9

OCCUPANCY GRID MAPPING

9.1 INTRODUCTION

The previous two chapters discussed the application of probabilistic techniques to a low-dimensional perceptual problem, that of estimating a robot's pose. We assumed that the robot was given a map in advance. This assumption is legitimate in quite a few real-world applications, as maps are often available a priori or can be constructed by hand. Some application domains, however, do not provide the luxury of coming with an a priori map. Surprisingly enough, most buildings do not comply with the blueprints generated by their architects. And even if blueprints were accurate, they would not contain furniture and other items that, from a robot's perspective, determine the shape of the environment just as much as walls and doors. Being able to learn a map from scratch can greatly reduce the efforts involved in installing a mobile robot, and enable robots to adapt to changes without human supervision. In fact, mapping is one of the core competencies of truly autonomous robots.

Acquiring maps with mobile robots is a challenging problem for a number of reasons:

- The hypothesis space, that is the space of all possible maps, is huge. Since maps are defined over a continuous space, the space of all maps has infinitely many dimensions. Even under discrete approximations, such as the grid approximation which shall be used in this chapter, maps can easily be described 10^5 or more variables. The sheer size of this high-dimensional space makes it challenging to calculate full posteriors over maps; hence, the Bayes filtering approach that worked well for localization is inapplicable to the problem of learning maps, at least in its naive form discussed thus far.

- Learning maps is a "chicken-and-egg" problem, for which reason is often referred to as the *simultaneous localization and mapping* (SLAM) or *concurrent mapping and localization problem* problem. When the robot moves through its environment, it accumulates errors in odometry, making it gradually less certain as to where it is. Methods exist for determining the robot's pose when a map is available, as we have seen in the previous chapter. Likewise, constructing a map when the robot's poses are known is also relatively easy—a claim that will be substantiated by this chapter and subsequent chapters. In the absence of both an initial map and exact pose information, however, the robot has to do both: estimating the map and localizing itself relative to this map.

Of course, not all mapping problems are equally hard. The hardness of the mapping problem is the result of a collection of factors, the most important of which are:

- **Size.** The larger the environment relative to the robot's perceptual range, the more difficult it is to acquire a map.
- **Noise in perception and actuation.** If robot sensors and actuators were noise-free, mapping would be a simple problem. The larger the noise, the more difficult the problem.
- **Perceptual ambiguity.** The more frequently different places look alike, the more difficult it is to establish correspondence between different locations traversed at different points in time.
- **Cycles.** Cycles in the environment are particularly difficult to map. If a robot just goes up and down a corridor, it can correct odometry errors incrementally when coming back. Cycles make robots return via different paths, and when closing the cycle the accumulated odometric error can be huge!

To fully appreciate the difficulty of the mapping problem, consider Figure 9.1. Shown there is a data set, collected in a large indoor environment. Figure 9.1a was generated using the robot's raw odometry information. Each black dot in this figure corresponds to an obstacle detected by the robot's laser range finder. Figure 9.1b shows the result of applying mapping algorithms to this data, including the techniques described in this chapter. This example gives a good flavor of problem at stake.

In this chapter, we first study the mapping problem under the restrictive assumption that the robot poses are known. Put differently, we side-step the hardness of the SLAM problem by assuming some oracle informs us of the exact robot path during mapping. We will discuss a popular family of algorithms, collectively called *occupancy grids*.

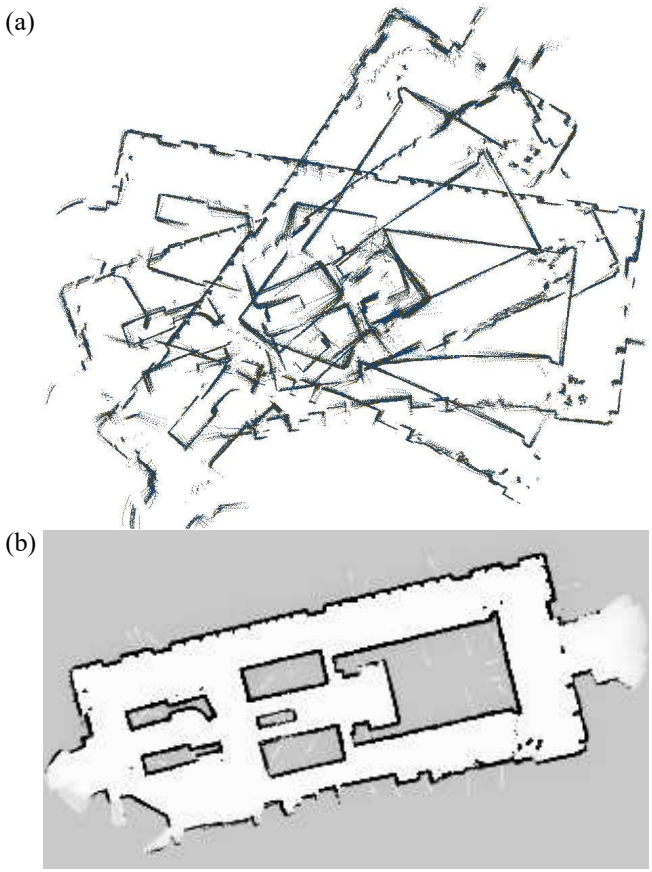

Figure 9.1 (a) Raw range data, position indexed by odometry. (b) Map

Occupancy grid maps address the problem of generating consistent maps from noisy and uncertain measurement data, under the assumption that the robot pose is known. The basic idea of the occupancy grids is to represent the map as a field of random variables, arranged in an evenly spaced grid. Each random variable is binary and corresponds to the occupancy of the location is covers. Occupancy grid mapping algorithms implement approximate posterior estimation for those random variables.

