. The wheels are a distance d apart, and each wheel has radius r. Suppose this robot uses only its encoders to attempt to describe a square, with sides of length $1000r$, returning to the origin. For each of range error, turn error, and drift error, supposing an error rate of 10%, compute the worst-case effect of each type of error on the different between the final actual robot position and the original position in both position and orientation.

2. Consider the environment of figure 5.6. Your robot begins in the top left room and has the goal of stopping in the large room at position B. Design a sequence for a behavior-based robot to navigate successfully to B. Behaviors available are:

LWF: left wall follow
RWF: right wall follow
HF: go down centerline of a hallway
Turn X: turn X degrees left/right
Move X: move X centimeters forward/backward
EnterD: center and enter through a doorway

Termination conditions available are:

DoorL: doorway on left
DoorR: doorway on right
HallwayI: hallway intersection

3. Consider exact cell decomposition. What is the worst-case and best-case number of nodes that may be created when using this method as a function of the number of convex polygons and the number of sides of each polygon?

4. Consider the case of figure 5.27 and the method of 5.6.7.5. Suppose an initial belief state: {1,1–2,2–3}, with the robot facing east with certainty and with uncertainty {0.4, 0.4, 0.2} respectively. Two perceptual events occur. First: {door on left; door on right}. Second: {nothing on left; hall on right}. Complete the resulting belief update and describe the belief state. There is no need to normalize the results.

5. Challenge Question.

Implement a simple EKF Visual SLAM for the case of a omnidirectional-drive robot moving in a two-dimensional environment. Assume a constant velocity model. For simplicity, you may also assume that the robot is constrained to move along a line, which means Visual SLAM in a one-dimensional environment.

6 Planning and Navigation

6.1 Introduction

This book has focused on the elements of a mobile robot that are critical to robust mobility: the kinematics of locomotion, sensors for determining the robot's environmental context, and techniques for localizing with respect to its map. We now turn our attention to the robot's cognitive level. Cognition generally represents the purposeful decision making and execution that a system utilizes to achieve its highest-order goals.

In the case of a mobile robot, the specific aspect of cognition directly linked to robust mobility is *navigation competence*. Given partial knowledge about its environment and a goal position or series of positions, navigation encompasses the ability of the robot to act based on its knowledge and sensor values so as to reach its goal positions as efficiently and as reliably as possible. The focus of this chapter is how the tools of the previous chapters can be combined to solve this navigation problem.

Within the mobile robotics research community, a great many approaches have been proposed for solving the navigation problem. As we sample from this research background, it will become clear that in fact there are strong similarities between all of these approaches, even though they appear, on the surface, quite disparate. The key difference between various navigation architectures is the manner in which they decompose the problem into smaller subunits. In sections 6.3, 6.4, and 6.5, we describe the most popular of these architectures, contrasting their relative strengths and weaknesses.

First, however, in section 6.2 we discuss two additional key competences required for mobile robot navigation. Given a map and a goal location, *path planning* involves identifying a trajectory that will cause the robot to reach the goal location when executed. Path planning is a strategic problem-solving competence, since the robot must decide what to do over the long term to achieve its goals.

The second competence is equally important but occupies the opposite, tactical extreme. Given real-time sensor readings, *obstacle avoidance* means modulating the trajectory of the robot in order to avoid collisions. A great variety of approaches have demonstrated competent obstacle avoidance, and we survey a number of these approaches as well.

6.2 Competences for Navigation: Planning and Reacting

In the artificial intelligence community, planning and reacting are often viewed as contrary approaches or even opposites. When applied to physical systems such as mobile robots, however, planning and reacting have a strong complementarity, each being critical to the other's success. The navigation challenge for a robot involves executing a course of action (or plan) to reach its goal position. During execution, the robot must react to unforeseen events (e.g., obstacles) in such a way as to still reach the goal. Without reacting, the planning effort will not pay off because the robot will never physically reach its goal. Without planning, the reacting effort cannot guide the overall robot behavior to reach a distant goal—again, the robot will never reach its goal.

An information-theoretic formulation of the navigation problem will make this complementarity clear. Suppose that a robot R at time i has a map M_i and an initial belief state b_i. The robot's goal is to reach a position p while satisfying some temporal constraints: $loc_g(R) = p \; ; (g \leq n)$. Thus, the robot must be at location p at or before timestep n.

Although the goal of the robot is distinctly physical, the robot can only really sense its belief state, not its physical location, and therefore we map the goal of reaching location p to reaching a belief state b_g, corresponding to the belief that $loc_g(R) = p$. With this formulation, a plan q is nothing more than one or more trajectories from b_i to b_g. In other words, plan q will cause the robot's belief state to transition from b_i to b_g if the plan is executed from a world state consistent with both b_i and M_i.

Of course, the problem is that the latter condition may not be met. It is entirely possible that the robot's position is not quite consistent with b_i, and it is even likelier that M_i is either incomplete or incorrect. Furthermore, the real-world environment is dynamic. Even if M_i is correct as a single snapshot in time, the planner's model regarding how M changes over time is usually imperfect.

In order to reach its goal nonetheless, the robot must incorporate new information gained during plan execution. As time marches forward, the environment changes and the robot's sensors gather new information. This is precisely where reacting becomes relevant. In the best of cases, reacting will modulate robot behavior locally in order to correct the planned-upon trajectory so that the robot still reaches the goal. At times, unanticipated new information will require changes to the robot's strategic plans, and so ideally the planner also incorporates new information as that new information is received.

Taken to the limit, the planner would incorporate every new piece of information in real time, instantly producing a new plan that in fact reacts to the new information appropriately. This extreme, at which point the concept of planning and the concept of reacting merge, is called *integrated planning and execution* and is discussed in section 6.5.4.3.

Completeness. A useful concept throughout this discussion of robot architecture involves whether particular design decisions sacrifice the system's ability to achieve a desired goal whenever a solution exists. This concept is termed *completeness*. More formally, the robot system is *complete* if and only if, for all possible problems (i.e., initial belief states, maps, and goals), when there exists a trajectory to the goal belief state, the system will achieve the goal belief state (see [40] for further details). Thus when a system is incomplete, then there is at least one example problem for which, although there is a solution, the system fails to generate a solution. As you may expect, achieving completeness is an ambitious goal. Often, completeness is sacrificed for computational complexity at the level of representation or reasoning. Analytically, it is important to understand how completeness is compromised by each particular system.

In the following sections, we describe key aspects of planning and reacting as they apply to mobile robot path planning and obstacle avoidance and describe how representational decisions impact the potential completeness of the overall system. For greater detail, refer to [32, 44, chapter 25].

6.3 Path Planning

Even before the advent of affordable mobile robots, the field of path planning was heavily studied because of its applications in the area of industrial manipulator robotics. Interestingly, the path-planning problem for a manipulator with, for instance, six degrees of freedom is far more complex than that of a differential-drive robot operating in a flat environment. Therefore, although we can take inspiration from the techniques invented for manipulation, the path-planning algorithms used by mobile robots tend to be simpler approximations owing to the greatly reduced degrees of freedom. Furthermore, industrial robots often operate at the fastest possible speed because of the economic impact of high throughput on a factory line. So, the dynamics and not just the kinematics of their motions are significant, further complicating path planning and execution. In contrast, a number of mobile robots operate at such low speeds that dynamics are rarely considered during path planning, further simplifying the mobile robot instantiation of the problem.

Configuration space. Path planning for manipulator robots and, indeed, even for most mobile robots, is formally done in a representation called *configuration space*. Suppose that a robot arm (e.g., SCARA robot) has k degrees of freedom. Every state or configuration of the robot can be described with k real values: q_1, …, q_k. The k-values can be regarded as a point p in a k-dimensional space called the configuration space C of the robot. This description is convenient because it allows us to describe the complex 3D shape of the robot with a single k-dimensional point.

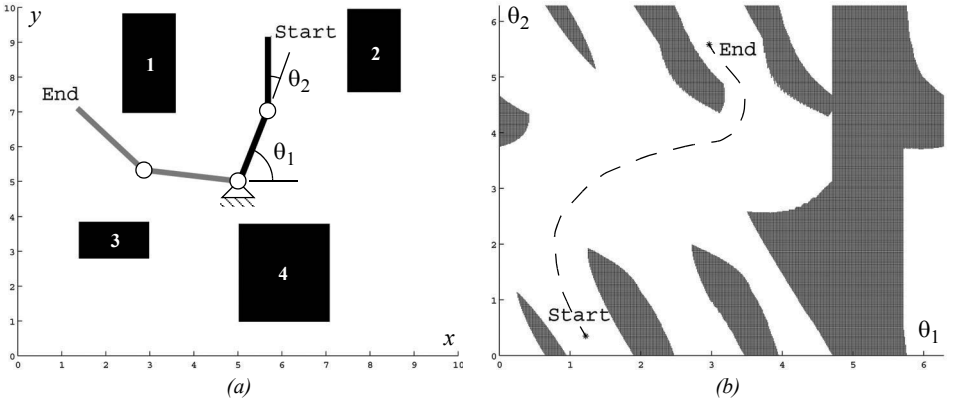


Figure 6.1
Physical space (a) and configuration space (b): (a) A two-link planar robot arm has to move from the configuration *start* to *end*. The motion is thereby constraint by the obstacles 1 to 4. (b) The corresponding configuration space shows the free space in joint coordinates (angle θ_1 and θ_2) and a path that achieves the goal.

Now consider the robot arm moving in an environment where the workspace (i.e., its physical space) contains known obstacles. The goal of path planning is to find a path in the physical space from the initial position of the arm to the goal position, avoiding all collisions with the obstacles. This is a difficult problem to visualize and solve in the physical space, particularly as k grows large. But in configuration space the problem is straightforward. If we define the *configuration space obstacle* O as the subspace of C where the robot arm bumps into something, we can compute the free space $F = C - O$ in which the robot can move safely.

Figure 6.1 shows a picture of the physical space and configuration space for a planar robot arm with two links. The robot's goal is to move its end effector from position *start* to *end*. The configuration space depicted is 2D because each of two joints can have any position from 0 to 2π. It is easy to see that the solution in C-space is a line from *start* to *end* that remains always within the free space of the robot arm.

For mobile robots operating on flat ground, we generally represent robot position with three variables (x, y, θ), as in chapter 3. But, as we have seen, most robots are nonholonomic, using differential-drive systems or Ackerman steered systems. For such robots, the nonholonomic constraints limit the robot's velocity $(\dot{x}, \dot{y}, \dot{\theta})$ in each configuration (x, y, θ). For details regarding the construction of the appropriate *free space* to solve such path-planning problems, see [32, p. 405].

In mobile robotics, the most common approach is to assume for path-planning purposes that the robot is in fact holonomic, simplifying the process tremendously. This is especially

common for differential-drive robots because they can rotate in place, and so a holonomic path can be easily mimicked if the rotational position of the robot is not critical.

Furthermore, mobile roboticists will often plan under the further assumption that the robot is simply a *point*. Thus we can further reduce the configuration space for mobile robot path planning to a 2D representation with just x- and y-axes. The result of all this simplification is that the configuration space looks essentially identical to a 2D (i.e., flat) version of the physical space, with one important difference. Because we have reduced the robot to a point, we must inflate each obstacle by the size of the robot's radius to compensate. With this new, simplified configuration space in mind, we can now introduce common techniques for mobile robot path planning.

Path-planning overview. The robot's environment representation can range from a continuous geometric description to a decomposition-based geometric map or even a topological map, as described in section 5.5. The first step of any path-planning system is thus to transform this possibly continuous environmental model into a discrete map suitable for the chosen path-planning algorithm. Path planners differ in how they use this discrete decomposition. In this book, we describe two general strategies:

1. Graph search: a connectivity graph in free space is first constructed and then searched. The graph construction process is often performed offline.
2. Potential field planning: a mathematical function is imposed directly on the free space. The gradient of this function can then be followed to the goal.

6.3.1 Graph search

Graph search techniques have traditionally been strongly rooted in the field of mathematics. Nonetheless, in recent years much of the innovation has been devised in the robotics community. This may be largely attributed to the need for real-time capable algorithms, which can accommodate evolving maps and thus changing graphs. For most of these methods we distinguish two main steps: graph construction, where nodes are placed and connected via edges, and graph search, where the computation of an (optimal) solution is performed.

6.3.1.1 Graph construction

Starting from a representation of free and occupied space, several methods are known to decompose this representation into a graph that can then be searched using any of the algorithms described in section 6.3.1.2 and 6.3.1.3. The challenge lies in constructing a set of nodes and edges that enable the robot to go anywhere in its free space while limiting the total size of the graph.

First, we describe two road map approaches that achieve this result with dramatically different types of roads. In the case of the *visibility graph*, roads come as close as possible

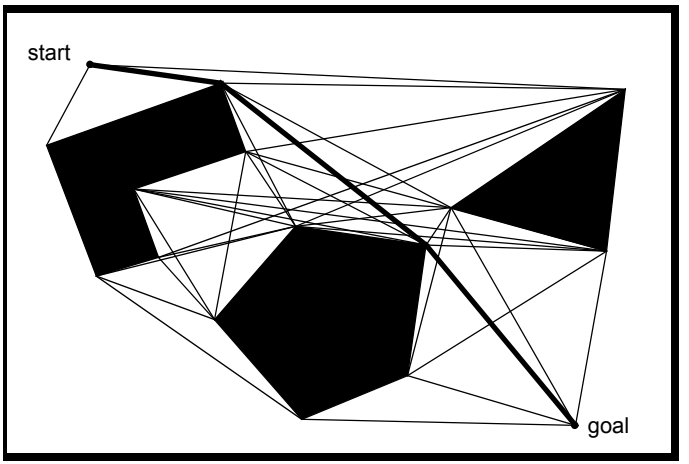


Figure 6.2
Visibility graph [32]. The nodes of the graph are the initial and goal points and the vertices of the configuration space obstacles (polygons). All nodes which are visible from each other are connected by straight-line segments, defining the road map. This means there are also edges along each polygon's sides.

to obstacles and resulting optimal paths are minimum-length solutions. In the case of the *Voronoi diagram*, roads stay as far away as possible from obstacles. We then detail cell decomposition methods where the idea is to discriminate between free and occupied geometric areas. Exact cell decomposition is a lossless decomposition, whereas approximate cell decomposition represents an approximation of the original map. A graph is then formed through a specified connectivity relation between cells. Finally, we describe the construction of lattice graphs, which are formed by shifting an underlying base set of edges over the free space. Lattice graphs are typically constructed by employing a mathematical model of the robot so that their edges become directly executable.

Visibility graph. The visibility graph for a polygonal configuration space C consists of edges joining all pairs of vertices that can see each other (including both the initial and goal positions as vertices as well). The unobstructed straight lines (roads) joining those vertices are obviously the shortest distances between them. The task of the path planner is then to find a (shortest) path from the initial position to the goal position along the roads defined by the visibility graph (figure 6.2).

Visibility graphs are moderately popular in mobile robotics, partly because their implementation is quite simple. Particularly when the environmental representation describes

objects in the environment as polygons in either continuous or discrete space, the visibility graph can employ the obstacle polygon descriptions readily.

There are, however, two important caveats when employing visibility graph search. First, the size of the representation and the number of edges and nodes increase with the number of obstacle polygons. Therefore, the method is extremely fast and efficient in sparse environments, but it can be slow and inefficient compared to other techniques when used in densely populated environments.

The second caveat is a much more serious potential flaw: solution paths found by graph search tend to take the robot as close as possible to obstacles on the way to the goal. More formally, we can prove that shortest solutions on the visibility graph are *optimal* in terms of path length. This powerful result also means that all sense of safety, with respect to staying a reasonable distance from obstacles, is sacrificed for this optimality. The common solution is to grow obstacles by significantly more than the robot's radius, or, alternatively, to modify the solution path after path planning to distance the path from obstacles where possible. Of course such actions sacrifice the optimal-length results of visibility graph path planning.

Voronoi diagram. Contrasting with the visibility graph approach, a Voronoi diagram is a complete road map method that tends to *maximize* the distance between the robot and obstacles in the map. For each point in free space, its distance to the nearest obstacle is computed. If you plot that distance as the height coming out of the page, it increases as you move away from an obstacle (see figure 6.3). At points that are equidistant from two or more obstacles, such a distance plot has sharp ridges. The Voronoi diagram consists of the edges formed by these sharp ridge points. When the configuration space obstacles are polygons, the Voronoi diagram consists of straight line and parabolic segments only. Algorithms that find paths on the Voronoi road map are complete, just as are visibility graph methods, because the existence of a path in the free space implies the existence of one on the Voronoi diagram as well (i.e., both methods guarantee completeness). However, the solution paths on the Voronoi diagram are usually far from optimal in the sense of total path length.

The Voronoi diagram has an important weakness in the case of limited range localization sensors. Since its edges maximize the distance to obstacles, any short-range sensor on the robot will be in danger of failing to sense its surroundings. If such short-range sensors are used for localization, then the chosen path will be quite poor from a localization point of view. On the other hand, the visibility graph method can be designed to keep the robot as close as desired to objects in the map.

There is, however, an important subtle advantage that the Voronoi diagram method has over most other graphs: *executability*. Given a particular planned path via Voronoi diagram planning, a robot with range sensors, such as a laser rangefinder or ultrasonics, can follow a Voronoi edge in the physical world using simple control rules that match those used to

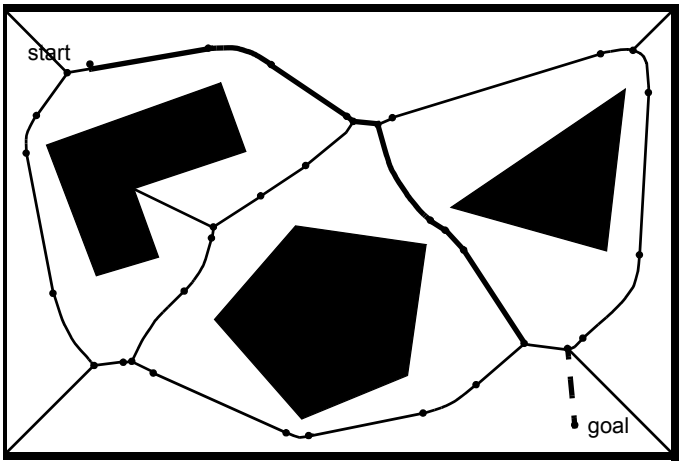


Figure 6.3
Voronoi diagram [32]. The Voronoi diagram consists of the lines constructed from all points that are equidistant from two or more obstacles. The initial q_{init} and goal q_{goal} configurations are mapped into the Voronoi diagram to q'_{init} and q'_{goal}, each by drawing the line along which its distance to the boundary of the obstacles increases the fastest. The points on the Voronoi diagram represent transitions from straight line segments (minimum distance between two lines) to parabolic segments (minimum distance between a line and a point).

create the Voronoi diagram: the robot maximizes the readings of local minima in its sensor values. This control system will naturally keep the robot on Voronoi edges, so that Voronoi motion can mitigate encoder inaccuracy. This interesting physical property of the Voronoi diagram has been used to conduct automatic mapping of an environment by finding and moving on unknown Voronoi edges, then constructing a consistent Voronoi map of the environment [103].

Exact cell decomposition. Figure 6.4 depicts exact cell decomposition, whereby the boundary of cells is based on geometric criticality. The resulting cells are each either completely free or completely occupied, and therefore path planning in the network is complete, like the road-map–based methods seen earlier. The basic abstraction behind such a decomposition is that the particular position of the robot within each cell of free space does not matter; what matters is rather the robot's ability to traverse from each free cell to adjacent free cells.

The key disadvantage of exact cell decomposition is that the number of cells and, therefore, the overall computational planning efficiency depends on the density and complexity of objects in the environment, just as with road-map–based systems. The key advantage is

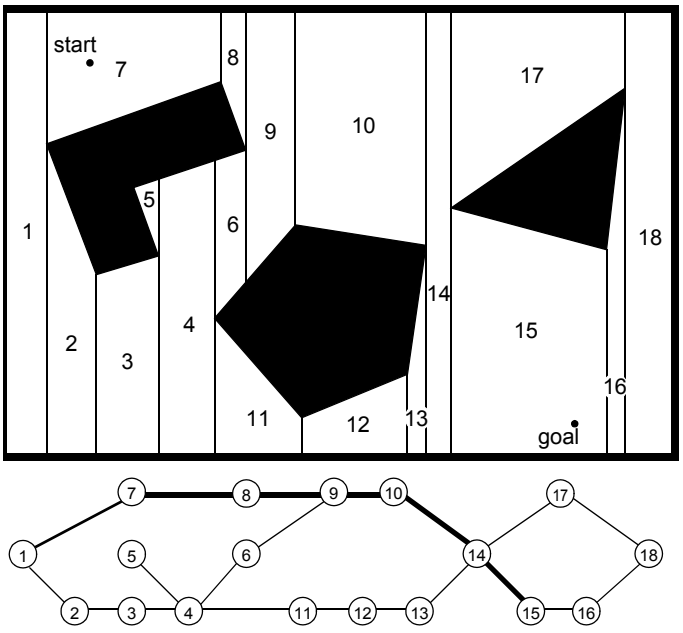


Figure 6.4
Example of exact cell decomposition. Cells are for example divided according to the horizontal coordinate of extremal obstacle points.

a result of this same correlation. In environments that are extremely sparse, the number of cells will be small, even if the geometric size of the environment is very large. Thus the representation will be efficient in the case of large, sparse environments. Practically speaking, due to complexities in implementation, the exact cell decomposition technique is used relatively rarely in mobile robot applications, although it remains a solid choice when a lossless representation is highly desirable—for instance, to preserve completeness fully.

Approximate cell decomposition. By contrast, approximate cell decomposition is one of the most popular graph construction techniques in mobile robotics. This is partly due to the popularity of grid environmental representations. These grid representations are themselves fixed grid-size decompositions and so they are identical to an approximate cell decomposition of the environment.

The most popular form of this, shown in figure 5.15, is the fixed-size cell decomposition. The cell size is not dependent on the particular objects in an environment, and so narrow passage ways can be lost due to the inexact nature of the tessellation. Practically

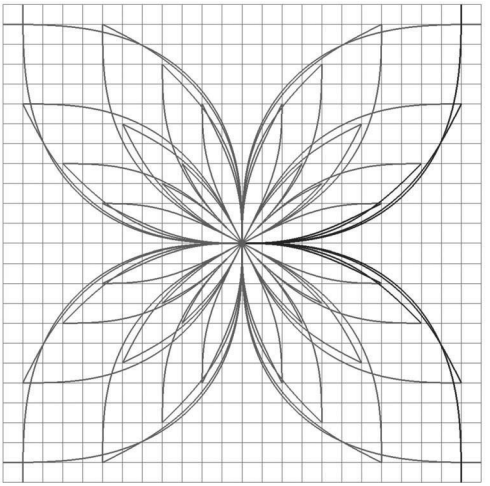


Figure 6.5 16-directional state lattice constructed for a planetary exploration rover. The state includes 2D position, heading, and curvature (x, y, θ, k). Note that straight segments are part of the lattice set but occluded by longer curved segments. The lattice is 2D-shift invariant and partially invariant to rotation. All successor edges of state (0, 0, 0, 0) are depicted in black. Image courtesy of M. Pivtoraiko [260].

speaking, this is rarely a problem owing to the very small cell size used (e.g., 5 cm on each side).