The reader may wonder about the significance of a mapping technique that requires exact pose information. After all, no robot's odometry is perfect! The main utility of the occupancy grid technique is in post-processing: Many of the SLAM techniques discussed in subsequent chapters do not generate maps fit for path planning and navi-

gation. Occupancy grid maps are often used after solving the SLAM problem by some other means, and taking the resulting path estimates for granted.

9.2 THE OCCUPANCY GRID MAPPING ALGORITHM

To gold standard of any occupancy grid mapping algorithm is to calculate the posterior over maps given the data

$$p(m \mid z_{1:t}, x_{1:t}) \tag{9.1}$$

As usual, m is the map, $z_{1:t}$ the set of all measurements up to time t, and $x_{1:t}$ is the path of the robot, that is, the sequence of all its poses. The controls $u_{1:t}$ play no role in occupancy grid maps, since the path is already known. Hence, they will be omitted throughout this chapter.

The types maps considered by occupancy grid maps are fine-grained grids defined over the continuous space of locations. By far the most common domain of occupancy grid maps are 2-D floorplan maps, which describe a 2-D slice of the 3-D world. 2-D maps are often sufficient, especially when a robot navigates on a flat surface and the sensors are mounted so that they capture only a slice of the world. Occupancy grid techniques generalize to 3-D representations, but at significant computational expenses.

Let $\mathbf{m}_i$ denote the grid cell with index i. An occupancy grid map partitions the space into finitely many grid cells:

$$m = \sum_i \mathbf{m}_i \tag{9.2}$$

Each $\mathbf{m}_i$ has attached to it a binary occupancy value, which specifies whether a cell is occupied or free. We will write "1" for occupied and "0" for free. The notation $p(\mathbf{m}_i = 1)$ or $p(\mathbf{m}_i)$ will refer to a probability that a grid cell is occupied.

The problem with the posterior (9.1) is its dimensionality: the number of grid cells in maps like the one shown in Figure 9.1 are in the tens of thousands, hence the number of maps defined in this space is often in the excess of $2^{10,000}$. Calculating a posterior for each single such map is therefore intractable.

The standard occupancy grid approach breaks down the problem of estimating the map into a collection of separate problems, namely that of estimating

$$p(\mathbf{m}_i \mid z_{1:t}, x_{1:t}) \tag{9.3}$$

for all grid cell $\mathbf{m}_i$. Each of these estimation problems is now a binary problem with static state. This decomposition is convenient but not without problems. In particular, it does enable us to represent dependencies among neighboring cells; instead, the posterior over maps is approximated as the product of its marginals:

$$p(m \mid z_{1:t}, x_{1:t}) \quad = \quad p(\mathbf{m}_i \mid z_{1:t}, x_{1:t}) \tag{9.4}$$

We will return to this issue in Section 9.4 below, when we discuss more advanced mapping algorithms. For now, we will adopt this factorization for convenience.

Thanks to our factorization, the estimation of the occupancy probability for each grid cell is now a binary estimation problem with static state. A filter for this problem was already discussed in Chapter 4.1.4, with the corresponding algorithm depicted in Table 4.2 on page 75. The algorithm in Table 9.1 applies this filter to the occupancy grid mapping problem. As in the original filter, our occupancy grid mapping algorithm uses the log-odds representation of occupancy:

$$l_{t,i} \quad = \quad \log \frac{p(\mathbf{m}_i \mid z_{1:t}, x_{1:t})}{1 - p(\mathbf{m}_i \mid z_{1:t}, x_{1:t})} \tag{9.5}$$

This representation is already familiar from Chapter 4.1.4. The advantage of the log-odds over the probability representation is that we can avoid numerical instabilities for probabilities near zero or one. The probabilities are easily recovered from the log-odds ratio:

$$p(\mathbf{m}_i \mid z_{1:t}, x_{1:t}) \quad = \quad 1 - \frac{1}{1 + \exp\{l_{t,i}\}} \tag{9.6}$$

The algorithm **occupancy_grid_mapping** in Table 9.1 loops through all grid cells i, and updates those that fall into the sensor cone of the measurement z_t. For those where it does, it updates the occupancy value by virtue of the function **inverse_sensor_model**

1: **Algorithm occupancy_grid_mapping($\{l_{t-1,i}\}, x_t, z_t$):**
2: for all cells $\mathbf{m}_i$ do
3: if $\mathbf{m}_i$ in perceptual field of z_t then
4: $l_{t,i} = l_{t-1,i} + \mathbf{inverse\_sensor\_model}(\mathbf{m}_i, x_t, z_t) - l_0$
5: else
6: $l_{t,i} = l_{t-1,i}$
7: endif
8: endfor
9: return $\{l_{t,i}\}$

Table 9.1 The occupancy grid algorithm, a version of the binary Bayes fi lter in Table 4.2.

in line 4 of the algorithm. Otherwise, the occupancy value remains unchanged, as indicated in line 6. The constant l_0 is the prior of occupancy represented as a log-odds ratio:

$$l_0 \quad = \quad \log \frac{p(\mathbf{m}_i = 1)}{p(\mathbf{m}_i = 0)} \quad = \quad \log \frac{p(\mathbf{m}_i)}{1 - p(\mathbf{m}_i)} \tag{9.7}$$

The function **inverse_sensor_model** implements the inverse measurement model $p(\mathbf{m}_i \mid z_t, x_t)$ in its log-odds form:

$$\mathbf{inverse\_sensor\_model}(\mathbf{m}_i, x_t, z_t) \quad = \quad p(\mathbf{m}_i \mid z_t, x_t) \tag{9.8}$$

A somewhat simplistic example of such a function for range finders is given in Table 9.2 and illustrated in Figure 9.7a&b. This model assigns to all cells within the sensor cone whose range is close to the measured range an occupancy value of l_{occ}. In Table 9.2, the width of this region is controlled by the parameter α, and the opening angle of the beam is given by β.

The algorithm **occupancy_grid_mapping** calculates the inverse model by first determining the beam index k and the range r for the center-of-mass of the cell $\mathbf{m}_i$. This calculation is carried out in lines 2 through 5 in Table 9.2. As usual, we assume that the robot pose is given by $x_t = (x \;\; y \;\; \theta)^T$. In line 7, it returns the prior for occupancy in log-odds form whenever the cell is outside the measurement range of this sensor beam, or if it lies more than $\alpha/2$ behind the detected range z_t^k. In line 9, it returns

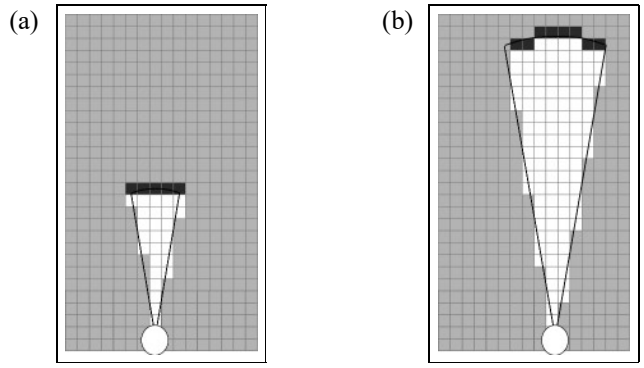

Figure 9.2 Two examples of our inverse measurement model **inverse_range_sensor_model** for two different measurement ranges. The darkness of each grid cell corresponds to the likelihood of occupancy.

$l_{\text{occ}} > l_0$ is the range of the cell is within $\pm\alpha/2$ of the detected range z_t^k. It returns $l_{\text{free}} < l_0$ if the range to the cell is shorter than the measured range by more than $\alpha/2$. The left and center panel of Figure 9.7 illustrates this calculation for a sonar beam with a 15° opening angle.

Figures **??** shows an example map next to a blueprint of a large open exhibit hall, and relates it to the occupancy map acquired by a robot. The map was generated using laser range data gathered in a few minutes' time. The grey-level in the occupancy map indicates the posterior of occupancy over an evenly spaced grid: The darker a grid cell, the more likely it is occupied. While occupancy maps are inherently probabilistic, they tend to quickly converge to estimates that are close to the two extreme posteriors, 1 and 0. In comparison between the learned map and the blueprint, the occupancy grid map shows all major structural elements, and obstacles as they were visible at the height of the laser. Close inspection alleviates discrepancies between the blueprint and the actual environment configuration.

Figure 9.4 compares a raw dataset with the occupancy grid maps generated from this data. The data in Panel (a) was preprocessed by a SLAM algorithm, so that the poses align. Some of the data is corrupted by the presence of people; the occupancy grid map filters out people quite nicely. This makes occupancy grid maps much better suited for robot navigation than sets of scan endpoint data: A planner fed the raw sensor endpoints would have a hard time finding a path through such scattered obstacles, even if the evidence that the corresponding cell is free outweigh that of occupancy.

(a)

(b)

Figure 9.3 (a) Occupancy grid map and (b) architectural blue-print of a large open exhibit space. Notice that the blue-print is inaccurate in certain places.