Figure 5.16 illustrates a variable-size approximate cell decomposition method. The free space is externally bounded by a rectangle and internally bounded by three polygons. The rectangle is recursively decomposed into smaller rectangles. Each decomposition generates four identical new rectangles. At each level of resolution only the cells whose interiors lie entirely in the free space are used to construct the connectivity graph. Path planning in such adaptive representations can proceed in a hierarchical fashion. Starting with a coarse resolution, the resolution is reduced until either the path planner identifies a solution or a limit resolution is attained (e.g, k • size of robot).

The great benefit of approximate cell decomposition is the low computational complexity induced to path planning.

Lattice graph. Lattice structures have only recently been adapted to graph search. They are formed by first constructing a base set of edges (such as the one depicted in figure 6.5) and then repeating it over the whole configuration space to form a graph. As such, the approximate cell decomposition technique could be interpreted as a simple lattice: the neighborhood structure of each cell forms a cross, which is then repeated by 2D shifts of

multiples of a single cell increment. The main benefit with respect to other graph construction methods lies in the design freedom in creating feasible edges, that is, edges that can be inherently executed by a robotic platform, however. To this end, Bicchi et al. [73] succeeded in applying an input discretization to a mathematical model of their robotic platform resulting in a configuration space lattice for certain simple kinematic vehicle models. Later, more broadly applicable methods have been devised in the configuration space directly: Pivtoraiko et al. [260] a priori fixed a problem specific configuration space discretization and dimensionality (e.g., in 2D position, orientation, curvature; figure 6.5). They then computed solutions to two point boundary problems between any two discrete states in the configuration space by also using a robot model. In a final step, the resulting large number of edges was pruned to a more manageable subset (the base lattice) by discarding edges which are similar to or can be decomposed into other edges already part of the subset.

Lattice graphs are typically precomputed for a given robotic platform and stored in memory. They thus belong to the class of approximate decomposition methods. Due to their inherent executability, edges along the solution path may be directly used as feed-forward commands to the controller.

Discussion. The fundamental cost of any fixed decomposition approach is memory. For a large environment, even when sparse, the grid must be represented in its entirety. Practically, because of the falling cost of RAM computer memory, this disadvantage has been mitigated in recent years.

In contrast to the exact decomposition methods, approximate approaches can sacrifice completeness but are mathematically less involved and thus easier to implement. In contrast to the fixed-size decompositions, variable-size decompositions will adapt to the complexity of the environment. Sparse environments will therefore contain appropriately fewer nodes and edges and consume dramatically less memory.

6.3.1.2 Deterministic graph search

Suppose now that our environment map has been converted into a connectivity graph using one of the graph generation methods presented earlier. Whatever map representation is chosen, the goal of path planning is to find the *best* path in the map's connectivity graph between the start and the goal, where *best* refers to the selected optimization criteria (e.g., the shortest path). In this section, we present several search algorithms that have become quite popular in mobile robotics. For an in-depth study on graph-search techniques we refer the reader to [44].

Discriminators. Due to the similarity between many graph search algorithms, we begin this section with an elaboration on their respective differences. To this end, it is beneficent to introduce the concepts of expected total cost $f(n)$, path cost $g(n)$, edge traversal cost $c(n, n')$, and heuristic cost $h(n)$, which are all functions of the node n (and an adjacent node

n'). In particular, we denote the accumulated cost from the start node to any given node n with $g(n)$. The cost from a node n to an adjacent node n' becomes $c(n, n')$, and the expected cost (heuristic cost) from a node n to the goal node is described with $h(n)$. The total expected cost from start to goal via state n can then be written as

$$f(n) = g(n) + \varepsilon \cdot h(n), \tag{6.1}$$

where ε is a parameter that assumes algorithm-dependent values.

In the special case that every individual edge in the graph assumes the same traversal cost (such as in an occupancy grid, introduced in section 5.5.2), optimal implementations may be developed in a simpler form and obtain faster execution speeds compared to the general instance. Examples of such algorithms include *depth-first* and *breadth-first* searches. On the other hand, Dijkstra's algorithm and variants allow for the computation of optimal paths in nonuniform cost maps as well. This comes at the cost of higher algorithmic complexity, however. In all of these implementations, $\varepsilon = 0$.

In the case of $\varepsilon \neq 0$, a heuristic function $h(n)$ is employed, which essentially incorporates additional information about the problem set and thus often allows for faster convergence of the search query. In this book, we restrict our attention to heuristics that are both consistent and underestimate the true cost. Most practical heuristics fulfill these requirements. For $\varepsilon = 1$, the optimal A* algorithm results, whereas for $\varepsilon > 1$ suboptimal or greedy A* variants are obtained.

Now that we have a general idea on the relation between some of the most popular graph search algorithms, we can proceed to introduce them in more detail.

Breadth-first search. This graph-search algorithm begins with the start node (denoted by A in figure 6.6) and explores all of its neighboring nodes. Then, for each of these nodes, it explores all their unexplored neighbors and so on. This process (that is, marking a node "active", exploring each of its neighbors and marking them "open", and finally marking the parent node "visited") is called node expansion. In breadth-first search, nodes are expanded in order of proximity to the start node with proximity defined as the shortest number of edge transitions. The algorithm proceeds until it reaches the goal node where it terminates. The computation of a solution is fast, since a reordering of nodes waiting for expansion is not necessary. They are already sorted in increasing order of proximity to the start node. Figure 6.6 illustrates the working principle of the breadth-first algorithm for a given graph.

It can be seen that the search always returns the path with the fewest number of edges between the start and goal node. If we assume that the cost of all individual edges in the graph is constant, then breadth-first search is also optimal in that it always returns the minimum-cost path. In this case, a node's reexpansion (as in the case of node G) can be easily circumvented by assigning a flag to a visited node. This addition does not affect solution

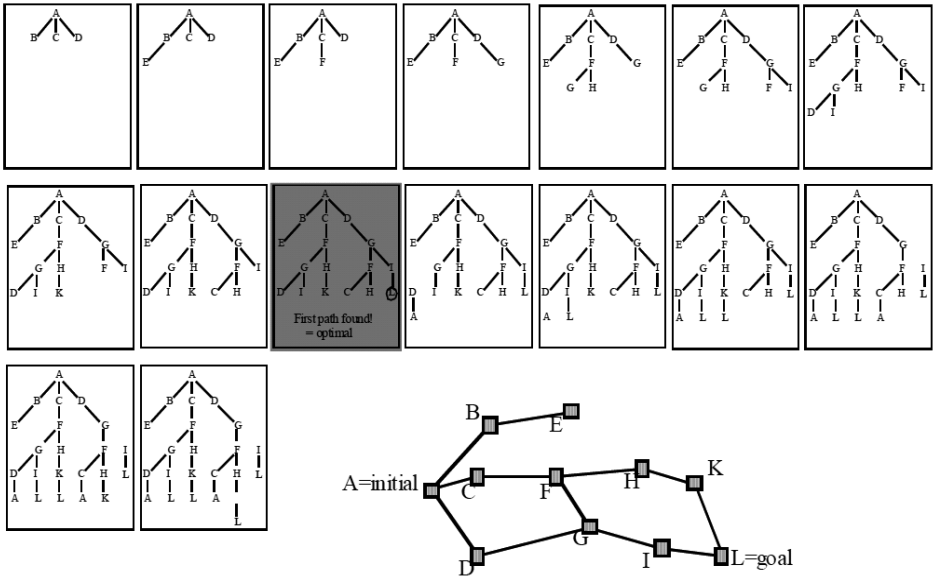


Figure 6.6 Working principle of breadth-first search.

optimality, since nodes are expanded in order of proximity to the start. However, if the graph has nonuniform costs associated with each edge, then breadth-first search is not guaranteed to be cost-optimal. Indeed, the path with the minimum number of edges does not necessarily coincide with the cheapest path, since there might be another path with more edges but lower total cost.

An example of breadth-first search algorithm in the context of robotics is the *wavefront expansion algorithm*, which is also known as *NF1* or *grassfire* [183]. This algorithm is an efficient and simple-to-implement technique for finding routes in fixed-size cell arrays. The algorithm employs wavefront expansion from the goal position outward, marking for each cell its L^1 (Manhattan) distance to the goal cell [154] (see figure 6.7). This process continues until the cell corresponding to the initial robot position is reached. At this point, the path planner can estimate the robot's distance to the goal position as well as recover a specific solution trajectory by simply linking together cells that are adjacent and always closer to the goal.

Given that the entire array can be in memory, each cell is only visited once when looking for the shortest discrete path from the initial position to the goal position. So, the search is linear in the number of cells only. Thus, complexity does not depend on the sparseness and

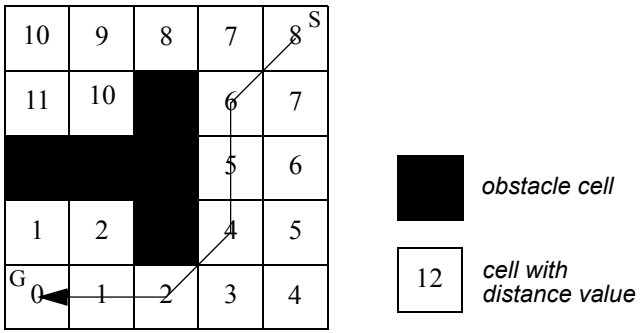


Figure 6.7
An example of the distance transform and the resulting path as it is generated by the NF1 function. S denotes the start, G the goal. The neighbors of each cell *i* are defined as the four adjacent cells that share an edge with *i* (4-neighborhood).

density of the environment, nor on the complexity of the objects' shapes in the environment.

Depth-first search. The working principle of depth-first search algorithm is shown in figure 6.8. In contrast to breadth-first search, depth-first search expands each node up to the deepest level of the graph (until the node has no more successors). As those nodes are expanded, their branch is removed from the graph and the search backtracks by expanding the next neighboring node of the start node until its deepest level and so on. An inconvenience of this algorithm is that it may revisit previously visited nodes or enter redundant paths. However, these situations may be easily avoided through an efficient implementation. A significant advantage of depth-first over breadth-first is space complexity. In fact, depth-first needs to store only a single path from the start node to the goal node along with all the remaining unexpanded neighboring nodes for each node on the path. Once each node has been expanded and all its children nodes have been explored, it can be removed from memory.

Dijkstra's algorithm. Named after its inventor, E.W. Dijkstra, this algorithm is similar to breadth-first search, except that edge costs may assume any positive value and the search still guarantees solution optimality [114]. This introduces some additional complexity into the algorithm for which we need to introduce the concept of the *heap*, a specialized tree-based data structure. Its elements (which are comprise to-be-expanded graph nodes) are ordered according to a key, which in our case amounts to the expected total path cost $f(n)$ at that given node n. Dijkstra's algorithm then expands nodes starting from the start similar

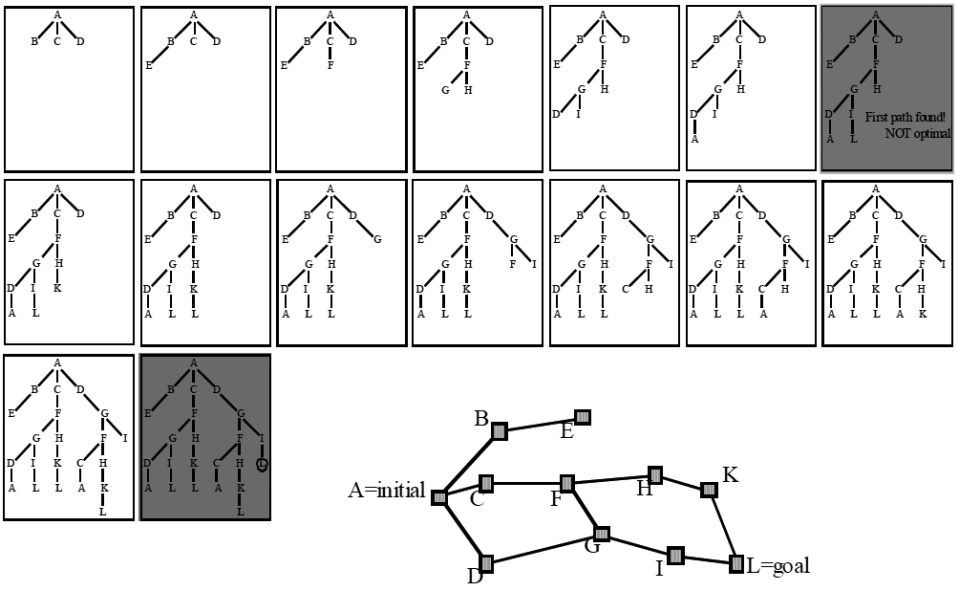


Figure 6.8 Working principle of depth-first search.

to breadth-first search, except that the neighbors of the expanded node are placed in the heap and reordered according to their $f(n)$ value, which corresponds to $g(n)$ since no heuristic is used. Subsequently, the cheapest state on the heap (the top element after reordering) is extracted and expanded. This process continues until the goal node is expanded, or no more nodes remain on the heap. A solution can then be backtracked from the goal to the start. Due to reorder operations on the heap, the time complexity rises from $O(n+m)$ in breadth-first search to $O(n\log(n)+m)$, with n the number of nodes, and m the number of edges.

In robotic applications, Dijkstra's search is typically computed from the robot's goal position. Consequently, not only the best path from the start node to the goal is computed, but also all lowest cost paths from any starting position in the graph to the goal node. The robot may thus localize and determine the best route toward the goal based on its current position. After moving some distance along this path, the process is repeated until the goal is reached, or the environment changes (which would require a recomputation of the solution). This technique allows the robot to reach the goal without replanning even in presence of localization and actuation noise.

A* algorithm. For consistent heuristics, the A* algorithm (pronounced "a star") [147] is similar to Dijkstra's algorithm. However, the inclusion of a heuristic function $h(n)$, which

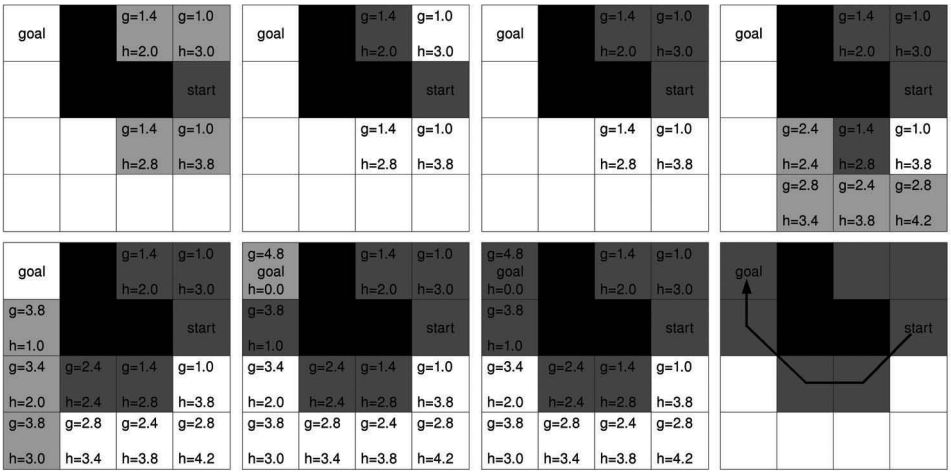


Figure 6.9 Working principle of the A* algorithm. Nodes are expanded in order of lowest $f(n) = g(n) + h(n)$ cost. $g(n)$ is indicated at the top left corner, $h(n)$ at the bottom right corner of each cell. The neighborhood of each cell is selected as the 8-neighborhood (all eight adjacent cells). Diagonal moves cost $\sqrt{2}$ times as much as horizontal and vertical moves. Obstacle cells are colored in black, expanded cells in dark gray, and cells put on the heap during this expansion step in light gray. Image courtesy of M. Rufli.

encodes additional knowledge about the graph, makes this algorithm especially efficient for single node to single node queries. In order to guarantee solution optimality, the heuristic is required to be an underestimating function of the cost to go. In robotics, A* is mainly employed on a grid, and the heuristic is then often chosen as the distance between any cell and the goal cell in absence of any obstacles. If such knowledge is available, it can be used to guide the search toward the goal node. Generally, this dramatically reduces the number of node expansions required to arrive at a solution compared to Dijkstra's algorithm.

A* search begins by expanding the start node and placing all of its neighbors on a heap. In contrast to Dijkstra's algorithm, the heap is ordered according to the smallest $f(n)$ value that includes the heuristic function $h(n)$. The lowest cost state is then extracted and expanded. This continues until the goal node is explored. The lowest cost solution can again be backtracked from the goal. For an example, see figure 6.9. The time complexity of A* largely depends on the chosen heuristic $h(n)$. On average, much better performance than with Dijkstra's algorithm can be expected, however.

Often it is not necessary to obtain an optimal solution, as long as there are guarantees on its suboptimality level. In such cases, a solution that costs at most ε times the (unknown) optimal solution may be obtained by setting $\varepsilon > 1$. The solution may then be improved as

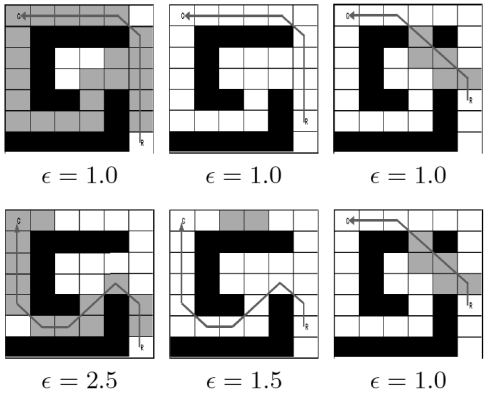


Figure 6.10 comparison of the number of expanded cells for D* (top) and Anytime D* (bottom, starting with a sub-optimality value of 2.5) in a planning and re-planning scenario. Note that an opening in the top wall is detected in the third frame, after the robot has moved upward twice. Obstacle cells are colored in black, cells expanded during a given time-step in gray. Image courtesy of M. Rufli.

search time allows, by reusing parts of the previous queries. This procedure results in the Anytime Replanning A* algorithm [191]. If the heuristic is accurate, far fewer states can be expected to be expanded than for optimal A*.

D* algorithm. The D* algorithm [304, 170] represents an incremental replanning version of A*, where the term incremental refers to the algorithm's reuse of previous search effort in subsequent search iterations. Let us illustrate this with an example (see figure 6.10): our robot is initially provided with a crude map of the environment (i.e., obtained from an aerial image). In this map, the navigation module plans an initial path by employing A*. After executing this path for a while, the robot observes some changes in the environment with its onboard sensors. Subsequent to updating the map, a new solution path needs to be computed. This is where D* comes into play. Instead of generating a new solution from scratch (as A* would do), only states affected by the added (or removed) obstacle cells are recomputed. Because changes to the map are most often observed locally (due to proprioceptive sensors), the planning problem is usually reversed; node expansion begins from the robot goal state. In this way, large parts of the previous solution remain valid for the new computation. Compared to A*, search time may decrease by a factor of one to two orders of magnitude. For more detail and a description on computing affected states, consult [170].

Analogous to A*, the D* algorithm has also been extended to an anytime version, called Anytime D* [192].

6.3.1.3 Randomized graph search

When encountering complex high-dimensional path planning problems (such as in manipulation tasks on robotic arms, or molecule folding and docking queries for drug placement, and so on) it becomes infeasible to solve them exhaustively within reasonable time limits. Reverting to heuristic search methods is often not possible due to the lack of an appropriate heuristic function and a reduction of the problem dimensionality frequently fails due to velocity and acceleration constraints imposed on the model, which should not be violated for security reasons. In such situations, randomized search becomes useful, since it forgoes solution optimality for faster solution computation.

Rapidly Exploring Random Trees (RRTs). RRTs typically grow a graph online during the search process and thus a priori only require an obstacle map but no graph decomposition. The algorithm begins with an initial tree (which might be empty) and then successively adds nodes, connected via edges, until a termination condition is triggered. Specifically, during each step a random configuration q_{rand} in the free space is selected. The tree node that is closest to q_{rand}, denoted as q_{near}, is then computed. Starting from q_{near}, an edge (with fixed length) is grown toward q_{rand} using an appropriate robot motion model. The configuration q_{new} at the end of this edge is then added to the tree, if the connecting edge is collision-free [185]

Typical extensions to the algorithm aim at speeding-up solution computation: bidirectional versions grow partial trees from both the start and goal configuration. Besides parallelization capability, faster convergence in nonconvex environments can be expected [186]. Another often employed modification biases the process of selecting a random free space configuration q_{rand}. The goal node is then selected instead of q_{rand} with a fixed nonzero probability, thus guiding tree growth toward the goal state. This process is especially efficient in sparse environments, but it may lead to a slowdown in presence of concave obstacles [33].

Even though the RRT algorithm and its extensions lack solution optimality guarantees and deterministic completeness, it can be proven that they are probabilistically complete. This signifies that if a solution exists, the algorithm will eventually find it as the number of nodes added to the tree grows toward infinity (see figure 6.11).

6.3.2 Potential field path planning

Potential field path planning creates a field, or gradient, across the robot's map that directs the robot to the goal position from multiple prior positions (see [32]). This approach was originally invented for robot manipulator path planning and is used often and under many variants in the mobile robotics community. The potential field method treats the robot as a point under the influence of an artificial potential field $U(q)$. The robot moves by follow-

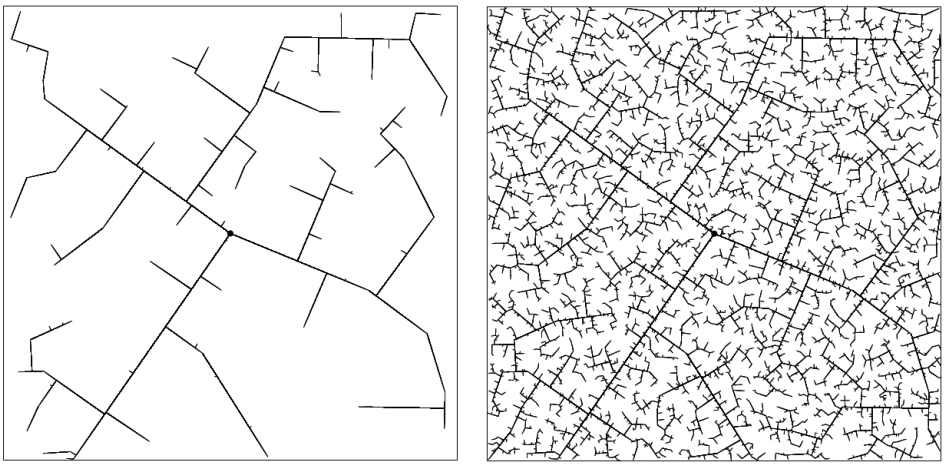


Figure 6.11 The evolution of a RRT. Image courtesy of S.M. LaValle [33].

ing the field, just as a ball would roll downhill. The goal (a minimum in this space) acts as an attractive force on the robot, and the obstacles act as peaks, or repulsive forces. The superposition of all forces is applied to the robot, which, in most cases, is assumed to be a point in the configuration space (see figure 6.12). Such an artificial potential field smoothly guides the robot toward the goal while simultaneously avoiding known obstacles.