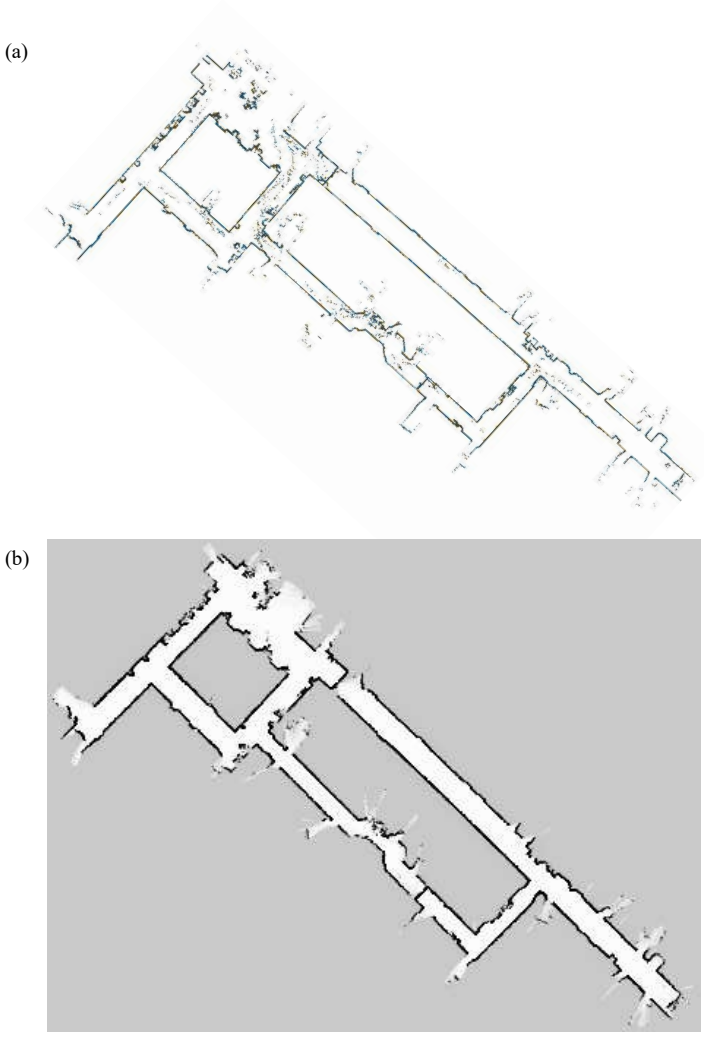

Figure 9.4 (a) Raw laser range data with correct(ed) pose information. Each dot corresponds to a detection of an obstacle. Most obstacles are static (walls etc.), but some were people that walked in the vicinity of the robot. (b) Occupancy grid map built from the data. The grey-scale indicates the posterior probability: Black corresponds to occupied with high certainty, and white to free with high certainty. The grey background color represents the prior. Figure (a) has been generated by Steffen Gutmann.

1: **Algorithm inverse_range_sensor_model**(i, x_t, z_t)**:**
2: Let x_i, y_i be the center-of-mass of $\mathbf{m}_i$
3: $r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$
4: $\phi = \text{atan2}(y_i - y, x_i - x) - \theta$
5: $k = \text{argmin}_j \, |\phi - \theta_{j,\text{sens}}|$
6: if $r > \min(z_{\max}, z_t^k + \alpha/2)$ or $|\phi - \theta_{k,\text{sens}}| > \beta/2$ then
7: return l_0
8: if $z_t^k < z_{\max}$ and $|r - z_{\max}| < \alpha/2$
9: return l_{occ}
10: if $r \leq z_t^k$
11: return l_{free}
12: endif

Table 9.2 A simple inverse measurement model for robots equipped with range finders Here α is the thickness of obstacles, and β the width of a sensor beam. The values l_{occ} and l_{free} in lines 9 and 11 denote the amount of evidence a reading carries for the two difference cases.

We note that our algorithm makes occupancy decisions exclusively based on sensor measurements. An alternative source of information is the space claimed by the robot itself: When the robot's pose is x_t, the region surrounding x_t must be navigable. Our inverse measurement algorithm in Table 9.2 can easily be modified to incorporate this information, by returning a large negative number for all grid cells occupied by a robot when at x_t. In practice, it is a good idea to incorporate the robot's volume when generating maps, especially if the environment is populated during mapping.

9.2.1 Multi-Sensor Fusion

Robots are often equipped with more than one type of sensor. Hence, a natural objective is to integrate information from more than one sensors into a single map. This question as to how to best integrate data from multiple sensors is particularly interesting if the sensors have different characteristics. For example, Figure 9.5 shows occupancy maps built with a stereo vision system, in which disparities are projected onto the plane and convolved with a Gaussian. Clearly, the characteristics of stereo are different from that of a sonar-based range finder, in that that are sensitive to different types obstacles.

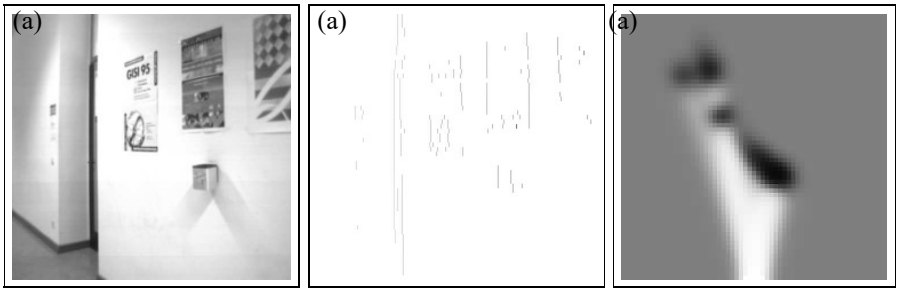

Figure 9.5 Estimation of occupancy maps using stereo vision: (a) camera image, (b) sparse disparity map, (c) occupancy map by projecting the disparity image onto the 2-D plane and convolving the result with a Gaussian.

There are two basic approaches for fusing data from multiple sensors is to use Bayes filters for sensor integration. We can execute the algorithm **occupancy_grid_mapping** in Table 9.1 with different sensor modalities, replacing the function **inverse_sensor_model** accordingly. However, such an approach has a clear drawback. If different sensors detect different types of obstacles, the result of Bayes filtering is ill-defined. Consider, for example, consider an obstacle that can be recognized by sensor A but not by sensor B. Then these two sensors will generate conflicting information, and the resulting map will depend on the amount of evidence brought by every sensor system. This is generally undesirable, since whether or not a cell is considered occupied depends on the relative frequency at which different sensors are polled.

The second, more appropriate approach to integrating information from multiple sensors is to build separate maps for each sensor types, and integrate them using the most conservative estimate. Let $m^k = \{\mathbf{m}_i^k\}$ denote the map built by the k-th sensor type. Then the combined map is given by

$$\mathbf{m}_i \quad = \quad \max_k \mathbf{m}_i^k \tag{9.9}$$

This map is the most pessimistic map given its components: If any of the sensor-specific map shows that a grid cell is occupied, so will the combined map. While this combined estimator is biased in factor of occupied maps, it is more appropriate than the Bayes filter approach when sensors with different characteristics are fused.

9.3 LEARNING INVERSE MEASUREMENT MODELS

9.3.1 Inverting the Measurement Model

The occupancy grid mapping algorithm requires a marginalized *inverse* measurement model, $p(\mathbf{m}_i \mid x, z)$. This probability is called "inverse" since it reasons from effects to causes: it provides information about the world conditioned on a measurement caused by this world. It is marginalized for the i-th grid cell; a full inverse would be of the type $p(m \mid x, z)$. In our exposition of the basic algorithm, we already provided an ad hoc procedure in Table 9.2 for implementing such an inverse model. This raises the question as to whether we can obtain an inverse model in a more principled manner, starting at the conventional measurement model.

The answer is positive but less straightforward than one might assume at first glance. Bayes rule suggests

$$p(m \mid x, z) = \frac{p(z \mid x, m)\, p(m \mid x)}{p(z \mid x)} \tag{9.10}$$

$$= \eta\, p(z \mid x, m)\, p(m) \tag{9.11}$$

Here we silently assume $p(m \mid x) = p(m)$, that is, the pose of the robot tells us nothing about the map—an assumption that we will adopt for sheer convenience. If our goal was to calculate the inverse model for the entire map at-a-time, we would now be done. However, our occupancy grid mapping algorithm approximates the posterior over maps by its marginals, one for each grid cell $\mathbf{m}_i$. The inverse model for the i-th grid cell is obtained by selecting the marginal for the i-th grid cell:

$$p(\mathbf{m}_i \mid x, z) = \eta \int_{m:m(i)=\mathbf{m}_i} p(z \mid x, m)\, p(m)\, dm \tag{9.12}$$

This expression integrates over all maps m for which the occupancy value of grid cell i equals $\mathbf{m}_i$. Clearly, this integral cannot be computed, since the space of all maps is too large. We will now describe an algorithm for approximating this expression. The algorithm involves generating samples from the measurement model, and approximating the inverse using a function approximator (or supervised learning algorithm), such as a polynomial, logistic regression, or a neural network.

9.3.2 Sampling from the Forward Model

The basic idea is simple and quite universal: If we can generate random triplets of poses $x_t^{[k]}$, measurements $z_t^{[k]}$ and map occupancy values $\mathbf{m}_i^{[k]}$ for any grid cell $\mathbf{m}_i$, we can learn a function that accepts a pose x and measurement z as an input, and outputs the probability of occupancy for $\mathbf{m}_i$.

A sample of the form $(x_t^{[k]} \;\; z_t^{[k]} \;\; \mathbf{m}_i^{[k]})$ can be generated by the following procedure.

1. Sample a random map $m^{[k]} \sim p(m)$. For example, one might already have a database of maps that represents $p(m)$ and randomly draws a map from the database.
2. Sample a pose $x_t^{[k]}$ inside the map. One may safely assume that poses are uniformly distributed.
3. Sample an measurement $z_t^{[k]} \sim p(z \mid x_t^{[k]}, m^{[k]})$. This sampling step is reminiscent of a robot simulator which stochastically simulates a sensor measurement.
4. Extract the desired "true" occupancy value $\mathbf{m}_i$ for the target grid cell from the map m.