It is important to note, though, that this is more than just path planning. The resulting field is also a control law for the robot. Assuming the robot can localize its position with respect to the map and the potential field, it can always determine its next required action based on the field.

The basic idea behind all potential field approaches is that the robot is attracted toward the goal, while being repulsed by the obstacles that are known in advance. If new obstacles appear during robot motion, one could update the potential field in order to integrate this new information. In the simplest case, we assume that the robot is a point; thus the robot's orientation θ is neglected, and the resulting potential field is only 2D (x, y). If we assume a differentiable potential field function $U(q)$, we can find the related artificial force $F(q)$ acting at the position $q = (x, y)$.

$$F(q) = -\nabla U(q), \tag{6.2}$$

where $\nabla U(q)$ denotes the gradient vector of U at position q.

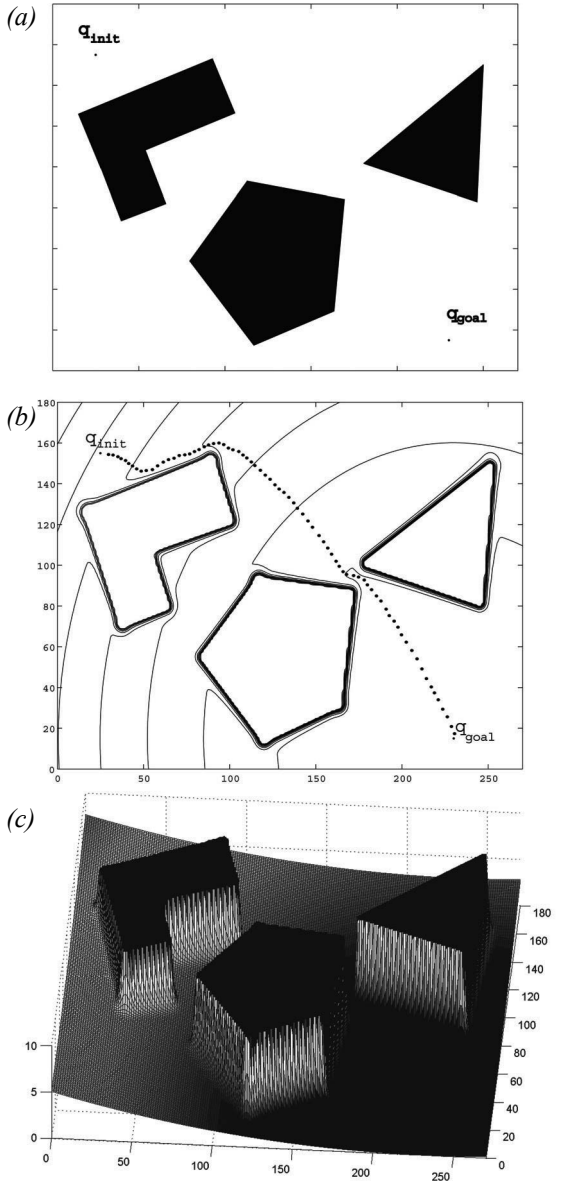


Figure 6.12
Typical potential field generated by the attracting goal and two obstacles (see [32]). (a) Configuration of the obstacles, start (top left) and goal (bottom right). (b) Equipotential plot and path generated by the field. (c) Resulting potential field generated by the goal attractor and obstacles.

$$\nabla U = \begin{bmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{bmatrix}. \tag{6.3}$$

The potential field acting on the robot is then computed as the sum of the attractive field of the goal and the repulsive fields of the obstacles:

$$U(q) = U_{att}(q) + U_{rep}(q). \tag{6.4}$$

Similarly, the forces can also be separated in a attracting and repulsing part:

$$\begin{aligned} F(q) &= F_{att}(q) - F_{rep}(q) \\ &= -\nabla U_{att}(q) - \nabla U_{rep}(q). \end{aligned} \tag{6.5}$$

Attractive potential. An attractive potential can, for example, be defined as a parabolic function.

$$U_{att}(q) = \frac{1}{2} k_{att} \cdot \rho_{goal}^2(q), \tag{6.6}$$

where k_{att} is a positive scaling factor and $\rho_{goal}(q)$ denotes the Euclidean distance $\|q - q_{goal}\|$. This attractive potential is differentiable, leading to the attractive force F_{att}

$$F_{att}(q) = -\nabla U_{att}(q) \tag{6.7}$$

$$= -k_{att} \cdot \rho_{goal}(q) \nabla \rho_{goal}(q) \tag{6.8}$$

$$= -k_{att} \cdot (q - q_{goal}) \tag{6.9}$$

that converges linearly toward 0 as the robot reaches the goal.

Repulsive potential. The idea behind the repulsive potential is to generate a force away from all known obstacles. This repulsive potential should be very strong when the robot is close to the object, but it should not influence its movement when the robot is far from the object. One example of such a repulsive field is

$$U_{rep}(q) = \begin{cases} \frac{1}{2}k_{rep}\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2 & \text{if } \rho(q) \le \rho_0 \\ 0 & \text{if } \rho(q) \ge \rho_0 \,, \end{cases} \tag{6.10}$$

where k_{rep} is again a scaling factor, $\rho(q)$ is the minimal distance from q to the object and ρ_0 the distance of influence of the object. The repulsive potential function U_{rep} is positive or zero and tends to infinity as q gets closer to the object.

If the object boundary is convex and piecewise differentiable, $\rho(q)$ is differentiable everywhere in the free configuration space. This leads to the repulsive force F_{rep}:

$$F_{rep}(q) = -\nabla U_{rep}(q) \tag{6.11}$$

$$= \begin{cases} k_{rep}\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\frac{1}{\rho^2(q)}\frac{q - q_{obstacle}}{\rho(q)} & \text{if } \rho(q) \le \rho_0 \\ 0 & \text{if } \rho(q) \ge \rho_0 \,. \end{cases}$$

The resulting force $F(q) = F_{att}(q) + F_{rep}(q)$ acting on a point robot exposed to the attractive and repulsive forces moves the robot away from the obstacles and toward the goal (see figure 6.12). Under ideal conditions, by setting the robot's velocity vector proportional to the field force vector, the robot can be smoothly guided toward the goal, similar to a ball rolling around obstacles and down a hill.

However, there are some limitations with this approach. One is local minima that appear dependent on the obstacle shape and size. Another problem might appear if the objects are concave. This might lead to a situation for which several minimal distances $\rho(q)$ exist, resulting in oscillation between the two closest points to the object, which could obviously sacrifice completeness. For more detailed analyses of potential field characteristics, refer to [32].

The extended potential field method. Khatib and Chatila proposed the extended potential field approach [164]. Like all potential field methods, this approach makes use of attractive and repulsive forces that originate from an artificial potential field. However, two additions to the basic potential field are made: the *rotation potential field* and the *task potential field*.

The rotation potential field assumes that the repulsive force is a function of the distance from the obstacle and the orientation of the robot relative to the obstacle. This is done using

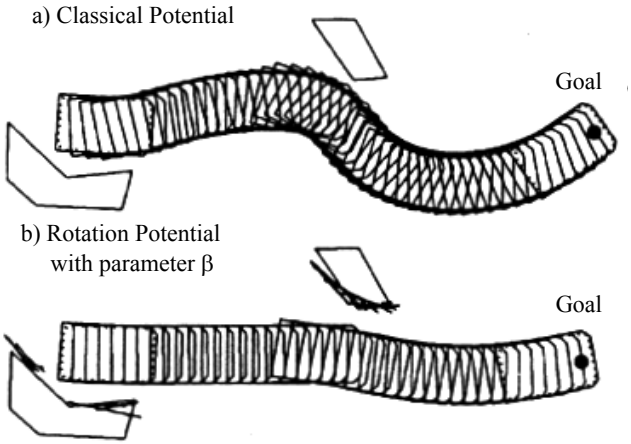


Figure 6.13
Comparison between a classical potential field and an extended potential field. Image courtesy of Raja Chatila [164].

a gain factor that reduces the repulsive force when an obstacle is parallel to the robot's direction of travel, since such an object does not pose an immediate threat to the robot's trajectory. The result is enhanced wall following, which was problematic for earlier implementations of potential fields methods.

The task potential field considers the present robot velocity, and from that it filters out those obstacles that should not affect the near-term potential based on robot velocity. Again a scaling is made, this time of all obstacle potentials when there are no obstacles in a sector named Z in front of the robot. The sector Z is defined as the space that the robot will sweep during its next movement. The result can be smoother trajectories through space. An example comparing a classical potential field and an extended potential field is depicted in figure 6.13.

Other extensions. A great variety of improvements to the artificial potential field method have been proposed and implemented since the development of Khatib's original approach in 1986 [163]. The most promising of these methods seems to be related to the harmonic potential field which is a solution to the Laplace equation [126, 230]

$$\nabla^2 U(q) \equiv 0\,,\ q \in \Omega\,, \tag{6.12}$$

where U again denotes the potential field as a function of the robot configuration q and Ω represents the workspace the robot operates in.

The main benefit of the harmonic potential field method with respect to earlier implementations is the complete absence of local minima inside the workspace. A unique solution may be generated through equation (6.12) and the specification of boundary conditions at start and goal locations and along the obstacle and workspace borders. In particular, the start location is pulled up to a high potential, whereas the goal position is pulled down to ground. For obstacle and workspace borders, we distinguish between two types of boundary conditions each resulting in a characteristic potential field. The Dirichlet condition requires that the potential is a known function along object boundaries (denoted with Γ)

$$U(q) = f(q), \; q \in \Gamma. \tag{6.13}$$

For $f(q) = const$, obstacle boundaries become equipotential lines. The robot then follows a path perpendicular to objects in their close vicinity. Excessively long but safe paths tend to emerge.

On the other hand, the von Neumann boundary condition requires

$$\frac{\partial U(q)}{\partial q} = g(q), \; q \in \Gamma. \tag{6.14}$$

where n is the normal vector to the obstacle boundary Γ. For $g(q) = 0$, robot motion parallel to object boundaries emerges. For all but the most elementary of obstacle geometries the Laplace equation needs to be solved numerically through a discretization of the workspace into cells. An iterative update rule (e.g the Gauss-Seidel method [155]) can then be applied until convergence. In several extensions, regional and directional constraints have been added to the harmonic potential field method to account for uneven terrain, nonholonomic and kinematic vehicle constraints [195], one-way roads [206], and external forces acting on the robot [207].

Potential fields are extremely easy to implement, much like the breadth-first search described on page 380. Thus, it has become a common tool in mobile robot applications in spite of its theoretical limitations.

This completes our brief summary of the path-planning techniques that are most popular in mobile robotics. Of course, as the complexity of a robot increases (e.g., large degree of freedom nonholonomics) and, particularly, as environment dynamics becomes more significant, then the path-planning techniques described earlier become inadequate for grappling with the full scope of the problem. However, for robots moving in largely flat terrain, the mobility decision-making techniques roboticists use often fall under one of the preceding categories.

But a path planner can take into consideration only the environmental obstacles that are known to the robot in *advance*. During path execution the robot's actual sensor values may disagree with expected values due to map inaccuracy or a dynamic environment. Therefore, it is critical that the robot modify its path in real time based on actual sensor values. This is the competence of *obstacle avoidance*, which we discuss next.

6.4 Obstacle avoidance

Local obstacle avoidance focuses on changing the robot's trajectory as informed by its sensors during robot motion. The resulting robot motion is both a function of the robot's current or recent sensor readings *and* its goal position and relative location to the goal position. The obstacle avoidance algorithms presented here depend to varying degrees on the existence of a global map and on the robot's precise knowledge of its location relative to the map. Despite their differences, all of the algorithms can be termed obstacle avoidance algorithms because the robot's local sensor readings play an important role in the robot's future trajectory. We first present the simplest obstacle avoidance systems that are used successfully in mobile robotics. The Bug algorithm represents such a technique in that only the most recent robot sensor values are used, and the robot needs, in addition to current sensor values, only approximate information regarding the direction of the goal. More sophisticated algorithms are presented afterward, taking into account recent sensor history, robot kinematics, and even dynamics.

6.4.1 Bug algorithm

The Bug algorithm [198, 199] is perhaps the simplest obstacle-avoidance algorithm one could imagine. The basic idea is to follow the contour of each obstacle in the robot's way and thus circumnavigate it.

With Bug1, the robot fully circles the object first, then departs from the point with the shortest distance toward the goal (figure 6.14). This approach is, of course, very inefficient but it guarantees that the robot will reach any reachable goal.

With Bug2 the robot begins to follow the object's contour, but departs immediately when it is able to move directly toward the goal. In general this improved Bug algorithm will have significantly shorter total robot travel, as shown in figure 6.15. However, one can still construct situations in which Bug2 is arbitrarily inefficient (i.e., nonoptimal).

A number of variations and extensions of the Bug algorithm exist. We mention one more, the Tangent Bug [161], which adds range sensing and a local environmental representation termed the local tangent graph (LTG). Not only can the robot move more efficiently toward the goal using the LTG, but it can also go along shortcuts when contouring obstacles and switch back to goal seeking earlier. In many simple environments, Tangent Bug approaches globally optimal paths.

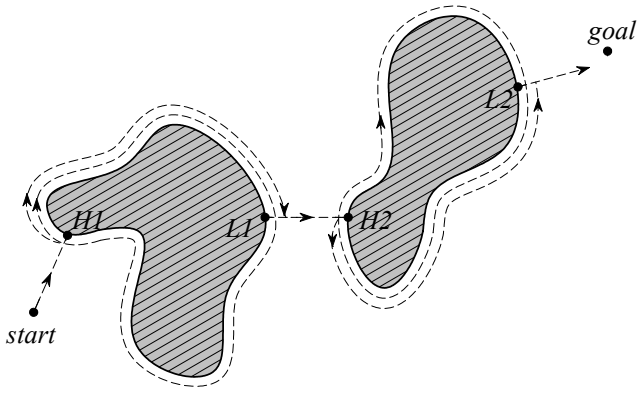


Figure 6.14
Bug1 algorithm with H1, H2, hit points, and L1, L2, leave points [199].

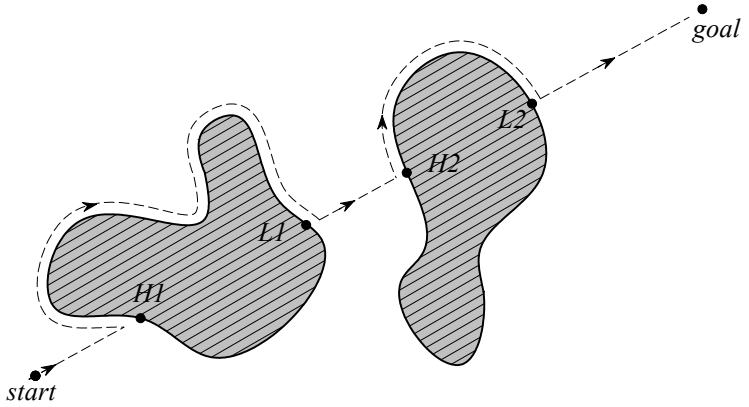


Figure 6.15
Bug2 algorithm with H1, H2, hit points, and L1, L2, leave points [199].

Practical application: example of Bug2. Because of the popularity and simplicity of Bug2, we present a specific example of obstacle avoidance using a variation of this technique. Consider the path taken by the robot in figure 6.15. One can characterize the robot's motion in terms of two states, one that involves moving toward the goal and a second that involves moving around the contour of an obstacle. We will call the former state `GOAL-`

SEEK and the latter WALLFOLLOW. If we can describe the motion of the robot as a function of its sensor values and the relative direction to the goal for each of these two states, and if we can describe when the robot should switch between them, then we will have a practical implementation of Bug2. The following pseudocode provides the highest-level control code for such a decomposition:

```
public void bug2(position goalPos){
 boolean atGoal = false;

 while( ! atGoal){
   position robotPos = robot.GetPos(&sonars);
   distance goalDist = getDistance(robotPos, goalPos);
  angle goalAngle = Math.atan2(goalPos, robotPos)-robot.GetAngle();
   velocity forwardVel, rotationVel;

   if(goalDist < atGoalThreshold){
     System.out.println("At Goal!");
     forwardVel = 0;
     rotationVel = 0;
     robot.SetState(DONE);
     atGoal = true;
   }
   else{
     forwardVel = ComputeTranslation(&sonars);
     if(robot.GetState() == GOALSEEK){
        rotationVel = ComputeGoalSeekRot(goalAngle);
        if(ObstaclesInWay(goalAngle, &sonars))
          robot.SetState(WALLFOLLOW);
     }
     if(robot.GetState() == WALLFOLLOW){
        rotationVel = ComputeRWFRot(&sonars);
        if( ! ObstaclesInWay(goalAngle, &sonars))
          robot.SetState(GOALSEEK);
     }
   }
   robot.SetVelocity(forwardVel, rotationVel);
 }
}
```

In the ideal case, when encountering an obstacle one would choose between left wall following and right wall following depending on which direction is more promising. In this simple example we have only right wall following, a simplification for didactic purposes that ought not find its way into a real mobile robot program.

Now we consider specifying each remaining function in detail. Consider for our purposes a robot with a ring of sonars placed radially around the robot. This imagined robot will be differential-drive, so that the sonar ring has a clear "front" (aligned with the forward direction of the robot). Furthermore, the robot accepts motion commands of the form shown above, with a rotational velocity parameter and a translational velocity parameter. Mapping these two parameters to individual wheel speeds for each of the two differential-drive chassis' drive wheels is a simple matter.

There is one condition we must define in terms of the robot's sonar readings, `ObstaclesInWay()`. We define this function to be true whenever any sonar range reading in the direction of the goal (within 45 degrees of the goal direction) is short:

```
private boolean ObstaclesInWay(angle goalAngle, sensorvals sonars) {
    int minSonarValue;
    minSonarValue=MinRange(sonars, goalAngle
                           -(pi/4),goalAngle+(pi/4));
    return (minSonarValue < 200);
} // end ObstaclesInWay() //
```

Note that the function `ComputeTranslation()` computes translational speed whether the robot is wall-following or heading toward the goal. In this simplified example, we define translation speed as being proportional to the largest range readings in the robot's approximate forward direction:

```
private int ComputeTranslation(sensorvals sonars) {
    int minSonarFront;
    minSonarFront = MinRange(sonars, -pi/4.0, pi/4.0);
    if (minSonarFront < 200) return 0;
    else return (Math.min(500, minSonarFront - 200));
} // end ComputeTranslation() //
```

There is a marked similarity between this approach and the potential field approach described in section 6.3.2. Indeed, some mobile robots implement obstacle avoidance by treating the current range readings of the robot as force vectors, simply carrying out vector addition to determine the direction of travel and speed. Alternatively, many will consider short-range readings to be repulsive forces, again engaging in vector addition to determine an overall motion command for the robot.

When faced with range sensor data, a popular way of determining rotation direction and speed is to simply subtract left and right range readings of the robot. The larger the difference, the faster the robot will turn in the direction of the longer range readings. The following two rotation functions could be used for our Bug2 implementation:

```
private int ComputeGoalSeekRot(angle goalAngle) {
    if (Math.abs(goalAngle) < pi/10) return 0;
    else return (goalAngle * 100);
} // end ComputeGoalSeekRot() //

private int ComputeRWFRot(sensorvals sonars) {
    int minLeft, minRight, desiredTurn;
    minRight = MinRange(sonars, -pi/2, 0);
    minLeft = MinRange(sonars, 0, pi/2);
    if (Math.max(minRight,minLeft) < 200) return (400);
                                          // hard left turn
    else {
        desiredTurn = (400 - minRight) * 2;
        desiredTurn = Math.inttorange(-400, desiredTurn, 400);
        return desiredTurn;
    } // end else
} // end ComputeRWFRot() //
```

Note that the rotation function for the case of right wall following combines a general avoidance of obstacles with a bias to turn right when there is open space on the right, thereby staying close to the obstacle's contour. This solution is certainly not the best solution for implementation of Bug2. For example, the wall follower could do a far better job by mapping the contour locally and using a PID control loop to achieve and maintain a specific distance from the contour during the right wall following action.

Although such simple obstacle avoidance algorithms are often used in simple mobile robots, they have numerous shortcomings. For example, the Bug2 approach does not take into account robot kinematics, which can be especially important with nonholonomic robots. Furthermore, since only the most recent sensor values are used, sensor noise can have a serious impact on real-world performance. The following obstacle avoidance techniques are designed to overcome one or more of these limitations.

6.4.2 Vector field histogram

Borenstein, together with Koren, developed the vector field histogram (VFH) [77]. Their previous work, which was concentrated on potential fields [176], was abandoned due to the method's instability and inability to pass through narrow passages. Later, Borenstein, together with Ulrich, extended the VFH algorithm to yield VFH+ [323] and VFH*[322].

One of the central criticisms of Bug-type algorithms is that the robot's behavior at each instant is generally a function of only its most recent sensor readings. This can lead to undesirable and yet preventable problems in cases where the robot's instantaneous sensor readings do not provide enough information for robust obstacle avoidance. The VFH techniques overcome this limitation by creating a local map of the environment around the robot. This local map is a small occupancy grid, as described in section 5.7 populated only by relatively

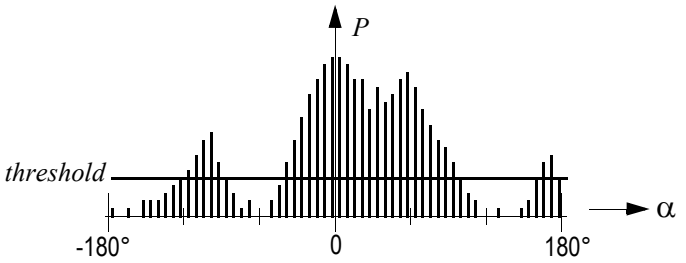


Figure 6.16
Polar histogram [177].

recent sensor range readings. For obstacle avoidance, VFH generates a polar histogram as shown in figure 6.16. The x-axis represents the angle α at which the obstacle was found, and the y-axis represents the probability P that there really is an obstacle in that direction based on the occupancy grid's cell values.