The result is a sampled pose $x_t^{[k]}$, a measurement $z_t^{[k]}$, and the occupancy value the the grid cell $\mathbf{m}_i$. Repeated application of this sampling step yields a data set

$$\begin{array}{cccc} x_t^{[1]} & z_t^{[1]} & \longrightarrow & \mathrm{occ}(\mathbf{m}_i)^{[1]} \\ x_t^{[2]} & z_t^{[2]} & \longrightarrow & \mathrm{occ}(\mathbf{m}_i)^{[2]} \\ x_t^{[3]} & z_t^{[3]} & \longrightarrow & \mathrm{occ}(\mathbf{m}_i)^{[3]} \\ \vdots & \vdots & & \vdots \end{array} \tag{9.13}$$

These triplets may serve as training examples for function approximator, to approximate the desired conditional probability $p(\mathbf{m}_i \mid z, x)$. Here the measurements z and the pose x are input variables, and the occupancy value $\mathrm{occ}(\mathbf{m}_i)$ is a target for the output of the function approximator.

This approach is somewhat inefficient, since it fails to exploit a number of properties that we know to be the case for the inverse sensor model.

- Measurements should carry no information about grid cells far outside their perceptual range. This observation has two implications: First, we can focus our

sample generation process on triplets where the cell $\mathbf{m}_i$ is indeed inside the measurement cone. And second, when making a prediction for this cell, we only have to include a subset of the data in a measurement z (e.g., nearby beams) as input to the function approximator.

- The characteristics of a sensor are invariant with respect to the absolute coordinates of the robot or the grid cell when taking a measurement. Only the relative coordinates matter. If we denote the robot pose by $x_t = (x \;\; y \;\; \theta)^T$ and the coordinates of the grid cell by $\mathbf{m}_i = (x_{\mathbf{m}_i} \;\; y_{\mathbf{m}_i})^T$, the coordinates of the grid cell are mapped into the robot's local reference frame via the following translation and rotation:

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_{\mathbf{m}_i} - x \\ y_{\mathbf{m}_i} - y \end{pmatrix} \tag{9.14}$$

  In robots with circular array of range finders, it makes sense to encode the relative location of a grid cell using the familiar polar coordinates (range and angle).

- Nearby grid cells should have a similar interpretation under the inverse sensor model. This smoothness suggest that it may be beneficial to learn a single function in which the coordinates of the grid cell function as an input, rather than learning a separate function for each grid cell.

- If the robot possesses functionally identical sensors, the inverse sensor model should be interchangeable for different sensors. For robots equipped with a circular array of range sensors, any of the resulting sensor beam is characterized by the same inverse sensor model.

The most basic way to enforce these invariances is to constrain the function approximator by choosing appropriate input variables. A good choice is to use relative pose information, so that the function approximator cannot base its decision on absolute coordinates. It is also a good idea to omit sensor measurements known to be irrelevant to occupancy predictions, and to confine the prediction to grid cells inside the perceptual field of a sensor. By exploiting these invariances, the training set size can be reduced significantly.

9.3.3 The Error Function

To train the function approximator, we need an approximate error function. A popular example are artificial neural networks trained with the Back-propagation algorithm. Back-propagation trains neural networks by gradient descent in parameter

space. Given an error function that measures the "mismatch" between the network's actual and desired output, Back-propagation calculates the first derivative of the target function and the parameters of the neural network, and then adapts the parameters in opposite direction of the gradient so as to diminish the mismatch. This raises the question as to what error function to use.

A common approach is to train the function approximator so as to maximize the log-likelihood of the training data. More specifically we are given a training set of the form

$$\begin{array}{ccc} \mathrm{input}^{[1]} & \longrightarrow & \mathrm{occ}(\mathbf{m}_i)^{[1]} \\ \mathrm{input}^{[2]} & \longrightarrow & \mathrm{occ}(\mathbf{m}_i)^{[2]} \\ \mathrm{input}^{[3]} & \longrightarrow & \mathrm{occ}(\mathbf{m}_i)^{[3]} \\ \vdots & & \vdots \end{array} \tag{9.15}$$

$occ(\mathbf{m}_i)^{[k]}$ is the i-th sample of the desired conditional probability, and $\mathrm{input}^{[k]}$ is the corresponding input to the function approximator. Clearly, the exact form of the input may vary as a result of the encoding known invariances, but the exact nature of this vector will plays no role in the form of the error function.

Let us denote the parameters of the function approximator by W. Assuming that each individual item in the training data has been generated independently, the likelihood of the training data is now

$$\prod_i p(\mathbf{m}_i^{[k]} \mid \mathrm{input}^{[k]}, W) \tag{9.16}$$

and its logarithm is

$$J(W) \quad = \quad \sum_i \log p(\mathbf{m}_i^{[k]} \mid \mathrm{input}^{[k]}, W) \tag{9.17}$$

Here J defines the function we seek to maximize during training.