From this histogram a steering direction is calculated. First, all openings large enough for the vehicle to pass through are identified. Then a cost function is applied to every such candidate opening. The passage with the lowest cost is chosen. The cost function G has three terms:

$$G = a \cdot \text{target\_direction} + b \cdot \text{wheel\_orientation} + c \cdot \text{previous\_direction} \tag{6.15}$$

target_direction = alignment of the robot path with the goal;

wheel_orientation = difference between the new direction and the current wheel orientation;

previous_direction = difference between the previously selected direction and the new direction.

The terms are calculated such that a large deviation from the goal direction leads to a big cost in the term "target direction." The parameters a, b, c in the cost function G tune the behavior of the robot. For instance, a strong goal bias would be expressed with a large value for a. For a complete definition of the cost function, refer to [176].

In the VFH+ improvement, one of the reduction stages takes into account a simplified model of the moving robot's possible trajectories based on its kinematic limitations (e.g., turning radius for an Ackerman vehicle). The robot is modeled to move in arcs or straight lines. An obstacle thus blocks all of the robot's allowable trajectories that pass through the obstacle (figure 6.17a). This results in a masked polar histogram where obstacles are

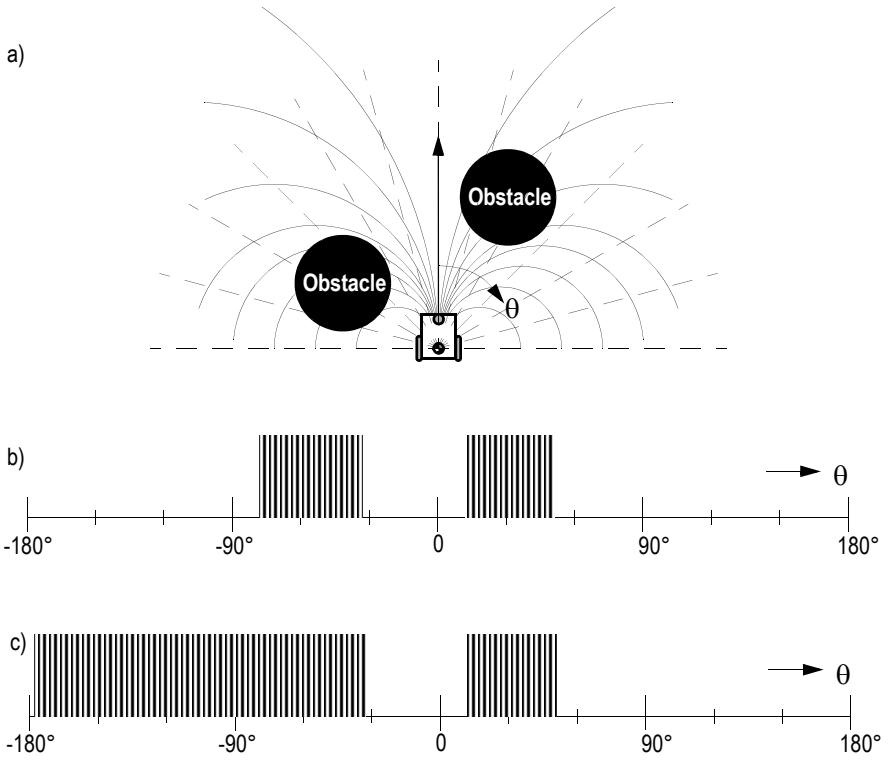


Figure 6.17
Example of blocked directions and resulting polar histograms [54]. (a) Robot and blocking obstacles. (b) Polar histogram. (b) Masked polar histogram.

enlarged so that all kinematically blocked trajectories are properly taken into account (figure 6.17c).

6.4.3 The bubble band technique

This idea is an extension for nonholonomic vehicles of the elastic band concept suggested by Khatib and Quinlan [166]. The original elastic band concept applied only to holonomic vehicles and so we focus only on the bubble band extension made by Khatib, Jaouni, Chatila, and Laumod [165].

A *bubble* is defined as the maximum local subset of the free space around a given configuration of the robot that which can be traveled in any direction without collision. The bubble is generated using a simplified model of the robot in conjunction with range information available in the robot's map. Even with a simplified model of the robot's geometry, it is possible to take into account the actual shape of the robot when calculating the bubble's

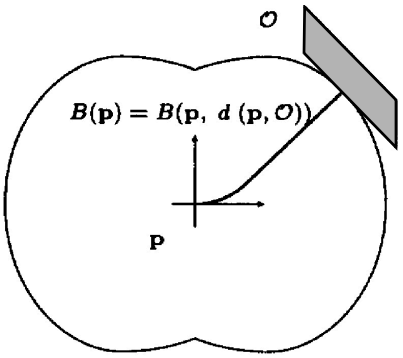


Figure 6.18
Shape of the bubbles around the vehicle. Courtesy of Raja Chatila [165].

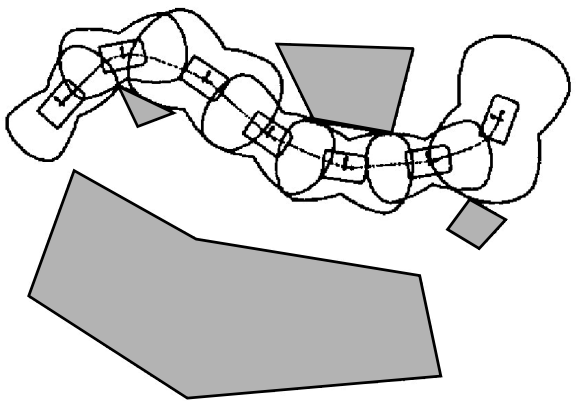


Figure 6.19
A typical bubble band. Courtesy of Raja Chatila [165].

size (figure 6.18). Given such bubbles, a band or string of bubbles can be used along the trajectory from the robot's initial position to its goal position to show the robot's expected free space throughout its path (see figure 6.19).

Clearly, computing the bubble band requires a global map and a global path planner. Once the path planner's initial trajectory has been computed and the bubble band is calculated, then modification of the planned trajectory ensues. The bubble band takes into account forces from modeled objects and internal forces. These internal forces try to minimize the "slack" (energy) between adjacent bubbles. This process, plus a final smoothing

operation, makes the trajectory smooth in the sense that the robot's free space will change as smoothly as possible during path execution.

Of course, so far this is more akin to path optimization than obstacle avoidance. The obstacle avoidance aspect of the bubble band strategy comes into play during robot motion. As the robot encounters unforeseen sensor values, the bubble band model is used to deflect the robot from its originally intended path in a way that minimizes bubble band *tension*.

An advantage of the bubble band technique is that one can account for the actual dimensions of the robot. However, the method is most applicable only when the environment configuration is well known ahead of time, just as with offline path-planning techniques.

6.4.4 Curvature velocity techniques

The basic curvature velocity approach. The curvature velocity approach (CVM) from Simmons [291] enables the actual kinematic constraints and even some dynamic constraints of the robot to be taken into account during obstacle avoidance, which is an advantage over more primitive techniques. CVM begins by adding physical constraints from the robot and the environment to a velocity space. The velocity space consists of rotational velocity ω and translational velocity v, thus assuming that the robot travels only along arcs of circles with curvature $c = \omega / v$.

Two types of constraints are identified: those derived from the robot's limitations in acceleration and speed, typically $-v_{max} < v < v_{max}$, $-\omega_{max} < \omega < \omega_{max}$; and, second, the constraints from obstacles blocking certain v and ω values due to their positions. The obstacles begin as objects in a Cartesian grid but are then transformed to the velocity space by calculating the distance from the robot position to the obstacle following some constant curvature robot trajectory, as shown in figure 6.20. Only the curvatures that lie within c_{min} and c_{max} are considered, since that curvature space will contain all legal trajectories.

To achieve real-time performance, the obstacles are approximated by circular objects, and the contours of the objects are divided into few intervals. The distance from an endpoint of an interval to the robot is calculated and in between the endpoints the distance function is assumed to be constant.

The final decision of a new velocity (v and ω) is made by an objective function. This function is only evaluated on that part of the velocity space that fulfills the kinematic and dynamic constraints as well as the constraints due to obstacles. The use of a Cartesian grid for initial obstacle representation enables straightforward sensor fusion if, for instance, a robot is equipped with multiple types of ranging sensors.

CVM takes into consideration the dynamics of the vehicle in useful manner. However a limitation of the method is the circular simplification of obstacle shape. In some environments this is acceptable, while in other environments such a simplification can cause seri-

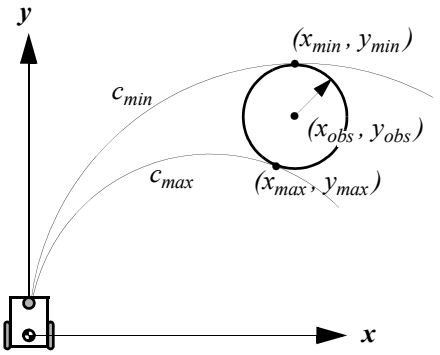


Figure 6.20
Tangent curvatures for an obstacle (from [291]).

ous problems. The CVM method can also suffer from local minima, since no *a priori* knowledge is used by the system.

The lane curvature method. Ko and Simmons presented an improvement of the CVM that they named the lane curvature method (LCM) [168], based on their experiences with the shortcomings of CVM, which had difficulty guiding the robot through intersections of corridors. The problems stemmed from the approximation that the robot moves only along fixed arcs, whereas in practice the robot can change direction many times before reaching an obstacle.

LCM calculates a set of desired lanes, trading off lane length and lane width to the closest obstacle. The lane with the best properties is chosen using an objective function. The local heading is chosen in such way that the robot will transition to the best lane if it is not in that lane already.

Experimental results have demonstrated better performance as compared to CVM. One caveat is that the parameters in the objective function must be chosen carefully to optimize system behavior.

6.4.5 Dynamic window approaches

Another technique for taking into account robot kinematics constraints is the dynamic window obstacle avoidance method. A simple but very effective dynamic model gives this approach its name. Two such approaches are represented in the literature. The dynamic window approach [130] of Fox, Burgard, and Thrun, and the global dynamic window approach [81] of Brock and Khatib.

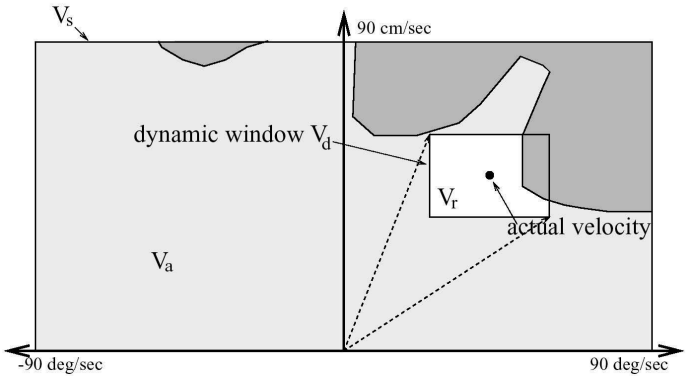


Figure 6.21
The dynamic window approach (courtesy of Dieter Fox [130]). The rectangular window shows the possible speeds (v, ω) and the overlap with obstacles in configuration space.

The local dynamic window approach. In the local dynamic window approach, the kinematics of the robot is taken into account by searching a well-chosen velocity space. The velocity space is all possible sets of tuples (v, ω) where v is the velocity and ω is the angular velocity. The approach assumes that robots move only in circular arcs representing each such tuple, at least during one timestamp.

Given the current robot speed, the algorithm first selects a *dynamic window* of all tuples (v, ω) that can be reached within the next sample period, taking into account the acceleration capabilities of the robot and the cycle time. The next step is to reduce the *dynamic window* by keeping only those tuples that ensure that the vehicle can come to a stop before hitting an obstacle. The remaining velocities are called admissible velocities. In figure 6.21, a typical dynamic window is represented. Note that the shape of the dynamic window is rectangular, which follows from the approximation that the dynamic capabilities for translation and rotation are independent.

A new motion direction is chosen by applying an objective function to all the admissible velocity tuples in the dynamic window. The objective function prefers fast forward motion, maintenance of large distances to obstacles and alignment to the goal heading. The objective function O has the form

$$O = a \cdot heading(v, \omega) + b \cdot velocity(v, \omega) + c \cdot dist(v, \omega) \tag{6.16}$$

heading = Measure of progress toward the goal location;

velocity = Forward velocity of the robot $\rightarrow$ encouraging fast movements;

dist = Distance to the closest obstacle in the trajectory.

The global dynamic window approach. The global dynamic window approach adds, as the name suggests, global thinking to the algorithm presented above. This is done by adding *NF1*, or grassfire, to the objective function *O* presented above (see section and figure 6.7). Recall that NF1 labels the cells in the occupancy grid with the total distance L to the goal. To make this faster, the global dynamic window approach calculates the NF1 only on a selected rectangular region that is directed from the robot toward the goal. The width of the region is enlarged and recalculated if the goal cannot be reached within the constraints of this chosen region.

This allows the global dynamic window approach to achieve some of the advantages of global path planning without complete a priori knowledge. The occupancy grid is updated from range measurements as the robot moves in the environment. The NF1 is calculated for every new updated version. If the NF1 cannot be calculated because the robot is surrounded by obstacles, the method degrades to the dynamic window approach. This keeps the robot moving so that a possible way out may be found and NF1 can resume.

The global dynamic window approach promises real-time, dynamic constraints, global thinking, and minimal free obstacle avoidance at high speed. An implementation has been demonstrated with an omnidirectional robot using a 450 MHz on-board PC. This system produced a cycle frequency of about 15 Hz when the occupancy grid was 30×30 m with a 5 cm resolution. Average robot speed in the tests was greater than 1 m/s.

6.4.6 The Schlegel approach to obstacle avoidance

Schlegel [280] presents an approach that considers the dynamics as well as the actual shape of the robot. The approach is adopted for raw laser data measurements and sensor fusion using a Cartesian grid to represent the obstacles in the environment. Real-time performance is achieved by use of precalculated lookup tables.

As with previous methods we have described, the basic assumption is that a robot moves in trajectories built up by circular arcs, defined as curvatures i_c. Given a certain curvature i_c Schlegel calculates the distance l_i to collision between a single obstacle point $[x, y]$ in the Cartesian grid and the robot, depicted in figure 6.22. Since the robot is allowed to be any shape, this calculation is time-consuming, and the result is therefore precalculated and stored in a lookup table.

For example, the search space window V_s is defined for a differential-drive robot to be all the possible speeds of the left and right wheels, v_r, v_l. The dynamic constraints of the robot are taken into account by refining V_s to only those values that are reachable within the next timestep, given the present robot motion. Finally, an objective function chooses the best speed and direction by trading off goal direction, speed, and distance until collision.

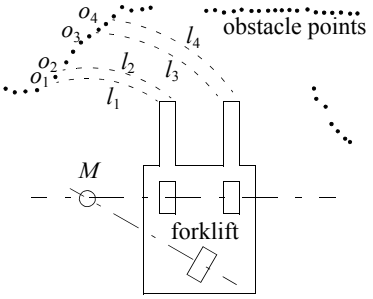


Figure 6.22
Distances l_i resulting from the curvature i_c, when the robot rotates around M (from [280]).

During testing Schlegel used a wavefront path planner. Two robot chassis were used, one with synchro-drive kinematics and one with tricycle kinematics. The tricycle-drive robot is of particular interest because it was a forklift with a complex shape that had a significant impact on obstacle avoidance. Thus the demonstration of reliable obstacle avoidance with the forklift is an impressive result. Of course, a disadvantage of this approach is the potential memory requirements for the lookup table. In their experiments, the authors used lookup tables of up to 2.5 Mb using a 6×6 m Cartesian grid with a resolution of 10 cm and 323 different curvatures.

6.4.7 Nearness diagram

Attempting to close a model fidelity gap in obstacle avoidance methods, the nearness diagram (ND) [222] can be considered to have some similarity to a VFH but solves several of its shortcomings, especially in very cluttered spaces. It was also used in [223] to take into account more precise geometric, kinematic, and dynamic constraints. This was achieved by breaking the problem down into generating the most promising direction of travel with the sole constraint a circular robot, then adapting this to the kinematic and dynamic constraints of the robot, followed by a correction for robot shape if is noncircular (only rectangular shapes were supported in the original publication). Global reasoning was added to the approach and termed the global nearness diagram (GND) in [225], somewhat similar to the GDWA extension to the DWA, but based on a workspace representation (instead of configuration space) and updating free space in addition to obstacle information.

6.4.8 Gradient method

Realizing that current computer technology allows fast recalculation of wavefront propagation techniques, the gradient method [171] formulates a grid global path planning that takes into account closeness to obstacles and allows generating continuous interpolations

of the gradient direction at any given point in the grid. The NF1 is a special case of the proposed algorithm, which calculates a navigation function at each timestep and uses the resulting gradient information to drive the robot toward the goal on a smooth path and not grazing obstacles unless necessary.

6.4.9 Adding dynamic constraints

Attempting to address the lack of dynamic models in most of the obstacle avoidance approaches discussed above, a new kind of space representation was proposed by Minguez, Montano, and Khatib in [224]. The ego-dynamic space is equally applicable to workspace and configuration space methods. It transforms obstacles into distances that depend on the braking constraints and sampling time of the underlying obstacle avoidance method. In combination with the proposed spatial window (PF) to represent acceleration capabilities, the approach was tested in conjunction with the ND and PF methods and gives satisfactory results for circular holonomic robots, with plans to extend it to nonholonomic, noncircular architectures.

6.4.10 Other approaches

The approaches described above are some of the most popularly referenced obstacle avoidance systems. There are, however, a great many additional obstacle avoidance techniques in the mobile robotics community. For example Tzafestas and Tzafestas [321] provide an overview of fuzzy and neurofuzzy approaches to obstacle avoidance. Inspired by nature, Chen and Quinn [98] present a biological approach in which they replicate the neural network of a cockroach. The network is then applied to a model of a four-wheeled vehicle.

The Liapunov functions form a well known theory that can be used to prove stability for nonlinear systems. In Vanualailai, Nakagiri, and Ha [326], the Liapunov functions are used to implement a control strategy for two-point masses moving in a known environment. All obstacles are defined as antitargets with an exact position and a circular shape. The antitargets are then used to build the control laws for the system.

6.4.11 Overview

Table 6.1 gives an overview on the presented obstacle-avoidance approaches.

Table 6.1
Overview of the most popular obstacle-avoidance algorithms

method																		
Vector Field Histogram (VFH)				Bug			model fidelity			view	other requisites			sensors	tested robots	performance		
VFH* [322]	VFH+ [176, 323]	VFH [77]	Tangent Bug [161]	Bug2 [198, 199]	Bug1 [198, 199]	shape	kinematics	dynamics		local map	global map	path planner			cycle time	architecture	remarks
circle	circle	simplistic	point	point	point				local				tactile				very inefficient, robust
basic	basic																
simplistic	simplistic																
essentially local	local	local	local	local	local												inefficient, robust
histogram grid	histogram grid	histogram grid	local tangent graph										tactile				
sonars	sonars	range	range														efficient in many cases, robust
nonholonomic (GuideCane)	nonholonomic (GuideCane)	synchro-drive (hexagonal)															
6 ... 242 ms	6 ms	27 ms															
66 MHz, 486 PC	66 MHz, 486 PC	20 MHz, 386 AT															
fewer local minima	local minima	local minima, oscillating trajectories															

Table 6.1
Overview of the most popular obstacle-avoidance algorithms

method		Dynamic window		Curvature velocity		Bubble band		model fidelity			view	other requisites			sensors	tested robots	performance		
								shape	kinematics	dynamics		local map	global map	path planner			cycle time	architecture	remarks
Global dynamic window [81]	Dynamic window approach [130]	Lane curvature method [168]	Curvature velocity method [291]	Bubble band [165]	Elastic band [166]														
circle	circle	circle	circle	C-space	C-space														
(holonomic)	exact	exact	exact	exact															
basic	basic	basic	basic																
global	local	local	local	local	global														
	obstacle line field	histogram grid	histogram grid																
C-space grid				polygonal	polygonal														
NF1				required	required														
180° FOV SCK laser scanner	24 sonars ring, 56 infrared ring, stereo camera	24 sonars ring, 30° FOV laser	24 sonars ring, 30° FOV laser																
holonomic (circular)	synchro-drive (circular)	synchro-drive (circular)	synchro-drive (circular)	various	various														
6.7 ms	250 ms	125 ms	125 ms																
450 MHz, PC	486 PC	200 MHz, Pentium	66 MHz, 486 PC																
turning into corridors	local minima	local minima	local minima, turning into corridors																

Table 6.1
Overview of the most popular obstacle-avoidance algorithms

method			model fidelity			view	other requisites			sensors	tested robots	performance		
			shape	kinematics	dynamics		local map	global map	path planner			cycle time	architecture	remarks
Other	Schlegel [280]	Nearness diagram [222, 223]	polygon	exact	basic	global		grid	wavefront	360° FOV laser scanner	synchrodrive (circular), tricycle (forklift)			allows shape change
			circle (but general formulation)	(holonomic)		local				180° FOV SCK laser scanner	holonomic (circular)			local minima
		Global nearness diagram [225]	circle (but general formulation)	(holonomic)		global	grid	NF1		180° FOV SCK laser scanner	holonomic (circular)			
	Gradient method [171]		circle	exact	basic	global		local perceptual space	fused	180° FOV distance sensor	nonholonomic (approx. circle)	100 ms (core algorithm: 10 ms)	266 MHz, Pentium	

6.5 Navigation Architectures

Given techniques for path planning, obstacle avoidance, localization, and perceptual interpretation, how do we combine all of these into one complete robot system for a real-world application? One way to proceed would be to custom-design an application-specific, monolithic software system that implements everything for a specific purpose. This may be efficient in the case of a trivial mobile robot application with few features and even fewer planned demonstrations. But for any sophisticated and long-term mobile robot system, the

issue of mobility architecture should be addressed in a principled manner. The study of *navigation architectures* is the study of principled designs for the software modules that constitute a mobile robot navigation system. Using a well-designed navigation architecture has a number of concrete advantages:

6.5.1 Modularity for code reuse and sharing

Basic software engineering principles embrace software modularity, and the same general motivations apply equally to mobile robot applications. But modularity is of even greater importance in mobile robotics because in the course of a single project the mobile robot hardware or its physical environmental characteristics can change dramatically, a challenge most traditional computers do not face. For example, one may introduce a Sick laser rangefinder to a robot that previously used only ultrasonic rangefinders. Or one may test an existing navigator robot in a new environment where there are obstacles that its sensors cannot detect, thereby demanding a new path-planning representation.

We would like to change part of the robot's competence without causing a string of side effects that force us to revisit the functioning of other robot competences. For instance we would like to retain the obstacle avoidance module intact, even as the particular ranging sensor suite changes. In a more extreme example, it would be ideal if the nonholonomic obstacle avoidance module could remain untouched even when the robot's kinematic structure changes from a tricycle chassis to a differential-drive chassis.

6.5.2 Control localization

Localization of robot control is an even more critical issue in mobile robot navigation. The basic reason is that a robot architecture includes multiple types of control functionality (e.g., obstacle avoidance, path planning, path execution, etc.). By localizing each functionality to a specific unit in the architecture, we enable individual testing as well as a principled strategy for control composition. For example, consider collision avoidance. For stability in the face of changing robot software, as well as for focused verification that the obstacle avoidance system is correctly implemented, it is valuable to localize all software related to the robot's obstacle avoidance process. At the other extreme, high-level planning and task decision making are required for robots to perform useful roles in their environment. It is also valuable to localize such high-level decision-making software, enabling it to be tested exhaustively in simulation and thus verified even without a direct connection to the physical robot. A final advantage of localization is associated with learning. Localization of control can enable a specific learning algorithm to be applied to just one aspect of a mobile robot's overall control system. Such targeted learning is likely to be the first strategy that yields successful integration of learning and traditional mobile robotics.

The advantages of localization and modularity provide a compelling case for the use of principled navigation architectures.

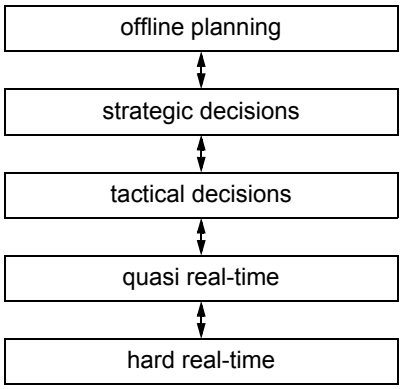


Figure 6.23
Generic temporal decomposition of a navigation architecture.

One way to characterize a particular architecture is by its decomposition of the robot's software. There are many favorite robot architectures, especially when one considers the relationship between artificial intelligence level decision making and lower-level robot control. For descriptions of such high-level architectures, refer to [2] and [39]. Here we concentrate on navigation competence. For this purpose, two decompositions are particularly relevant: temporal decomposition and control decomposition. In section 6.5.3 we define these two types of decomposition, then present an introduction to *behaviors*, which are a general tool for implementing control decomposition. Then, in section 6.5.4 we present three types of navigation architectures, describing for each architecture an implemented mobile robot case study.

6.5.3 Techniques for decomposition

Decompositions identify axes along which we can justify discrimination of robot software into distinct modules. Decompositions also serve as a way to classify various mobile robots into a more quantitative taxonomy. *Temporal decomposition* distinguishes between real-time and non real-time demands on mobile robot operation. *Control decomposition* identifies the way in which various control outputs within the mobile robot architecture combine to yield the mobile robot's physical actions. We will describe each type of decomposition in greater detail.

6.5.3.1 Temporal decomposition

A temporal decomposition of robot software distinguishes between processes that have varying real-time and non-real-time demands. Figure 6.23 depicts a generic temporal decomposition for navigation. In this figure, the most real-time processes are shown at the

bottom of the *stack*, with the highest category being occupied by processes with no real-time demands.

The lowest level in this example captures functionality that must proceed with a guaranteed fast cycle time, such as a 40 Hz bandwidth. In contrast, a quasi real-time layer may capture processes that require, for example, 0.1 second response time, with large allowable worst-case individual cycle times. A tactical layer can represent decision making that affects the robot's immediate actions and is therefore subject to some temporal constraints, while a strategic or offline layer represents decisions that affect the robot's behavior over the long term, with few temporal constraints on the module's response time.

Four important, interrelated trends correlate with temporal decomposition. These are not set in stone; there are exceptions. Nevertheless, these general properties of temporal decompositions are enlightening.

Sensor response time. A particular module's sensor response time can be defined as the amount of time between acquisition of a sensor event and a corresponding change in the output of the module. As one moves up the stack in figure 6.23, the sensor response time tends to increase. For the lowest-level modules, the sensor response time is often limited only by the raw processor and sensor speeds. At the highest-level modules, sensor response can be limited by slow and deliberate decision-making processes.

Temporal depth. Temporal depth is a useful concept applying to the temporal window that affects the module's output, both backward and forward in time. *Temporal horizon* describes the amount of look ahead used by the module during the process of choosing an output. *Temporal memory* describes the historical time span of sensor input that is used by the module to determine the next output. Lowest-level modules tend to have very little temporal depth in both directions, whereas the deliberative processes of highest-level modules make use of a large temporal memory and consider actions based on their long-term consequences, making note of large temporal horizons.

Spatial locality. Hand in hand with temporal span, the spatial impact of layers increases dramatically as one moves from low-level modules to high-level modules. Real-time modules tend to control wheel speed and orientation, controlling spatially localized behavior. High-level strategic decision making has little or no bearing on local position, but it informs global position far into the future.

Context specificity. A module makes decisions as a function not only of its immediate inputs but also as a function of the robot's context as captured by other variables, such as the robot's representation of the environment. Lowest-level modules tend to produce outputs directly as a result of immediate sensor inputs, using little context and therefore being

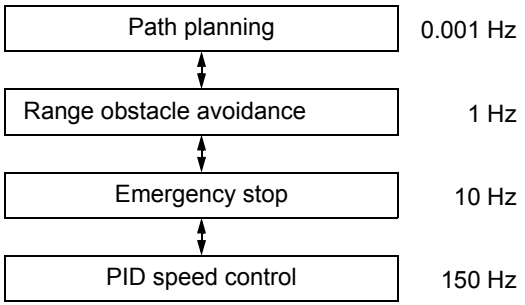


Figure 6.24
Sample four-level temporal decomposition of a simple navigating mobile robot. The column on the right indicates realistic bandwidth values for each module.

relatively context insensitive. Highest-level modules tend to exhibit very high context specificity. For strategic decision making, given the same sensor values, dramatically different outputs are nevertheless conceivable depending on other contextual parameters.

An example demonstrating these trends is depicted in figure 6.24, which shows a temporal decomposition of a simplistic navigation architecture into four modules. At the lowest level, the PID control loop provides feedback to control motor speeds. An emergency stop module uses short-range optical sensors and bumpers to cut current to the motors when it predicts an imminent collision. Knowledge of robot dynamics means that this module by nature has a greater temporal horizon than the PID module. The next module uses longer-range laser rangefinding sensor returns to identify obstacles well ahead of the robot and make minor course deviations. Finally, the path planner module takes the robot's initial and goal positions and produces an initial trajectory for execution, subject to change based on actual obstacles that the robot collects along the way.

Note that the cycle time, or bandwidth, of the modules changes by orders of magnitude between adjacent modules. Such dramatic differences are common in real navigation architectures, and so temporal decomposition tends to capture a significant axis of variation in a mobile robot's navigation architecture.

6.5.3.2 Control decomposition

Whereas temporal decomposition discriminates based on the time behavior of software modules, control decomposition identifies the way in which each module's output contributes to the overall robot control outputs. Presentation of control decomposition requires the evaluator to understand the basic principles of discrete systems representation and analysis. For a lucid introduction to the theory and formalism of discrete systems, see [25, 136].

Consider the robot algorithm and the physical robot instantiation (i.e., the robot form and its environment) to be members of an overall system whose connectivity we wish to

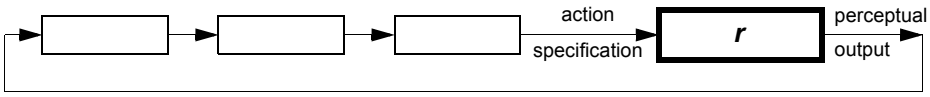


Figure 6.25
Example of a pure serial decomposition.

examine. This overall system S comprises a set M of modules, each module m connected to other modules via inputs and outputs. The system is *closed*, meaning that the input of every module m is the output of one or more modules in M. Each module has precisely one output and one or more inputs. The one output can be connected to any number of other modules inputs.

We further name a special module r in M to represent the physical robot and environment. Usually by r we represent the physical object on which the robot algorithm is intended to have impact, and from which the robot algorithm derives perceptual inputs. The module r contains one input and one output line. The input of r represents the complete action specification for the physical robot. The output of r represents the complete perceptual output to the robot. Of course the physical robot may have many possible degrees of freedom and, equivalently, many discrete sensors. But for this analysis we simply imagine the entire input/output vector, thus simplifying *r* to just one input and one output. For simplicity, we will refer to the input of r as O and to the robot's sensor readings I. From the point of view of the rest of the control system, the robot's sensor values I are inputs, and the robot's actions O are the outputs, explaining our choice of I and O.

Control decomposition discriminates between different types of control pathways through the portion of this system comprising the robot algorithm. At one extreme, depicted in figure 6.25 we can consider a perfectly linear, or sequential control pathway.

Such a serial system uses the internal state of all associated modules and the value of the robot's percept I in a sequential manner to compute the next robot action O. A pure serial architecture has advantages relating to predictability and verifiability. Since the state and outputs of each module depend entirely on the inputs it receives from the module upstream, the entire system, including the robot, is a single well-formed loop. Therefore, the overall behavior of the system can be evaluated using well-known discrete forward simulation methods.

Figure 6.26 depicts the extreme opposite of pure serial control, a fully parallel control architecture. Because we choose to define *r* as a module with precisely one input, this parallel system includes a special module n that provides a single output for the consumption of r. Intuitively, the fully parallel system distributes responsibility for the system's control output O across multiple modules, possibly simultaneously. In a pure sequential system, the control flow is a linear sequence through a string of modules. Here, the control flow

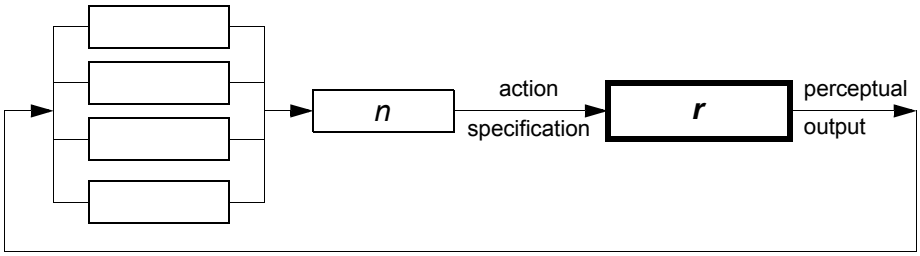


Figure 6.26
Example of a pure parallel decomposition.

contains a *combination* step at which point the result of multiple modules may impact O in arbitrary ways.

Thus parallelization of control leads to an important question: how will the output of each component module inform the overall decision concerning the value of O? One simple combination technique is temporal switching. In this case, called *switched parallel*, the system has a parallel decomposition but at any particular instant in time the output O can be attributed to one specific module. The value of O can of course depend on a different module at each successive time instant, but the instantaneous value of O can always be determined based on the functions of a single module. For instance, suppose that a robot has an obstacle avoidance module and a path-following module. One switched control implementation may involve execution of the path-following recommendation whenever the robot is more than 50 cm from all sensed obstacles and execution of the obstacle-avoidance recommendation when any sensor reports a range closer than 50 cm.

The advantage of such switched control is particularly clear if switching is relatively rare. If the behavior of each module is well understood, then it is easy to characterize the behavior of the switched control robot: it will obstacle avoid at times, and it will path-follow other times. If each module has been tested independently, there is a good chance the switched control system will also perform well. Two important disadvantages must be noted. First, the overall behavior of the robot can become quite poor if the switching is itself a high-frequency event. The robot may be unstable in such cases, switching motion modes so rapidly as to dramatically devolve into behavior that is neither path-following nor obstacle-avoiding. Another disadvantage of switched control is that the robot has no path-following bias when it is obstacle avoiding (and vice versa). Thus in cases where control *ought* to mix recommendations from among multiple modules, the switched control methodology fails.

In contrast, the much more complex *mixed parallel* model allows control at any given time to be shared between multiple modules. For example, the same robot could take the obstacle avoidance module's output at all times, convert it to a velocity vector, and combine

it with the path-following module's output using vector addition. Then the output of the robot would never be due to a single module, but would result from the mathematical combination of both modules outputs. Mixed parallel control is more general than switched control, but by that token it is also a more challenging technique to use well. Whereas with switched control most poor behavior arises out of inopportune switching behavior, in mixed control the robot's behavior can be quite poor even more readily. Combining multiple recommendations mathematically does not guarantee an outcome that is globally superior, just as combining multiple vectors when deciding on a swerve direction to avoid an obstacle can result in the very poor decision of going straight ahead. Thus, great care must be taken in mixed parallel control implementations to fashion mixture formulas and individual module specifications that lead to effective mixed results.

Both the switched and mixed parallel architectures are popular in the behavior robotics community. Arkin [2] proposes the *motor-schema* architecture in which *behaviors* (i.e., modules in the earlier discussion) map sensor value vectors to motor value vectors. The output of the robot algorithm is generated, as in mixed parallel systems, using a linear combination of the individual behavior outputs. In contrast, Maes [201, 202] produces a switched parallel architecture by creating a *behavior network* in which a behavior is chosen discretely by comparing and updating activation levels for each behavior. The subsumption architecture of Brooks [82] is another example of a switched parallel architecture, although the active model is chosen via a suppression mechanism rather than activation level. For further discussion, see [2].

One overall disadvantage of parallel control is that verification of robot performance can be extremely difficult. Because such systems often include truly parallel, multithreaded implementations, the intricacies of robot-environment interaction and sensor timing required to represent properly all conceivable module-module interactions can be difficult or impossible to simulate. So, much testing in the parallel control community is performed empirically using physical robots.

An important advantage of parallel control is its biomimetic aspect. Complex organic organisms benefit from large degrees of true parallelism (e.g., the human eye), and one goal of the parallel control community is to understand this biologically common strategy and leverage it to advantage in robotics.

6.5.4 Case studies: tiered robot architectures

We have described temporal and control decompositions of robot architecture, with the common theme that the roboticist is always composing multiple modules together to make up that architecture. Let us turn again toward the overall mobile robot navigation task with this understanding in mind. Clearly, robot behaviors play an important role at the real-time levels of robot control, for example, path-following and obstacle avoidance. At higher temporal levels, more tactical tasks need to modulate the activation of behaviors, or modules,

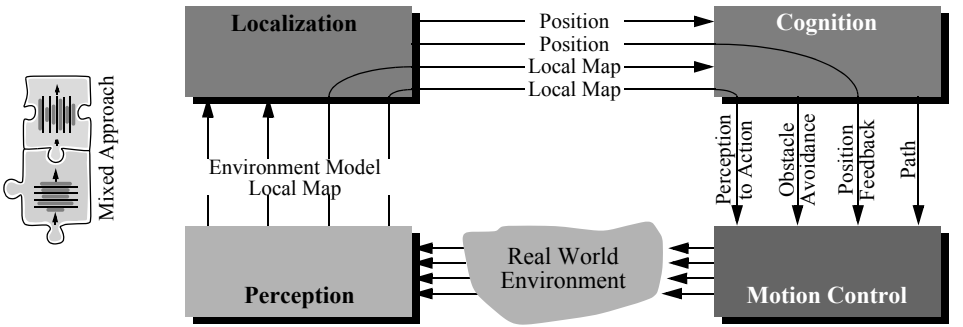


Figure 6.27
The basic architectural example used throughout this text.

in order to achieve robot motion along the intended path. Higher still, a global planner could generate paths to provide tactical tasks with global foresight.

In chapter 1, we introduced a functional decomposition showing such modules of a mobile robot navigator from the perspective of information flow. The relevant figure is shown here again as figure 6.27.

In such a representation, the arcs represent aspects of real-time and non-real-time competence. For instance, obstacle avoidance requires little input from the localization module and consists of fast decisions at the cognition level followed by execution in motion control. In contrast, PID position feedback loops bypass all high-level processing, tying the perception of encoder values directly to lowest-level PID control loops in motion control. The trajectory of arcs through the four software modules provides temporal information in such a representation.

Using the tools of this chapter, we can now present this same architecture from the perspective of a temporal decomposition of functionality. This is particularly useful because we wish to discuss the interaction of strategic, tactical, and real-time processes in a navigation system.

Figure 6.28 depicts a generic tiered architecture based on the approach of Pell and colleagues [256] used in designing an autonomous spacecraft, *Deep Space One*. This figure is similar to figure 6.24 in presenting a temporal decomposition of robot competence. However, the boundaries separating each module from adjacent modules are specific to robot navigation.

Path planning embodies strategic-level decision making for the mobile robot. Path planning uses all available global information in non-real-time to identify the right sequence of local actions for the robot. At the other extreme, *real-time control* represents competences requiring high bandwidth and tight sensor-effector control loops. At its lowest level, this

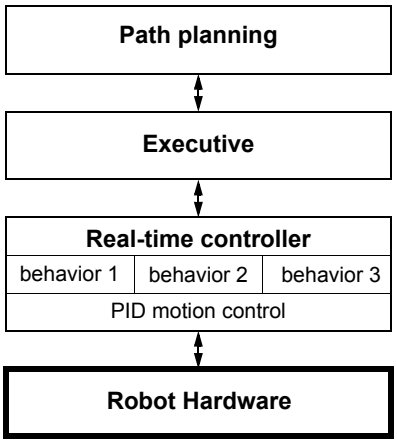


Figure 6.28
A general tiered mobile robot navigation architecture based on a temporal decomposition.

includes motor velocity PID loops. Above those, real-time control also includes low-level behaviors that may form a switch or mixed parallel architecture.

In between the path planner and real-time control tiers sits the *executive*, which is responsible for mediating the interface between planning and execution. The executive is responsible for managing the activation of behaviors based on information it receives from the planner. The executive is also responsible for recognizing failure, saving (placing the robot in a stable state), and even reinitiating the planner as necessary. It is the executive in this architecture that contains all tactical decision making as well as frequent updates of the robot's short-term memory, as is the case for localization and mapping.

It is interesting to note the similarity between this general architecture, used in many specialized forms in mobile robotics today, and the architecture implemented by Shakey, one of the very first mobile robots, in 1969 [242]. Shakey had *LLA* (low-level actions) that formed the lowest architectural tier. The implementation of each LLA included the use of sensor values in a tight loop just as in today's behaviors. Above that, the middle architectural tier included the *ILA* (intermediate-level actions), which would activate and deactivate LLA as required based on perceptual feedback during execution. Finally, the topmost tier for Shakey was STRIPS (Stanford Research Institute Planning System), which provided global look ahead and planning, delivering a series of tasks to the intermediate executive layer for execution.

Although the general architecture shown in figure 6.28 is useful as a model for robot navigation, variant implementations in the robotics community can be quite different. Next, we present three particular versions of the general tiered architecture, describing for each

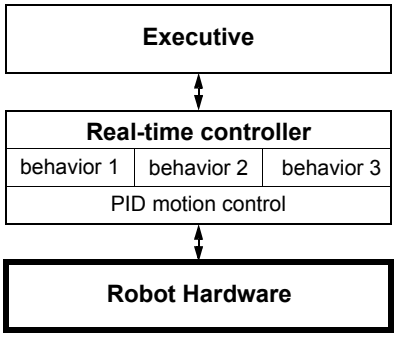


Figure 6.29
A two-tiered architecture for offline planning.

version at least one real-world mobile robot implementation. For broader discussions of various robot architectures, see [39].

6.5.4.1 Offline planning

Certainly the simplest possible integration of planning and execution is no integration at all. Consider figure 6.29, in which there are only two software tiers. In such navigation architectures, the executive does not have a planner at its disposal but must contain a priori all relevant schemes for traveling to desired destinations.

The strategy of leaving out a planner altogether is of course extremely limiting. Moving such a robot to a new environment demands a new instantiation of the navigation system, and so this method is not useful as a general solution to the navigation problem. However such robotic systems do exist, and this method can be useful in two cases.

Static route applications. In mobile robot applications where the robot operates in a completely static environment using a route navigation system, it is conceivable that the number of discrete goal positions is so small that the environmental representation can directly contain paths to all desired goal points. For example, in factory or warehouse settings, a robot may travel a single looping route by following a buried guidewire. In such industrial applications, path-planning systems are sometimes altogether unnecessary when a precompiled set of route solutions can be easily generated by the robot programmers. The Chips mobile robot is an example of a museum robot that also uses this architecture [251]. Chips operates in a unidirectional looping track defined by its colored landmarks. Furthermore, it has only twelve discrete locations at which it is allowed to stop. Due to the simplicity of this environmental model, Chips contains an executive layer that directly caches

the path required to reach each goal location rather than a generic map with which a path planner could search for solution paths.

Extreme reliability demands. Not surprisingly, another reason to avoid online planning is to maximize system reliability. Since planning software can be the most sophisticated portion of a mobile robot's software system, and since in theory at least planning can take time exponential to the complexity of the problem, imposing hard temporal constraints on successful planning is difficult if not impossible. By computing all possible solutions offline, the industrial mobile robot can trade versatility for effective constant-time planning (while sacrificing significant memory of course). A real-world example of offline planning for this reason can be seen in the contingency plans designed for space shuttle flights. Instead of requiring astronauts to solve problems online, thousands of conceivable issues are postulated on Earth, and complete conditional plans are designed and published in advance of the shuttle flights. The fundamental goal is to provide an absolute upper limit on the amount of time that passes before the astronauts begin resolving the problem, sacrificing a great deal of ground time and paperwork to achieve this performance guarantee.

6.5.4.2 Episodic planning

The fundamental information-theoretic disadvantage of planning offline is that, during runtime, the robot is sure to encounter perceptual inputs that provide information, and it would be rational to take this additional information into account during subsequent execution. Episodic planning is the most popular method in mobile robot navigation today because it solves this problem in a computationally tractable manner.

As shown in figure 6.30, the structure is three-tiered, as is the general architecture of figure 6.28. The intuition behind the role of the planner is as follows. Planning is computationally intensive, and therefore planning too frequently would have serious disadvantages. But the executive is in an excellent position to identify when it has encountered enough information (e.g., through feature extraction) to warrant a significant change in strategic direction. At such points, the executive will invoke the planner to generate, for example, a new path to the goal.

Perhaps the most obvious condition that triggers replanning is detection of a blockage on the intended travel path. For example, in [281] the path-following behavior returns failure if it fails to make progress for a number of seconds. The executive receives this failure notification, modifies the short-term occupancy grid representation of the robot's surroundings, and launches the path planner in view of this change to the local environment map.