Let us denote the function approximator by $f(\mathrm{input}^{[k]}, W)$. The output of this function is a value in the interval $[0; 1]$. After training, we want the function approximator

output the probability of occupancy, that is:

$$p(\mathbf{m}_i^{[k]} \mid \text{input}^{[k]}, W) \quad = \quad \begin{cases} f(\text{input}^{[k]}, W) & \text{if } \mathbf{m}_i^{[k]} = 1 \\ 1 - f(\text{input}^{[k]}, W) & \text{if } \mathbf{m}_i^{[k]} = 0 \end{cases} \tag{9.18}$$

Thus, we seek an error function that adjusts W so as to minimize the deviation of this predicted probability and the one communicated by the training example. To find such an error function, we re-write (9.18) as follows:

$$p(\mathbf{m}_i^{[k]} \mid \text{input}^{[k]}, W) \quad = \quad f(\text{input}^{[k]}, W)^{\mathbf{m}_i^{[k]}} \; (1 - f(\text{input}^{[k]}, W))^{1-\mathbf{m}_i^{[k]}} \tag{9.19}$$

It is easy to see that this product and Expression (9.18) are identical. In the product, one of the terms is always 1, since its exponent is zero. Substituting the product into (9.20) and multiplying the result by minus one gives us the following function:

$$\begin{aligned} J(W) \quad &= \quad -\sum_i \log \left[f(\text{input}^{[k]}, W)^{\mathbf{m}_i^{[k]}} \; (1 - f(\text{input}^{[k]}, W))^{1-\mathbf{m}_i^{[k]}} \right] \\ &= \quad -\sum_i \mathbf{m}_i^{[k]} \log f(\text{input}^{[k]}, W) + (1 - \mathbf{m}_i^{[k]}) \log(1 - f(\text{input}^{[k]}, W)) \end{aligned} \tag{9.20}$$

$J(W)$ is the error function to minimize when training the function approximator. It is easily folded into any function approximator that uses gradient descent.

9.3.4 Further Considerations

Figure 9.6 shows the result of an artificial neural network trained to mimic the inverse sensor model. The robot in this example is equipped with a circular array of sonar range sensors mounted at approximate table height. The input to the network are relative coordinates in polar coordinates, and the set of five adjacent range measurements. The output is a probability of occupancy: the darker a cell, the more likely it is occupied. As this example illustrates, the approach correctly learns to distinguish freespace from occupied space. The gray-ish area behind obstacles matches the prior probability of occupancy, which leads to no change when used in the occupancy grid mapping algorithm. Figure 9.6b contains a faulty short reading on the bottom left. Here a single reading seems to be insufficient to predict an obstacle with high probability.

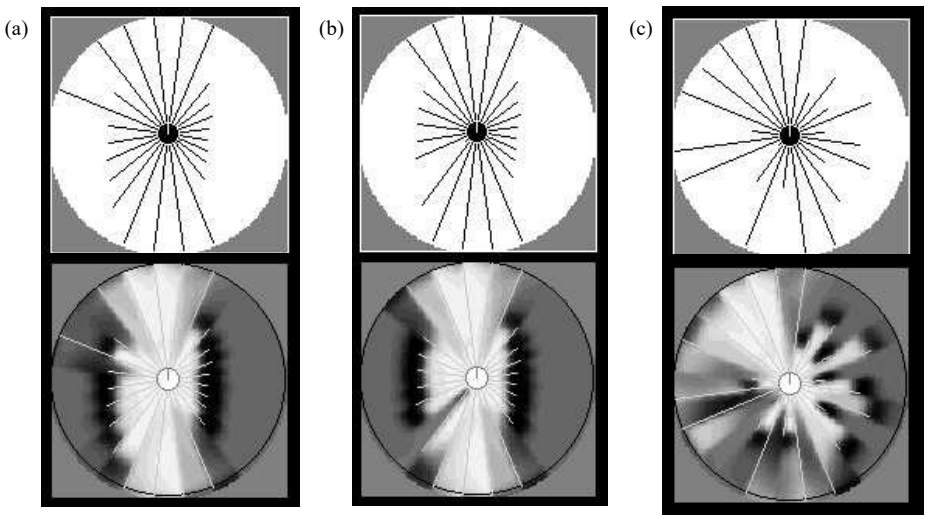

Figure 9.6 Sensor interpretation: Three sample sonar scans (top row) and local occupancy maps (bottom row), as generated by the neural network. Bright regions indicate free-space, and dark regions indicate walls and obstacles (enlarged by a robot diameter).

We note that there exists a number of ways to train a function approximator using actual data collected by a robot, instead the simulated data from the forward model. In general, this is the most accurate data one can use for learning, since the measurement model is necessarily just an approximation. One such way involves a robot operating in a known environment with a known map. With Markov localization, we can localize the robot, and then use its actual recorded measurements and the known map occupancy to assemble training examples. It is even possible to start with an approximate map, use the learned sensor model to generate a better map, and from that map use the procedure just outlined to improve the inverse measurement model. As this book is being written, the issue of learning inverse sensor models from data remains relatively unexplored.