A common technique to delay planning until more information has been acquired is called *deferred planning*. This technique is particularly useful in mobile robots with dynamic maps that become more accurate as the robot moves. For example, the commercially available Cye robot can be given a set of goal locations. Using its grassfire breadth-

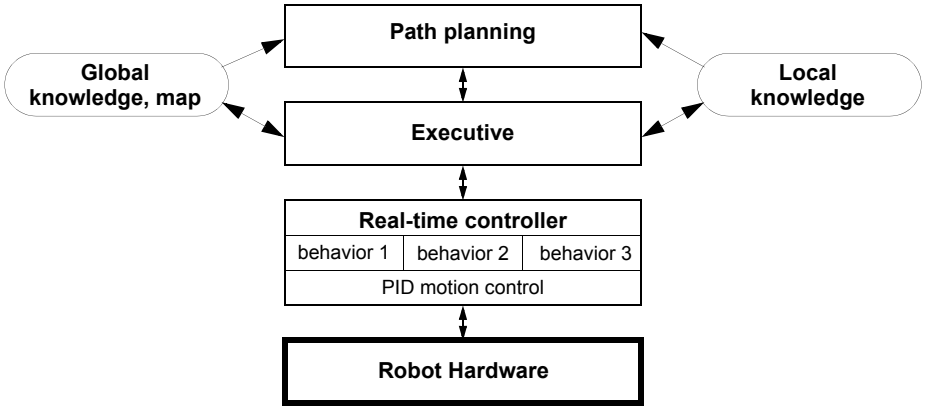


Figure 6.30
A three-tiered episodic planning architecture.

first planning algorithm, this robot will plot a detailed path to the closest goal location only and will execute this plan. Upon reaching this goal location, its map will have changed based on the perceptual information extracted during motion. Only then will Cye's executive trigger the path planner to generate a path from its new location to the next goal location.

The robot Pygmalion implements an episodic planning architecture along with a more sophisticated strategy when encountering unforeseen obstacles in its way [58, 259]. When the lowest-level behavior fails to make progress, the executive attempts to find a way past the obstacle by turning the robot 90 degrees and trying again. This is valuable because the robot is not kinematically symmetric, and so servoing through a particular obstacle course may be easier in one direction than the other.

Pygmalion's environment representation consists of a continuous geometric model as well as an abstract topological network for route planning. Thus, if repeated attempts to clear the obstacle fail, then the robot's executive will temporarily cut the topological connection between the two appropriate nodes and will launch the planner again, generating a new set of waypoints to the goal. Next, using recent laser rangefinding data as a type of local map (see figure 6.30), a geometric path planner will generate a path from the robot's current position to the next waypoint.

In summary, episodic planning architectures are extremely popular in the mobile robot research community. They combine the versatility of responding to environmental changes and new goals with the fast response of a tactical executive tier and behaviors that control real-time robot motion. As shown in figure 6.30, it is common in such systems to have both a short-term local map and a more strategic global map. Part of the executive's job in such

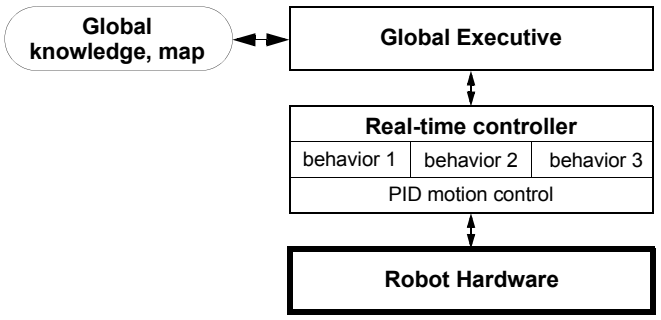


Figure 6.31
An integrated planning and execution architecture in which planning is nothing more than a real-time execution step (behavior).

dual representations is to decide when and if new information integrated into the local map is sufficiently nontransient to be copied into the global knowledge base.

6.5.4.3 Integrated planning and execution

Of course, the architecture of a commercial mobile robot must include more functionality than just navigation. But limiting this discussion to the question of *navigation* architectures leads to what may at first seem a degenerate solution.

The architecture shown in figure 6.31 may look similar to the offline planning architecture of figure 6.29, but in fact it is significantly more advanced. In this case, the planner tier has disappeared because there is no longer a temporal decomposition between the executive and the planner. Planning is simply one small part of the executive's nominal cycle of activities, where the local and global representations are the same. The advantage of this approach is that the robot's actions at every cycle are guided by a global path planner, and they are therefore optimal in view of all of the information the robot has gathered.

Integrated planning and execution has largely been made feasible due to innovations in graph search algorithms (e.g. the D* algorithm described on page page 385) and graph representation (e.g., state lattice graphs, whose edges can be inherently executed by the robotic platform). As a result, formidable real-time implementations devoid of obstacle avoidance modules have emerged: Pivtoraiko et al. [260] showed that graph search on a 3D state lattice (including 2D position and heading) can be as efficient as 2D grid search. In the work of Ferguson et al. [127], an extension to this strategy was successfully employed to navigate an autonomous car in large scale parking lots. Rufli et al. [271, 272] added velocity dimensions to the aforementioned state lattice representation which allowed them to take into account position and velocity information of nearby dynamic obstacles during the planning step. The result is a feasible, globally optimal, time-parametrized path that inherently

avoids dynamic obstacles, a task that has traditionally been carried out by local collision avoidance modules.

The described methods naturally face limits of applicability as the size of the environment increases. This issue can be accounted for by applying multiresolution graph approaches, however. Close to the robot, a high-fidelity lattice may be employed, farther away a lower-resolution one. Hundreds of meters distant, the lattice may transition into a road network such as the ones often used in commercial GPS-based navigation systems.

Still, the recent success of integrated planning and execution methods underlines the fact that the designer of a robot navigation architecture must consider not only all aspects of the robot and its environmental task but also the state of processor, GPU, and memory technology. We expect that mobile robot architecture design is sure to remain an active area of innovation for years to come. All forms of technological progress, from robot sensor inventions to processor speed increases, and further parallelization are likely to catalyze new innovations in mobile robot architecture as previously unimaginable tactics become realizable.

6.6 Problems

1. Consider completeness and optimality properties for each of:

Visibility graph
Voronoi diagram
Exact cell decomposition
Approximate cell decomposition

In the framework of path planning, categorize for each whether it is complete/incomplete, and optimal/ not guaranteed-optimal for path planning.

2. Consider an Ackerman steering 4-wheel high speed Martian rover. Consider all the obstacle avoidance techniques described in 6.2. Explain for every option its advantage or disadvantage, in one sentence each, for this specific application. Specifically do so for: Schlegel, local dynamic window, LCM, CVM, VFH, Bubble band, Bug.

3. Consider an autonomous driving robot for highway driving. Propose a temporal decomposition, as in figure 6.24, with at least five levels, describing control frequency at each level and the specific driving skills/behaviors incorporated at that level.

4. Challenge Question.
Consider a robot that navigates with range sensors that have a limited useful range r. Propose a path-planning method based on the ones in 6.3 that is complete and maintains a safe distance from objects while also staying within distance r from objects whenever possible.

Bibliography

Books

[1] Adams, M.D., *Sensor Modelling: Design and Data Processing for Autonomous Navigation*. World Scientific Series in Robotics and Intelligent Systems. Singapore, World Scientific Publishing, 1999.

[2] Arkin, R.C., *Behavioral Robotics*. Cambridge MA, MIT Press, 1998.

[3] Bar-Shalom, Y., Li, X.-R., *Estimation and Tracking: Principles, Techniques, and Software.* Norwood, MA, Artech House, 1993.

[4] Benosman, R., Kang, S. B., *Panoramic Vision: Sensors, Theory, and Applications*, New York, Springer-Verlag, 2001.

[5] Borenstein, J., Everett, H.R., Feng, L., *Navigating Mobile Robots: Systems and Techniques*. Natick, MA, A.K. Peters, Ltd., 1996.

[6] Borenstein, J., Everett, H.R., Feng, L., *Where Am I? Sensors and Methods for Mobile Robot Positioning.* Technical report, Ann Arbor, University of Michigan, 1996. Available at http://www-personal.engin.umich.edu/~johannb/position.htm.

[7] Bradski, G., Kaehler, A., *Learning OpenCV: Computer Vision with the OpenCV Library*, Sebastopol, CA, O'Reilly Media, Inc., 1st edition, 2008.

[8] Breipohl, A.M., *Probabilistic Systems Analysis: An Introduction to Probabilistic Models, Decisions, and Applications of Random Processes.* New York, John Wiley & Sons, 1970.

[9] Bundy, A. (editor), *Artificial Intelligence Techniques: A Comprehensive Catalogue*. New York, Springer-Verlag, 1997.

[10] Canudas de Wit, C., Siciliano, B., and Bastin G. (editors), *Theory of Robot Control*. New York, Spinger, 1996.

[11] Carroll, R.J., Ruppert, D., *Transformation and Weighting in Regression*. New York, Chapman and Hall, 1988.

[12] Cox, I.J., Wilfong, G.T. (editors), *Autonomous Robot Vehicles*. New York, Springer-Verlag, 1990.

[13] Craig, J.J., *Introduction to Robotics: Mechanics and Control*. 2nd edition. Boston, Addison-Wesley, 1989.

[14] De Silva, C.W., *Control Sensors and Actuators.* Upper Saddle River, NJ, Prentice-Hall, 1989.

[15] Daniillidis, K., Klette, R., *Imaging Beyond the Pinhole Camera*. New York, Springer, 2006.

[16] Dietrich, C.F., *Uncertainty, Calibration and Probability*. Bristol, UK, Adam Hilger, 1991.

[17] Draper, N.R., Smith, H., *Applied Regression Analysis*. 3rd edition. New York, John Wiley & Sons, 1988.

[18] Duda, R.O., Hart, P.E., Stork, D.G., *Pattern Classification*. New York, Wiley, 2001.

[19] Duda, R. O., Hart, P.E. *Pattern Classification and Scene Analysis*. New York, John Wiley & Sons, 1973.

[20] Everett, H.R., *Sensors for Mobile Robots: Theory and Applications*. New York, Natick, MA, A.K. Peters, Ltd., 1995.

[21] Faugeras, O., *Three-Dimensional Computer Vision: A Geometric Viewpoint*. Cambridge, MA, MIT Press, 1993.

[22] Faugeras, O., Luong, Q.T., *The Geometry of Multiple Images*. Cambridge, MA, MIT Press, 2001.

[23] Floreano, D., Zufferey, J.C., Srinivasan, M.V., Ellington, C., *Flying Insects and Robots*, Springer, 2009.

[24] Forsyth, D. A., Ponce, J., *Computer Vision: A Modern Approach*. Upper Saddle River, NJ, Prentice Hall, 2003.

[25] Genesereth, M.R., Nilsson, N.J., *Logical Foundations of Artificial Intelligence.* Palo Alto, CA, Morgan Kaufmann, 1987.

[26] Gonzalez, R., Woods, R., *Digital Image Processing*. 3rd edition. New York, Pearson Prentice Hall, 2008.

[27] Hammond, J. H., *The Camera Obscura: A Chronicle*. Bristol, UK, Adam Hilger, 1981.

[28] Haralick, R.M., Shapiro, L.G., *Computer and Robot Vision, 1+2*. Boston, Addison-Wesley, 1993.

[29] Hartley, R.I., Zisserman, A. *Multiple View Geometry*. Cambridge, UK, Cambridge University Press, 2004.

[30] Jones, J., Flynn, A., *Mobile Robots, Inspiration to Implementation*. Natick, MA, A.K. Peters, Ltd., 1993.

[31] Kortenkamp, D., Bonasso, R.P., Murphy, R.R. (editors), *Artificial Intelligence and Mobile Robots; Case Studies of Successful Robot Systems*. Cambridge, MA, AAAI Press / MIT Press, 1998.

[32] Latombe, J.C., *Robot Motion Planning*. Norwood, MA, Kluwer Academic, 1991.

[33] LaValle, S.M. *Planning Algorithms*, Cambridge, UK, Cambridge University Press, 2006.

[34] Lee, D., *The Map-Building and Exploration Strategies of a Simple Sonar-Equipped Mobile Robot*. Cambridge, UK, Cambridge University Press, 1996.

[35] Leonard, J.E., Durrant-Whyte, H.F., *Directed Sonar Sensing for Mobile Robot Navigation*. Norwood, MA, Kluwer Academic, 1992.

[36] Ma, Y., S. Soatto, S., Kosecka, J., Sastry, S., *An Invitation to 3-D Vision: From Images to Geometric Models*. New York, Springer-Verlag, 2003.

[37] Manyika, J., Durrant-Whyte, H.F., *Data Fusion and Sensor Management: A Decentralized Information-Theoretic Approach*. Palo Alto, CA, Ellis Horwood, 1994.

[38] Mason, M., *Mechanics of Robotics Manipulation*. Cambridge, MA, MIT Press, 2001.

[39] Murphy, R.R., *Introduction to AI Robotics*. Cambridge, MA, MIT Press, 2000.

[40] Nourbakhsh, I., *Interleaving Planning and Execution for Autonomous Robots*. Norwood, MA, Kluwer Academic, 1997.

[41] Papoulis, A. *Probability, Random Variables, and Stochastic Processes*, 4th edition. New York, McGraw-Hill, 2001.

[42] Raibert, M.H., *Legged Robots That Balance*. Cambridge, MA, MIT Press, 1986.

[43] Ritter, G.X., Wilson, J.N., *Handbook of Computer Vision Algorithms in Image Algebra*. Boca Raton, FL, CRC Press, 1996.

[44] Russell, S., Norvig, P., *Artificial Intelligence: A Modern Approach*. 3rd edition. New York, Prentice Hall International, 2010.

[45] Schraft, R.D., Schmierer, G., *Service Roboter*. Natick, MA, A.K. Peters, Ltd, 2000.

[46] Sciavicco, L., Siciliano, B., *Modeling and Control of Robot Manipulators*. New York, McGraw-Hill, 1996.

[47] Siciliano, B., Khatib, O., *Springer Handbook of Robotics*, Springer, 2008.

[48] Slama, C.C., *Manual of Photogrammetry*. 4th edition. Falls Church VA, American Society of Photogrammetry,1980.

[49] Szeliski, R., *Computer Vision: Algorithms and Applications*, New York, Springer, 2010.

[50] Tennekes, H., *The Simple Science of Flight: From Insects to Jumbo Jets*. Cambridge, MA, MIT Press, 1996.

[51] Thrun, S., Burgard, W., Fox, D., *Probabilistic Robotics*. Cambridge, MA, MIT Press, 2005.

[52] Todd, D.J, *Walking Machines: An Introduction to Legged Robots*. London, Kogan Page Ltd, 1985.

[53] Trucco, E., Verri, A., *Introductory Techniques for 3-D Computer Vision*. New York, Prentice Hall, 1998.

[54] Zufferey, J.C., *Bio-inspired Flying Robots: Experimental Synthesis of Autonomous Indoor Flyers*, EPFL Press, 2008.

Papers

[55] Aho, A.V., "Algorithms for finding patterns in strings," in J. van Leeuwen (editor), *Handbook of Theoretical Computer Science*, Cambridge, MA, MIT Press, 1990, Volume A, chapter 5, 255–300.

[56] Angeli, A., Filliat, D., Doncieux, S., Meyer, J.A., "Fast and incremental method for loop-closure detection using bags of visual words," *IEEE Transactions on Robotics*, 24(5): 1027–1037, October, 2008.

[57] Arras, K.O., Castellanos, J.A., Siegwart, R., "Feature multi-hypothesis localization and tracking for mobile robots using geometric constraints," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'2002)*, Washington, DC, May , 2002.

[58] Arras, K.O., Persson, J., Tomatis, N., Siegwart, R., "Real-time obstacle avoidance for polygonal robots with a reduced dynamic window," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2002),* Washington, DC, May, 2002.

[59] Arras, K.O., Siegwart, R.Y., "Feature extraction and scene interpretation for map navigation and map building," in *Proceedings of SPIE, Mobile Robotics XII*, 1997.

[60] Arras, K.O., Tomatis, N., "Improving robustness and precision in mobile robot localization by using laser range finding and monocular vision," in *Proceedings of the Third European Workshop on Advanced Mobile Robots (Eurobot 99)*, Zurich, September, 1999.

[61] Astolfi, A., "Exponential stabilization of a mobile robot," *in Proceedings of 3rd European Control Conference,* Rome, September, 1995.

[62] Bailey, T., Durrant-Whyte, H., "Simultaneous localization and mapping: Part II," *IEEE Robotics and Automation Magazine*, 108–117, 2006.

[63] Bailey, T., "Mobile robot localisation and mapping in extensive outdoor environments," Ph.D. thesis, University of Sydney, 2002.

[64] Baker, S., Nayar, S., "A theory of single-viewpoint catadioptric image formation," *International Journal of Computer Vision* 35, no. 2: 175–196, 1999.

[65] Barnard, K., Cardei V., Funt, B., "A comparison of computational color constancy algorithms," *IEEE Transactions on Image Processing* 11: 972–984, 2002.

[66] Barreto, J. P., Araujo, H., "Issues on the geometry of central catadioptric image formation. *International Conference on Computer Vision and Pattern Recognition* (CVPR), 2001.

[67] Barreto, J. P., Araujo, H., "Fitting conics to paracatadioptric projection of lines," *Computer Vision and Image Understanding* 101(3): 151–165. March, 2006.

[68] Barreto, J. P., Araujo, H., "Geometric properties of central catadioptric line images and their application in calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8): 1237–1333, August 2005.

[69] Barron, J.L., Fleet, D.J., Beauchemin, S.S., "Performance of optical flow techniques," *International Journal of Computer Vision*, 12: 43–77, 1994.

[70] Batavia, P., Nourbakhsh, I., "Path planning for the cye robot," *in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'00)*, Takamatsu, Japan, November 2000.

[71] Bay, H., Ess, A., Tuytelaars, T., Van Gool, L., "Speeded-up robust features (SURF)," *International Journal on Computer Vision and Image Understanding* 110, no. 3: 346–359, 2008.

[72] Besl, P., McKay, N., "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence* (PAMI) 14, no. 2: 239–256, February 1992.

[73] Bicchi, A., Marigo, A., Piccoli, B., "On the reachability of quantized control systems," *IEEE Transactions on Automatic Control*,. 4, no. 47: 546–563, 2002.

[74] Biederman, I., "Recognition-by-components: A theory of human image understanding," *Psychological Review*, 2, no. 94: 115–147, 1987.

[75] Blackwell, D., "Conditional expectation and unbiased sequential estimation," *Annals of Mathematical Statistics* 18: 105–110, 1947.

[76] Blösch, M., Weiss, S., Scaramuzza, D., Siegwart, R., "Vision based MAV navigation in unknown and unstructured environments," *IEEE International Conference on Robotics and Automation* (ICRA 2010), Anchorage, Alaska, May 2010.

[77] Borenstein, J., Koren, Y., "The vector field histogram – fast obstacle avoidance for mobile robots." *IEEE Journal of Robotics and Automation* 7: 278–288, 1991.

[78] Borges, G. A., Aldon, M.-J., "Line Extraction in 2D Range Images for Mobile Robotics," *Journal of Intelligent and Robotic Systems* 40: 267–297, 2004.

[79] Bosse, M., Newman, P., Leonard, J., Teller, S., "Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework," *International Journal of Robotics Research* 23, no. 12: 1113–1139, 2004.

[80] Bosse, M., Rikoski, R., Leonard, J., Teller, S., "Vanishing points and 3d lines from omnidirectional video," *International Conference on Image Processing*, 2002.

[81] Brock, O., Khatib, O., "High-speed navigation using the global dynamic window approach," *in Proceeding of the IEEE International Conference on Robotics and Automation*, Detroit, May 1999.

[82] Brooks, R., "A robust layered control system for a mobile robot," *IEEE Transactions of Robotics and Automation*, RA-2:14–23, March 1986.

[83] Brown, H.B., Zeglin, G.Z., "The bow leg hopping robot", *in Proceedings of the IEEE International Conference on Robotics and Automation*, Leuwen, Belgium, May 1998.

[84] Bruce, J., Balch,T., and Veloso, M., "Fast and inexpensive color image segmentation for interactive robots," *in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'00)*, Takamatsu, Japan, 2000.

[85] Burgard,W., Cremers, A., Fox, D., Hahnel, D., Lakemeyer, G., Schulz, D., Steiner, W., Thrun, S., "Experiences with an interactive museum tour-guide robot," *Artificial Intelligence* 114: 1–53, 2000.

[86] Burgard, W., Derr, A., Fox, D., Cremers, A., "Integrating Global Position Estimation and Position Tracking for Mobile Robots: The Dynamic Markov Localization Approach," *in Proceedings of the 1998 IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS'98)*, Victoria, Canada, October 1998.

[87] Burgard, W., Fox, D., Henning, D., "Fast grid-based position tracking for mobile robots," *in Proceedings of the 21th German Conference on Artificial Intelligence (KI97)*, Freiburg, Germany, Springer-Verlag, 1997.

[88] Burgard, W., Fox, D., Jans, H., Matenar, C., Thrun, S., "sonar mapping of large-scale mobile robot environments using EM," *in Proceedings of the International Conference on Machine Learning*, Bled, Slovenia, 1999.

[89] Cabani, C., Mac Lean, W. J., "Implementation of an affine-covariant feature detector in field-programmable gate arrays," in *Proceedings of the International Conference on Computer Vision Systems*, 2007.

[90] Campion, G., Bastin, G., D'Andréa-Novel, B., "Structural properties and classification of kinematic and dynamic models of wheeled mobile robots." *IEEE Transactions on Robotics and Automation* 12, no. 1: 47–62, 1996.

[91] Canny, J. F., "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 679–698, 1986.

[92] Canudas de Wit, C., Sordalen, O.J., "Exponential stabilization of mobile robots with nonholonomic constraints." *IEEE Transactions on Robotics and Automation* 37: 1791–1797, 1993.

[93] Caprari, G., Estier, T., Siegwart, R., "Fascination of down scaling–alice the sugar cube robot." *Journal of Micro-Mechatronics* 1: 177–189, 2002.

[94] Caprile, B., Torre, V., "Using vanishing points for camera calibration." *International Journal of Computer Vision*. 4: 127–140, 1990.

[95] Castellanos, J.A., Tardos, J.D., Schmidt, G., "Building a global map of the environment of a mobile robot: The importance of correlations," in *Proceedings of the* 1997 *IEEE Conference on Robotics and Automation*, Albuquerque, NM, April 1997.

[96] Castellanos, J.A., Tardos, J.D., "Laser-based segmentation and localization for a mobile robot," in *Robotics and Manufacturing: Recent Trends in Research and Applications*, volume 6. ASME Press, 1996.

[97] Censi, A., Carpin, S., "HSM3D: Feature-less global 6DOF scan-matching in the hough/radon domain," *IEEE International Conference on Robotics and Automation* (ICRA), 2009.

[98] Chen, C.T., Quinn, R.D., "A crash avoidance system based upon the cockroach escape response circuit," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Albuquerque, NM, April 1997.

[99] Chenavier, F., Crowley, J.L., "Position estimation for a mobile robot using vision and odometry," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, France, May 1992.

[100] Cheeseman, P., Smith, P. "On the representation and estimation of spatial uncertainty," *International Journal of Robotics* 5: 56–68, 1986.

[101] Chomat, O., Colin deVerdiere, V., Hall, D., Crowley, J., "Local scale selection for gaussian based description techniques," in *Proceedings of the European Conference on Computer Vision*, Dublin, Ireland, 117–133, 2000.

[102] Chong, K.S., Kleeman, L., "Accurate odometry and error modelling for a mobile robot," *in Proceedings of the IEEE International Conference on Robotics and Automation*, Albuquerque, NM, April 1997.

[103] Choset, H., Walker, S., Eiamsa-Ard, K., Burdick, J., "Sensor exploration: Incremental construction of the hierarchical generalized voronoi graph." *The International Journal of Robotics Research* 19: 126–148, 2000.

[104] Collins, A. Ruina, R. Tedrake, M. Wisse, "Efficient bipedal robots based on passive-dynamic walkers," *Science* 307, no. 5712: 1082 - 1085, 2005.

[105] Csorba, M. "Simultaneous localisation and map building," *Ph.D. thesis*, University of Oxford, Oxford, 1997.

[106] Cox, I.J., Leonard, J.J., "Modeling a dynamic environment using a bayesian multiple hypothesis approach," *Artificial Intelligence* 66: 311–44, 1994.

[107] Corke, P.I., Strelow, D., Singh, S., "Omnidirectional visual odometry for a planetary rover," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

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

[109] Cummins, M., Newman, P., "Highly scalable appearance-only SLAM – FAB-MAP 2.0," *In Robotics Science and Systems (RSS)*, Seattle, USA, June 2009.

[110] Davison, A.J., "Real-time simultaneous localisation and mapping with a single camera," *International Conference on Computer Vision*, 2003.

[111] Davison, A.J. "Active search for real-time vision," *In International Conference on Computer Vision*, 2005.

[112] Davison, A. J., Reid, I., Molton, N., Stasse, O., "MonoSLAM: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, no. 6, June, 2007.

[113] Dellaert, F. "Square root SAM," *Proceedings of the Robotics Science and Systems Conference*, 2005.

[114] Dijkstra, E.W. "A note on two problems in connexion with graphs," *Numerische Mathematik* 1: 269–271, 1959.

[115] Dowlingn, K., Guzikowski, R., Ladd, J., Pangels, H., Singh, S., Whittaker, W.L., "NAVLAB: An autonomous navigation testbed," *Technical report CMU-RI-TR-87-24, Robotics Institute*, Pittsburgh, Carnegie Mellon University, November 1987.

[116] Duckett, T., Marsland, S.,Shapiro, J. "Learning globally consistent maps by relaxation," *IEEE International Conference on Robotics and Automation*, 2000.

[117] Duckett, T., Marsland, S.,Shapiro, J. "Fast, on-line learning of globally consistent maps," *Autonomous Robots* 12, no. 3: 287–300, 2002.

[118] Dudek, G., Jenkin, M., "Inertial sensors, GPS, and odometry," *Springer Handbook of Robotics*, Springer, 2008.

[119] Dugan, B., "Vagabond: A demonstration of autonomous, robust outdoor navigation," *in Video Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 1993.

[120] Durrant-Whyte, H., Bailey, T., "Simultaneous localization and mapping: Part I," *IEEE Robotics and Automation Magazine*, 99–108, 2006.

[121] Einsele, T., "Real-time self-localization in unknown indoor environments using a panorama laser range finder," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 697–702, 1997.

[122] Elfes, A., "Sonar real world mapping and navigation," in [12].

[123] Ens, J., Lawrence, P., "An investigation of methods for determining depth from focus." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15: 97–108, 1993.

[124] Espenschied, K.S., Quinn, R.D., "Biologically-inspired hexapod robot design and simulation," *in AIAA Conference on Intelligent Robots in Field, Factory, Service and Space*, Houston, Texas, March, 1994.

[125] Falcone, E., Gockley, R., Porter, E., Nourbakhsh, I., "The personal rover project: the comprehensive design of a domestic personal robot," *Robotics and Autonomous Systems, Special Issue on Socially Interactive Robots* 42: 245–258, 2003.

[126] Feder, H.J.S., Slotine, J-J.E., "Real-time path planning using harmonic potentials in dynamic environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Albuquerque, NM, April 1997.

[127] Ferguson, D., Howard, T., Likhachev, M., "Motion planning in urban environments: Part II," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS), 2008.

[128] Fischler, M. A., Bolles, R. C. "RANSAC random sampling concensus: A paradigm for model fitting with applications to Image analysis and automated cartography,". *Communications of ACM* 26: 381–395, 1981.

[129] Fox, D., "KLD-sampling: Adaptive particle filters and mobile robot localization," *Advances in Neural Information Processing Systems 14*. MIT Press, 2001.

[130] Fox, D., Burgard,W., Thrun, S., "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine* 4: 23–33, 1997.

[131] Fraundorfer, F., Engels, C., Nister, D., "Topological mapping, localization and navigation using image collections," *IEEE/RSJ Conference on Intelligent Robots and Systems* 1, 2007.

[132] Freedman, B., Shpunt, A., Machline, M., Arieli, Y., "Depth mapping using projected patterns," *US Patent no. US20100118123A1*, May 13, 2010. http://www.freepatentsonline.com/20100118123.pdf

[133] Fusiello, A., Trucco, E., Verri, A., "A compact algorithm for rectification of stereo pairs," *Machine Vision and Applications*, 12(1): 16–22, 2000.

[134] Gächter, S., Harati, A., Siegwart, R., "Incremental object part detection toward object classification in a sequence of noisy range images," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2008)*, Pasadena, USA, May 2008.

[135] Gander,W., Golub, G.H., Strebel, R., "Least-squares fitting of circles and ellipses," *BIT Numerical Mathematics* 34, no. 4: 558–578, December 1994.

[136] Genesereth, M.R. "Deliberate agents," *Technical Report Logic-87-2.* Stanford, CA, Stanford University, Logic Group, 1987.

[137] Geyer, C., Daniilidis, K., "A unifying theory for central panoramic systems and practical applications," *European Conference on Computer Vision* (ECCV), 2000.

[138] Goedeme, T., Nuttin, M., Tuytelaars, T., Van Gool, L., "Markerless computer vision based localization using automatically generated topological maps," *European Navigation Conference* GNSS, Rotterdam, 2004.

[139] Golfarelli, M., Maio, D., Rizzi, S. "Elastic correction of dead-reckoning errors in map building," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1998.

[140] Golub, G., Kahan,W., "Calculating the singular values and pseudo-inverse of a matrix." *Journal SIAM Numerical Analysis* 2: 205–223, 1965.

[141] Grisetti, G., Stachniss, C., Grzonka, S., Burgard, W., "A tree parameterization for efficiently computing maximum likelihood maps using gradient descent," *Robotics Science and Systems* (RSS), 2007.

[142] Grzonka, S., Grisetti, G., Burgard, W. "Towards a navigation system for autonomous indoor flying," *IEEE International Conference on Robotics and Automation*, 2009.

[143] Gutmann, J.S., Burgard, W., Fox, D., Konolige, K., "An experimental comparison of localization methods," in *Proceedings of the 1998 IEEE/RSJ International. Conference of Intelligent Robots and Systems* (IROS'98), Victoria, Canada, October 1998.

[144] Guttman, J.S., Konolige, K., "Incremental mapping of large cyclic environments," in P*roceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, Monterey, November 1999.

[145] Hähnel, D., Fox, D., Burgard, W., Thrun, S. "A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements," *Proceedings of the Conference on Intelligent Robots and Systems*, 2003.

[146] Harris, C., Stephens, M., "A combined corner and edge detector," *Proceedings of the 4th Alvey Vision Conference*, 1988.

[147] Hart, P. E., Nilsson, N. J., Raphael, B. "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics* 4, no. 2: 100–107, 1968.

[148] Hashimoto, S., "Humanoid robots in Waseda University—Hadaly-2 and WABIAN," *in IARP First International Workshop on Humanoid and Human Friendly Robotics*, Tsukuba, Japan, October 1998.

[149] Heale, A., Kleeman, L.: "A real time DSP sonar echo processor," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'00)*, Takamatsu, Japan, 2000.

[150] Heymann, S., Maller, K., Smolic, A., Froehlich, B., Wiegand, T., "SIFT implementation and optimization for general-purpose GPU," *in Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2007.

[151] Horn, B.K.P., Schunck, B.G., "Determining optical flow," *Artificial Intelligence*, 17: 185–203, 1981.

[152] Horswill, I., "Visual collision avoidance by segmentation," in *Proceedings of IEEE International Conference on Robotics and Automation*, 902–909, 1995, IEEE Press, Munich, November 1994.

[153] Hoyt, D.F., Taylor, C.R, "Gait and the energetics of locomotion in horses," *Nature* 292: 239–240, 1981.

[154] Jacobs, R. and Canny, J., "Planning smooth paths for mobile robots," in *Proceeding. of the IEEE Conference on Robotics and Automation*, IEEE Press, 2–7, 1989.

[155] Jeffreys, H. and Jeffreys, B. S. "Methods of mathematical physics," *Cambridge, Cambridge University Press*, 305-306, 1988.

[156] Jennings, J., Kirkwood-Watts, C., Tanis, C., "Distributed map-making and navigation in dynamic environments," in *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98)*, Victoria, Canada, October 1998.

[157] Jensfelt, P., Austin, D., Wijk, O., Andersson, M., "Feature based condensation for mobile robot localization," in *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, May 24–28, 2000.

[158] Jensfelt, P., Christensen, H., "Laser based position acquisition and tracking in an indoor environment," in *Proceedings of the IEEE International Symposium on Robotics and Automation* 1, 1998.

[159] Jogan, M., Leonardis, A. "Robust localization using panoramic viewbased recognition," in *Proceedings of ICPR00* 4: 136–139, 2000.

[160] Jung, I., Lacroix, S., "Simultaneous localization and mapping with stereovision," in *Proceedings of the 11th International Symposium Robotics Research*, Siena, Italy, 2005.

[161] Kamon, I., Rivlin, E., Rimon, E., "A new range-sensor based globally convergent navigation algorithm for mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Minneapolis, April 1996.

[162] Kelly, A., "Pose determination and tracking in image mosaic based vehicle position estimation," in *Proceeding of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'00)*, Takamatsu, Japan, 2000.

[163] Khatib, O., Real-time obstacle avoidance for manipulators and mobile robots, *International Journal of Robotics Research* 5, no. 1, 1986.

[164] Khatib, M., Chatila, R., "An extended potential field approach for mobile robot sensor motions," in *Proceedings of the Intelligent Autonomous Systems IAS-4*, IOS Press, Karlsruhe, Germany, March 1995, 490–496.

[165] Khatib, M., Jaouni, H., Chatila, R., Laumod, J.P., "Dynamic path modification for car-like nonholonomic mobile robots," in *Proceedings of IEEE International Conference on Robotics and Automation*, Albuquerque, NM, April 1997.

[166] Khatib, O., Quinlan, S., "Elastic bands: connecting, path planning and control," in *Proceedings of IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 1993.

[167] Klein, G., Murray, D., "Parallel Tracking and Mapping for Small AR Workspaces," *Proceedings of the International Symposium on Mixed and Augmented Reality* (ISMAR'07), Nara, Japan, 2007.

[168] Ko, N.Y., Simmons, R., "The lane-curvature method for local obstacle avoidance," in *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98)*, Victoria, Canada, October 1998.

[169] Koenig, S., Simmons, R., "Xavier: A robot navigation architecture based on partially observable markov decision process models," in [31].

[170] Koenig, S., Likhachev, M., "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics* 21(3): 354–363, 2005.

[171] Konolige, K.,. "A gradient method for realtime robot control," in *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, Takamatsu, Japan, 2000.

[172] Konolige, K., "Small vision systems: Hardware and implementation," in P*roceedings of Eighth International Symposium on Robotics Research*, Hayama, Japan, October 1997.

[173] Konolige, K., "Large-scale map-making," *AAAI National Conference on Artificial Intelligence*, 2004.

[174] Konolige, K., Agrawal, M., Solà, J., "Large scale visual odometry for rough terrain," *International Symposium on Research in Robotics* (ISRR), November, 2007.

[175] Koperski, K., Adhikary, J., Han, J., "Spatial data mining: Progress and challenges survey paper," in *Proceedings of the ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, Montreal, June 1996.

[176] Koren, Y., Borenstein, J., "High-speed obstacle avoidance for mobile robotics," in *Proceedings of the IEEE Symposium on Intelligent Control* 382–384, Arlington, VA, August 1988.

[177] Koren, Y., Borenstein, J., "Real-time obstacle avoidance for fast mobile robots in cluttered environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Los Alamitos, CA, May 1990.

[178] Kruppa, E., "Zur ermittlung eines objektes aus zwei perspektiven mit innerer orientierung," *Sitzungsberichte Österreichische Akademie der Wissenschaften, Mathematisch-naturwissenschaftliche Klasse, Abteilung II* a, volume 122: 1939-1948, 1913.

[179] Kuipers, B., Byun, Y.T., "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations," *Journal of Robotics and Autonomous Systems*, 8: 47–63, 1991.

[180] Kuo, A., "Choosing your steps carfully," *Robotics & Automation Magazine*, 2007.

[181] Lacroix, S., Mallet, A., Chatila, R., Gallo, L., "Rover self localization in planetary-like environments," *in Proc. Int. Symp. Artic. Intell., Robot., Autom. Space* (i-SAIRAS), Noordwijk, The Netherlands, 1999.

[182] Lamon, P., Nourbakhsh, I., Jensen, B., Siegwar,t R., "Deriving and matching image fingerprint sequences for mobile robot localization," in *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, Seoul, Korea, May 2001.

[183] Latombe, J.C., Barraquand, J., "Robot motion planning: A distributed presentation approach." *International Journal of Robotics Research*, 10: 628–649, 1991.

[184] Lauria, M., Estier, T., Siegwart, R.: "An innovative space rover with extended climbing abilities," in *Video Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, San Francisco, May 2000.

[185] LaValle, S. M., "Rapidly-exploring random trees: A new tool for path planning," *Technical Report, Computer Science Dept*., Iowa State University, October 1998.

[186] Lavalle, S. M.: "Rapidly-exploring random trees: Progress and prospects," In *Algorithmic and Computational Robotics: New Directions*, pp. 293-308, 2000.

[187] Lazanas, A., Latombe, J.C., "Landmark robot navigation," in *Proceedings of the Tenth National Conference on AI*. San Jose, CA, July 1992.

[188] Lazanas, A. Latombe, J.C., "Motion planning with uncertainty: A landmark approach." *Artificial Intelligence*, 76: 285–317, 1995.

[189] Lee, S.-O., Cho, Y.-J., Hwang-Bo, M., You, B.-J., Oh, S.-R.: "A stabile target-tracking control for unicycle mobile robots," in *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Takamatsu, Japan, 2000.

[190] Leonard, J.J., Rikoski, R.J., Newman, P.M., Bosse, M., "Mapping partially observable features from multiple uncertain vantage points," *International Journal of Robotics Research* 21, no. 10: 943–975, 2002.

[191] Likhachev, M., Gordon, G., Thrun, S. "ARA*: Anytime A* with provable bounds on sub-optimality," *Advances in Neural Information Processing Systems* (NIPS), 2003.

[192] Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., Thrun, S., "Anytime dynamic A*: An anytime, replanning algorithm," *Proceedings of the International Conference on Automated Planning and Scheduling* (ICAPS), 2005.

[193] Lindeberg, T., "Feature detection with automatic scale selection," *International Journal of Computer Vision* 30, no. 2: 79-116, 1998.

[194] Longuet-Higgins, H.C., "A computer algorithm for reconstructing a scene from two projections," *Nature* 293: 133–135, September, 1981.

[195] Louste, C. and Liegois, A., Path planning for non-holonomic vehicles: a potential viscous fluid method, Robotica 20: 291–298, 2002.

[196] Lowe, David G., "Object recognition from local scale-invariant features," *Proceedings of the International Conference on Computer Vision,* 1999.

[197] Lowe, D. G., "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision* 60 (2): 91-110, 2004.

[198] Lumelsky, V., Skewis, T., "Incorporating range sensing in the robot navigation function," *IEEE Transactions on Systems, Man, and Cybernetics* 20: 1058–1068, 1990.

[199] Lumelsky, V., Stepanov, A., "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape," in [12].

[200] Lu, F., Milios, E. "Globally consistent range scan alignment for environment mapping," *Autonomous Robots* 4: 333–349,1997.

[201] Maes, P., "The dynamics of action selection," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, 1989.

[202] Maes, P., "Situated Agents Can Have Goals," *Robotics and Autonomous Systems*, 6: 49–70. 1990.

[203] Maimone, M., Cheng, Y., Matthies, L., "Two years of visual odometry on the mars exploration rovers," *Journal on Field Robotics* 24, no. 3: 169–186, 2007.

[204] Makadia, A., Patterson, A., Daniilidis, K., "Fully automatic registration of 3D point clouds," *IEEE Conference on Computer Vision and Pattern Recognition*, New York, June 2006.

[205] Martinelli, A., Siegwart, R., "Estimating the odometry error of a mobile robot during navigation," in *Proceedings of the European Conference on Mobile Robots (ECMR 2003)*, Warsaw, September 4–6, 2003.

[206] Masoud, S.A., Masoud, A.A., "Motion planning in the presence of directional and regional avoidance constraints unsing nonlinear, anisotropic, harmonic potential fields: a physical metaphor," *IEEE Transactions on Systems, Man and Cybernetics* 32, no. 6: 705–723, 2002.

[207] Masoud, S.A., Masoud, A.A., "Kinodynamic motion planning: a novel type of nonlinear, passive damping forces and advantages," *IEEE Robotics Automation Magazine* 17, no. 1: 85–99, 2010.

[208] Matsumoto, Y., Inaba, M., Inoue, H., "Visual navigation using viewsequenced route representation," *IEEE International Conference on Robotics and Automation*, 1996.

[209] Maybeck,P.S., "The Kalman filter: An introduction to concepts," in [12].

[210] Matas, J., Chum, O., Urban, M., Pajdla, T., "Robust wide-baseline stereo from maximally stable extremal regions," in *Proceedings of the British Machine Vision Conference*, 384–393, 2002.

[211] McGeer, T., "Passive dynamic walking," *International Journal of Robotics Research* 9, no. 2: 62–82, 1990.

[212] Mei, C., Rives, P., "Single view point omnidirectional camera calibration from planar grids," *IEEE International Conference on Robotics and Automation* (ICRA), 2007.

[213] Menegatti, E., Maedab, T., Ishiguro, H., "Image-based memory for robot navigation using properties of omnidirectional images," *Robotics and Autonomous System* 47, no. 4: 251–267, July, 2004.

[214] Meng, M., Kak, A.C.. "Mobile robot navigation using neural networks and nonmetrical environmental models," *IEEE Control Systems Magazine*, 13(5): 30–39, October 1993.

[215] Metropolis, N., Ulam, S. "The Monte Carlo method," *Journal of the American Statistical Association* 44, no. 247: 335–341, 1949.

[216] Mikolajczyk, K., C. Schmid, "Indexing based on scale-invariant interest points," *in Proceedings of the International Conference on Computer Vision*, 525–531, Vancouver, Canada, 2001.

[217] Mikolajczyk, K., Schmid, C., "Scale and affine invariant interest point detectors," *International Journal of Computer Vision* 1, no. 60: 63–86, 2004.

[218] Mikolajcyk, K. and Schmid, C., "An affine invariant interest point detector," *in Proceedings of the 7th European Conference on Computer Vision*, Denmark, 2002.

[219] Mikolajczyk, K., "Scale and Affine Invariant Interest Point Detectors," PhD thesis, INRIA Grenoble, 2002.

[220] Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T.,Van Gool, L. "A comparison of affine region detectors," *International Journal of Computer Vision*, 65(1-2): 43–72, 2005.

[221] Minetti, A.E. ,Ardigò, L.P., Reinach, E., Saibene, F., "The relationship between mechanical work and energy expenditure of locomotion in horses," *Journal of Experimental Biology* 202, no. 17, 1999.

[222] Minguez, J., Montano, L., "Nearness diagram navigation (ND): A new real time collision avoidance approach," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Takamatsu, Japan, October 2000.

[223] Minguez, J., Montano, L., "Robot navigation in very complex, dense, and cluttered indoor / outdoor environments," in *Proceeding of International Federation of Automatic Control (IFAC2002)*, Barcelona, April 2002.

[224] Minguez, J., Montano, L., Khatib, O., "Reactive collision avoidance for navigation with dynamic constraints," in *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[225] Minguez, J., Montano, L., Simeon, T., Alami, R., "Global nearness diagram navigation (GND)," in *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, 2001.

[226] Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D. and Martinoli, A. "The e-puck, a robot designed for education in engineering," *The 9th Conference on Autonomous Robot Systems and Competitions*, 2009.

[227] Montiel, J.M.M. , Civera, J., Davison, A.J., "Unified inverse depth parametrization for monocular SLAM," *Proc. of the Robotics Science and Systems Conference*, 2006.

[228] Moutarlier, P., Chatila, R., "An experimental system for incremental environment modeling by an autonomous mobile robot," *1st International Symposium on Experimental Robotics*, 1989.

[229] Moutarlier, P., Chatila, R. "Stochastic multisensory data fusion for mobile robot location and environment modeling," *5th Int. Symposium on Robotics Research*, 1989.

[230] Montano, L., Asensio, J.R., "Real-time robot navigation in unstructured environments using a 3D laser range finder," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robot and Systems*, IROS 97, September 1997.

[231] Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B. "FastSLAM: A factored solution to the simultaneous localization and mapping problem," *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2002.

[232] Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B. "Fast-SLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," *International Joint Conference on Artificial Intelligence*, 2003.

[233] Moravec, H. and Elfes, A.E., "High Resolution Maps from Wide Angle Sonar," in *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, March 1985.

[234] Moravec, H. P., "Towards automatic visual obstacle avoidance," *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, 1977.

[235] Moravec, H. P., "Visual mapping by a robot rover," *International Joint Conference on Artificial Intelligence*, 1979.

[236] Moravec, H., "Obstacle avoidance and navigation in the real world by a seeing robot rover," *PhD thesis*, Stanford University, 1980.

[237] Moutarlier, P., Chatila, R., "Stochastic multisensory data fusion for mobile robot location and environment modelling," in *Proceedings of the 5th International Symposium of Robotics Research*, Tokyo, 1989.

[238] Murillo, A.C., Kosecka, J., "Experiments in Place Recognition using Gist Panoramas," *Proceedings of the International Workshop on Omnidirectional Vision* (OMNIVIS'09), 2009.

[239] Murphy, K., Russell, S. "Rao-Blackwellized particle filtering for dynamic Bayesian networks," *In Sequential Monte Carlo Methods in Practice*, ed. by A. Doucet, N. de Freitas, N. Gordon, 499–516, Springer, 2001.

[240] Nayar, S.K., "Catadioptric omnidirectional camera." *IEEE CVPR*, 482–488, 1997.

[241] Nayar, S., Watanabe, M., and Noguchi, M., "Real-time focus range sensor." *In Fifth International Conference on Computer Vision*, 995–1001, Cambridge, Massachusetts, 1995.

[242] Nilsson, N.J., "Shakey the robot." *SRI, International, Technical Note*, Menlo Park, CA, 1984, No. 325.

[243] Nistér, D. Stewénius, H., "Scalable recognition with a vocabulary tree," *IEEE International Conference on Computer Vision and Pattern Recognition*, 2006.

[244] Nistér, D., Naroditsky, O., Bergen, J., "Visual odometry for ground vehicle applications," *Journal of Field Robotics* 23, no. 1: 3–20, 2006.

[245] Nistér, D., Naroditsky, O., Bergen, J., "Visual odometry," *IEEE International Conference on Computer Vision and Pattern Recognition*, 2004.

[246] Nistér, D., "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence* (PAMI), 26(6): 756-770, June 2004.

[247] Nguyen, V., Martinelli, A., Tomatis, N., Siegwart, R. "A comparison of line extraction algorithms using 2D laser rangefinder for indoor mobile robotics," *IEEE/RSJ Intenational Conference on Intelligent Robots and Systems*, IROS, 2005.

[248] Noth, André, "Design of solar powered airplanes for continuous flight," *Ph.D. thesis, Autonomous Systems Lab, ETH Zurich*, Switzerland, December 2008.

[249] Nourbakhsh, I.R., "Dervish: An office-navigation robot," in [31].

[250] Nourbakhsh, I.R., Andre. D., Tomasi, C., Genesereth, M.R., "Mobile robot obstacle avoidance via depth from focus," *Robotics and Autonomous Systems*, 22: 151–158, 1997.

[251] Nourbakhsh, I.R., Bobenage, J., Grange, S., Lutz, R., Meyer, R, Soto, A., "An affective mobile educator with a full-time job," *Artificial Intelligence*, 114: 95–124, 1999.

[252] Nourbakhsh, I.R., Powers, R., Birchfield, S., "DERVISH, an office-navigation robot." *AI Magazine*, 16: 39–51, summer 1995.

[253] Oliva, A., Torralba, A., "Modeling the shape of the scene: A holistic representation of the spatial envelope," International Journal of Computer Vision, 42(3):145–175, 2001.

[254] Oliva, A., Torralba, A., "Building the gist of a scene: The role of global image features in recognition," in *Visual Perception, Progress in Brain Research*, 155:23–36, Elsevier, 2006.

[255] Omer, A.M.M., Ghorbani, R., Hun-ok Lim, Takanishi, A., "Semi-passive dynamic walking for biped walking robot using controllable joint stiffness based on dynamic simulation," *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Singapore, 2009.

[256] Pell, B., Bernard, D., Chien, S., Gat, E., Muscettola, N., Nayak, P., Wagner, M., Williams, B., "An autonomous spacecraft agent prototype," *Autonomous Robots* 5: 1–27, 1998.

[257] Pavlidis, T., Horowitz, S. L. "Segmentation of plane curves," *IEEE Transactions on Computers* C-23(8): 860–870, 1974.

[258] Pentland, A.P., "A new sense for depth of field," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI),* 9: 523–531, 1987.

[259] Philippsen, R., Siegwart, R., "Smooth and efficient obstacle avoidance for a tour guide robot," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2003)*, Taipei, Taiwan, 2003.

[260] Pivtoraiko, M., Knepper, R., A., Kelly, A. "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics* 26, no. 1: 308–333, 2009.

[261] Pfister, S. T., Roumeliotis, S. I., Burdick, J. W. "Weighted line fitting algorithms for mobile robot map building and efficient data representation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2003.

[262] Pratt, J., Pratt, G., "Intuitive control of a planar bipedal walking robot," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '98)*, Leuven, Belgium, May 1998.

[263] Rao, C.R."Information and accuracy obtainable in estimation of statistical parameters," *Bulletin of the Calcutta Mathematical Society* 37: 81–91, 1945.

[264] Raibert, M. H., Brown, H. B., Jr., Chepponis, M., "Experiments in balance with a 3D one-legged hopping machine," *International Journal of Robotics Research*, 3: 75–92, 1984.

[265] Remy, C., Buffinton, K., Siegwart, R., "Stability analysis of passive dynamic walking of quadrupeds," *International Journal of Robotics Research*, 2009.

[266] Ringrose, R., "Self-stabilizing running," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '97)*, Albuquerque, NM, April 1997.

[267] Rosten, E., Drummond, T., "Fusing points and lines for high performance tracking," in *Proceedings of the International Conference on Computer Vision*, 1508–1511, 2005.

[268] Rosten, E., Drummond, T., "Machine learning for high-speed corner detection," in *Proceedings of the European Conference on Computer Vision*, 430-443, 2006.

[269] Rowe, A., Rosenberg, C., Nourbakhsh, I., "A simple low cost color vision system," in *Proceedings of Tech Sketches for CVPR 2001*, Kuaii, Hawaii, December 2001.

[270] Rubner, Y., Tomasi, C., Guibas, L., "The earth mover's distance as a metric for image retrieval," *STAN-CS-TN-98-86, Stanford University*, 1998.

[271] Rufli, M., Ferguson, D., Siegwart, R., "Smooth path planning in constrained environments," *Proceedings of the IEEE International Conference on Robotics and Automation* (ICRA), 2009.

[272] Rufli, M., Siegwart, R., "On the application of the D* search algorithm to time based planning on lattice graphs," *Proceedings of the European Conference on Mobile Robots* (ECMR), 2009.

[273] Scaramuzza, D., "Omnidirectional vision: from calibration to robot motion estimation,", *PhD thesis n. 17635, ETH Zurich*, February 2008.

[274] Scaramuzza, D., Martinelli, A., Siegwart, R., "A flexible technique for accurate omnidirectional camera calibration and structure from motion," *IEEE International Conference on Computer Vision Systems (ICVS 2006)*, New York, January 2006.

[275] Scaramuzza, D., Martinelli, A. Siegwart, R., "A toolbox for easily calibrating omnidirectional cameras," *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS 2006), Beijing, China, October 2006.

[276] Scaramuzza, D., Fraundorfer, F., Pollefeys, M., "Closing the loop in appearance-guided omnidirectional visual odometry by using vocabulary trees," *Robotics and Autonomous System Journal (Elsevier)*, 2010.

[277] Scaramuzza, D., Fraundorfer, F., Pollefeys, M., and Siegwart, R., "Absolute scale in structure from motion from a single vehicle mounted camera by exploiting nonholonomic constraints," *IEEE International Conference on Computer Vision* (ICCV 2009), Kyoto, October, 2009.

[278] Scaramuzza, D., Fraundorfer, F., and Siegwart, R., Real-time monocular visual odometry for on-road vehicles with 1-point RANSAC, *IEEE International Conference on Robotics and Automation* (ICRA 2009), Kobe, Japan, May 2009.

[279] Scaramuzza, D., Siegwart, R., "Appearance guided monocular omnidirectional visual odometry for outdoor ground vehicles," *IEEE Transactions on Robotics* 24, no. 5, October 2008.

[280] Schlegel, C., "Fast local obstacle under kinematic and dynamic constraints," in *Proceedings of the IEEE International Conference on Intelligent Robot and Systems (IROS 98)*, Victoria, Canada 1998.

[281] Schultz, A., Adams, W., "Continuous localization using evidence grids," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'98)*, May 1998.

[282] Schweitzer, G., Werder, M., "ROBOTRAC – a mobile manipulator platform for rough terrain," in *Proceedings of the International Symposium on Advanced Robot Technology (ISART),* Tokyo, Japan, March, 1991.

[283] Shi, J., Malik, J., "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 82: 888–905, 2000.

[284] Shi, J., Tomasi, C., "Good features to track," *IEEE Conference on Computer Vision and Pattern Recognition*, 1994.

[285] Schmid, C., Mohr, R., Bauckhage, C., "Evaluation of interest point detectors," *International Journal of Computer Vision* 37, no. 2: 151–172, 2000.

[286] Se, S., Barfoot, T., Jasiobedzki, P., "Visual motion estimation and terrain modeling for planetary rovers," *Proceedings of the International Symposium on Artificial Intelligence for Robotics and Automation in Space*, 2005.

[287] Siadat, A., Kaske, A., Klausmann, S., Dufaut, M., Husson, R. "An optimized segmentation method for a 2D laser-scanner applied to mobile robot navigation," *Proceedings of the 3rd IFAC Symposium on Intelligent Components and Instruments for Control Applications*, 1997.

[288] Siegwart R., Arras, K., Bouabdallah, S., Burnier, D., Froidevaux, G., Greppin, X., Jensen, B., Lorotte, A., Mayor, L., Meisser, M., Philippsen, R., Piguet, R., Ramel, G., Terrien, G., Tomatis, N., "Robox at Expo.02: A large scale installation of personal robots," *Journal of Robotics and Autonomous Systems* 42: 203–222, 2003.

[289] Siegwart, R., Lamon, P., Estier, T., Lauria, M, Piguet, R., "Innovative design for wheeled locomotion in rough terrain," *Journal of Robotics and Autonomous Systems* 40: 151–162, 2002.

[290] Simhon, S., Dudek, G., "A global topological map formed by local metric maps," *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98)*, Victoria, Canada, October 1998.

[291] Simmons, R., "The curvature velocity method for local obstacle avoidance," *Proceedings of the IEEE International Conference on Robotics and Automation*, Minneapolis, April 1996.

[292] Sinha, S. N., Frahm, J. M., Pollefeys, M., Genc, Y., "GPU video feature tracking and matching," *in EDGE, Workshop on Edge Computing Using New Commodity Architectures*, 2006.

[293] Sivic, J. and Zisserman, A., "Video Google: A text retrieval approach to object matching in videos," *Proceedings of the International Conference on Computer Vision*, 2003.

[294] Smith, R., Self, M., Cheeseman, P., "Estimating uncertain spatial relationships in robotics," *Autonomous Robot Vehicles,* I. J. Cox and G. T. Wilfong (editors), Springer-Verlag, 167–193, 1990.

[295] Smith, R.C. , Cheeseman, P., "On the representation and estimation of spatial uncertainty, *International Journal of Robotics Research* 5, no. 4: 56–68, 1986.

[296] Smith, S. M., Brady, J. M., "SUSAN - A new approach to low level image processing," *International Journal of Computer Vision* 23, no. 34: 45–78, 1997.

[297] Snavely, N., Seitz, S.M., Szeliski, R., "Photo Tourism: Exploring photo collections in 3D," *ACM Transactions on Graphics*, 25(3), August 2006.

[298] Snavely, N., Seitz, S.M., Szeliski, R., "Modeling the World from Internet Photo Collections," *International Journal of Computer Vision*, 2007

[299] Soatto, S., Brockett, R., "Optimal structure from motion: Local ambiguities and global estimates,", *International Conference on Computer Vision and Pattern Recognition*, 1998.

[300] Sordalen, O.J., Canudas de Wi,t C., "Exponential control law for a mobile robot: extension to path following," *IEEE Transactions on Robotics and Automation*, 9: 837–842, 1993.

[301] Sorg, H.W., "From serson to draper – two centuries of gyroscopic development," *Navigation* 23: 313–324, 1976.

[302] Steinmetz, B.M., Arbter, K., Brunner, B., Landzettel, K., "Autonomous vision navigation of the nanokhod rover," *Proceedings of i-SAIRAS 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2001.

[303] Stentz, A., "The focussed D* algorithm for real-time replanning," in *Proceedings of IJCAI-95*, August 1995.

[304] Stentz, A., "Optimal and efficient path planning for partially-known environments," *Proceedings of the International Conference on Robotics and Automation*, 1994.

[305] Stevens, B.S., Clavel, R., Rey, L., "The DELTA parallel structured robot, yet more performant through direct drive," *Proceedings of the 23rd International Symposium on Industrial Robots*, 1992.

[306] Takeda, H., Facchinetti, C., Latombe, J.C., "Planning the motions of a mobile robot in a sensory uncertainty field," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16: 1002–1017, 1994.

[307] Tardif, J., Pavlidis, Y., Daniilidis, K., "Monocular visual odometry in urban environments using an omnidirectional camera," *IEEE/RSJ International Confrence on Intelligent Robots and Systems*, 2008.

[308] Taylor, R., Probert, P., "Range finding and feature extraction by segmentation of images for mobile robot navigation," *Proceedings of the IEEE International Conference on Robotics and Automation*, ICRA, 1996.

[309] Thrun, S., Burgard, W., Fox, D., "A probabilistic approach to concurrent mapping and localization for mobile robots." *Autonomous Robots* 31: 1–25. 1998.

[310] Thrun, S., et al., "Minerva: A second generation museum tour-guide robot," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'99)*, Detroit, May 1999.

[311] Thrun, S., Fox, D., Burgard, W., Dellaert, F., "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence*, 128: 99–141, 2001.

[312] Thrun, S. "A probabilistic online mapping algorithm for teams of mobile robots," *International Journal of Robotics Research* 20, no. 5: 335–363, 2001.

[313] Thrun, S. "Simultaneous localization and mapping," *Springer Tracts in Advanced Robotics* 38, no. 5: 13–41, 2008.

[314] Thrun, S., Gutmann, J.-S., Fox, D., Burgard, W., Kuipers, B., "Integrating topological and metric maps for mobile robot navigation: A statistical approach," *Proceedings of the National Conference on Artificial Intelligence (AAAI)*,1998.

[315] Thrun, S., Thayer, S., Whittaker, W., Baker, C., Burgard, W., Ferguson, D., Hähnel, D., Montemerlo, M., Morris, A., Omohundro, Z., Reverte, C., Whittaker, W. "Autonomous exploration and mapping of abandoned mines," *IEEE Robotics and Automation Mag*azine 11, no. 4: 79–91, 2004.

[316] Tomasi, C., Shi, J., "Image deformations are better than optical flow," *Mathematical and Computer Modelling* 24: 165–175, 1996.

[317] Tomatis, N., Nourbakhsh, I., Siegwart, R., "Hybrid simultaneous localization and map building: A natural integration of topological and metric," *Robotics and Autonomous Systems* 44, 3–14, 2003.

[318] Triggs, B., McLauchlan, P., Hartley, R., Fitzgibbon, A., "Bundle adjustment — a modern synthesis," *International Conference on Computer Vision*, 1999.

[319] Tsai, R. "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," *IEEE Journal of Robotics and Automation* 3, no. 4: 323–344, August 1987.

[320] Tuytelaars, T., Mikolajczyk, K., "Local invariant feature detectors: a survey," *Source, Foundations and Trends in Computer Graphics and Vision* 3 , no. 3, 2007.

[321] Tzafestas, C.S., Tzafestas, S.G., "Recent algorithms for fuzzy and neurofuzzy path planning and navigation of autonomous mobile robots," *Systems-Science* 25: 25–39, 1999.

[322] Ulrich, I., Borenstein, J., "VFH*: Local obstacle avoidance with look-ahead verification," *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, May 2000.

[323] Ulrich, I., Borenstein, J., "VFH+: Reliable obstacle avoidance for fast mobile robots," *Proceedings of the International Conference on Robotics and Automation (ICRA'98)*, Leuven, Belgium, May 1998.

[324] Ulrich, I., Nourbakhsh, I., "Appearance obstacle detection with monocular color vision," *the Proceedings of the AAAI National Conference on Artificial Intelligence*. Austin, TX. August 2000.

[325] Ulrich, I., Nourbakhsh, I., "Appearance-based place recognition for topological localization," *Proceedings of t he IEEE International Conference on Robotics and Automation*, San Francisco, 1023–1029, April 2000.

[326] Vanualailai, J., Nakagiri, S., Ha, J-H., "Collision avoidance in a two-point system via Liapunov's second method," *Mathematics and Simulation* 39: 125–141, 1995.

[327] Van Winnendael, M., Visenti G., Bertrand, R., Rieder, R., "Nanokhod microrover heading towards Mars," *Proceedings of the Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space* (ESA SP-440), Noordwijk, Netherlands, 1999.

[328] Vandorpe, J., Brussel, H. V., Xu, H. "Exact dynamic map building for a mobile robot using geometrical primitives produced by a 2D range finder," *Proceedings of the IEEE International Conference on Robotics and Automation*, ICRA, 901–908, 1996.

[329] Weiss, G., Wetzler, C., Puttkamer, E., "Keeping track of position and orientation of moving indoor systems by correlation of range-finder scans," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, Munich, September 1994.

[330] Weingarten, J., Gruener, G. and Siegwart, R., "A state-of-the-art 3D sensor for robot navigation," *Proceedings of IROS*, Sendai, September 2004.

[331] Weingarten, J. and Siegwart, R., "3D SLAM using planar segments," Proceedings of IROS, Beijing, October 2006.

[332] Wullschleger, F.H., Arra,s K.O., Vestli, S.J., "A flexible exploration framework for map building," *Proceedings of the Third European Workshop on Advanced Mobile Robots (Eurobot 99)*, Zurich, September 1999.

[333] Yagi, Y., Kawato, S.,"Panorama scene analysis with conic projection," *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS), Workshop on Towards a New Frontier of Applications*, 1990.

[334] Yamauchi, B., Schultz, A., Adams, W., "Mobile robot exploration and map-building with continuous localization," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'98)*, Leuven, Belgium, May 1998.

[335] Ying, X., Hu, Z., "Can we consider central catadioptric cameras and fisheye cameras within a unified imaging model?," *European Conference on Computer Vision* (ECCV), Lecture Notes in Computer Science, Springer Verlag, May 2004.

[336] Zhang, L., Ghosh, B. K., "Line segment based map building and localization using 2D laser rangefinder," *Proceedings of the IEEE International Conference on Robotics and Automation*, 2000.

[337] Zhang, Z., "A flexible new technique for camera calibration," *Microsoft Research Technical Report 98-71*, December 1998
see also http://research.microsoft.com/~zhang.

Referenced Webpages

[338] Fisher, R.B. (editor), "CVonline: On-line Compendium of Computer Vision," Available at www.dai.ed.ac.uk/CVonline.

[339] The Intel Image Processing Library/Integrated Performance Primitives (Intel IPP): http://software.intel.com/en-us/intel-ipp.

[340] Source code release site: www.cs.cmu.edu/~jbruce/cmvision.

[341] Newton Labs website: www.newtonlabs.com.

[342] For probotics: http://www.personalrobots.com.

[343] OpenCV, the Open Source Computer Vision library: http://opencv.willowgarage.com/wiki.

[344] Passive walking: www-personal.umich.edu/~artkuo/Passive_Walk/passive_walking.html.

[345] Passive walking, the Cornell Ranger: http://ruina.tam.cornell.edu/research/topics/locomotion_and_robotics/ranger/ranger2008.php.

[346] Computer Vision industry: http://www.cs.ubc.ca/spider/lowe/vision.html.

[347] Camera Calibration Toolbox for Matlab: http://www.vision.caltech.edu/bouguetj/calib_doc.

[348] List of camera calibration softwares: http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/links.html.

[349] Omnidirectional camera calibration toolbox from Christopher Mei http://www.robots.ox.ac.uk/~cmei/Toolbox.html.

[350] Omnidirectional camera calibration toolbox from Joao Barreto http://www.isr.uc.pt/~jpbar/CatPack/pag1.htm.

[351] Omnidirectional camera calibration toolbox from Davide Scaramuzza: google "ocamcalib" or go to http://robotics.ethz.ch/~scaramuzza/Davide_Scaramuzza_files/Research/OcamCalib_Tutorial.htm.

[352] Open source software for SLAM and loop-closing: http://openslam.org.

[353] Open source software for multi-view structure from motion: http://phototour.cs.washington.edu/bundler

[354] Microsoft Photosynth: http://photosynth.net

[355] Photo Tourism: http://phototour.cs.washington.edu/

[356] Voodoo Camera Tracker: A tool for the integration of virtual and real scenes http://www.digilab.uni-hannover.de/docs/manual.html

[357] Augmented-reality toolkit (ARToolkit): http://www.hitl.washington.edu/artoolkit
[358] Parallel Tracking and Mapping (PTAM): http://www.robots.ox.ac.uk/~gk/PTAM

Index

Intelligent Robotics and Autonomous Agents
Edited by Ronald C. Arkin

Dorigo, Marco, and Marco Colombetti, *Robot Shaping: An Experiment in Behavior Engineering*

Arkin, Ronald C., *Behavior-Based Robotics*

Stone, Peter, *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*

Wooldridge, Michael, *Reasoning about Rational Agents*

Murphy, Robin R., *An Introduction to AI Robotics*

Mason, Matthew T., *Mechanics of Robotic Manipulation*

Kraus, Sarit, *Strategic Negotiation in Multiagent Environments*

Nolfi, Stefano, and Dario Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*

Siegwart, Roland, and Illah R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*

Breazeal, Cynthia L., *Designing Sociable Robots*

Bekey, George A., *Autonomous Robots: From Biological Inspiration to Implementation and Control*

Choset, Howie, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*

Thrun, Sebastian, Wolfram Burgard, and Dieter Fox, *Probabilistic Robotics*

Mataric, Maja J., *The Robotics Primer*

Wellman, Michael P., Amy Greenwald, and Peter Stone, *Autonomus Bidding Agents: Strategies and Lessons from Trading Agent Competition*

Floreano, Dario, and Claudio Mattiusi, *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*

Sterling, Leon S., and Kuldar Taveter, *The Art of Agent-Oriented Modeling*

Stoy, Kasper, David Brandt, and David J. Christensen, *An Introduction to Self-Reconfigurable Robots*

Siegwart, Roland, Illah R. Nourbakhsh, and Davide Scaramuzza, *Introduction to Autonomous Mobile Robots*, second edition