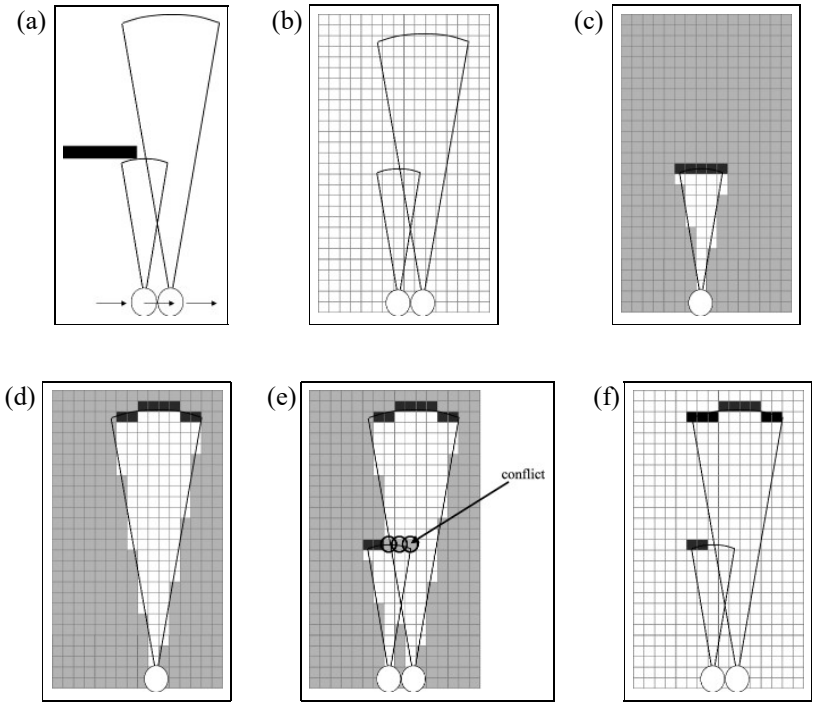

Figure 9.7 The problem with the standard occupancy grid mapping algorithm in Section 9.2: For the environment shown in Figure (a), a passing robot might receive the (noise-free) measurement shown in (b). The factorial approach maps these beams into probabilistic maps separately for each grid cell and each beam, as shown in (c) and (d). Combining both interpretations yields the map shown in (e). Obviously, there is a conflict in the overlap region, indicated by the circles in (e). The interesting insight is: There exist maps, such as the one in diagram (f), which perfectly explain the sensor measurement without any such conflict. For a sensor reading to be explained, it suffices to assume an obstacle *somewhere* in the cone of a measurement, and not everywhere.

9.4 MAXIMUM A POSTERIOR OCCUPANCY MAPPING

9.4.1 The Case for Maintaining Dependencies

In the remainder of this chapter, we will return to one of the very basic assumptions of the occupancy grid mapping algorithm. In Section 9.2, we assumed that we can safely

1: **Algorithm MAP_occupancy_grid_mapping($x_{1:t}, z_{1:t}$):**
2: set $m = \{0\}$
3: repeat until convergence
4: for all cells $\mathbf{m}_i$ do
5: $$m_i = \operatorname*{argmax}_{k=0,1} (l_0)^k + \sum_t \log \mathbf{measurement\_model}(z_t, x_t, m \text{ with } \mathbf{m}_i = k)$$
6: endfor
7: endrepeat
8: return m

Table 9.3 The maximum a posteriori occupancy grid algorithm, which uses conventional measurement models instead of inverse models.

decompose the map inference problem defined over high-dimensional space of all maps, into a collection of single-cell mapping problems. This assumption culminated into the factorization in (9.4). This raises the question as to how faithful we should be in the result of any algorithm that relies on such a strong decomposition.

Figure 9.7 illustrates a problem that arises directly as a result of this factorization. Shown there is a situation in which the the robot facing a wall receives two noise-free sonar range measurements. Because the factored approach predicts an object along the entire arc at the measured range, the occupancy values of all grid cells along this arc are increased. When combining the two different measurements shown in Figure 9.7c&d, a conflict is created, as shown in Figure 9.7e. The standard occupancy grid mapping algorithm "resolves" this conflict by summing up positive and negative evidence for occupancy; however, the result will reflect the relative frequencies of the two types of measurements, which is undesirable.

However, there exist maps, such as the one in Figure 9.7f, which perfectly explains the sensor measurements without any such conflict. This is because for a sensor reading to be explained, it suffices to assume an obstacle *somewhere* in its measurement cone. Put differently, the fact that cones sweep over multiple grid cells induces important dependencies between neighboring grid cells. When decomposing the mapping into thousands of individual grid cell estimation problems, we lose the ability to consider these dependencies.

9.4.2 Occupancy Grid Mapping with Forward Models

These dependencies are incorporated by an algorithm that outputs the mode of the posterior, instead of the full posterior. The mode is defined as the maximum of the logarithm of the map posterior, which we already encountered in Equation (9.1):

$$m^* = \operatorname*{argmax}_m \log p(m \mid z_{1:t}, x_{1:t}) \tag{9.21}$$

The map posterior factors into a map prior and a measurement likelihood (c.f., Equation (9.11)):

$$\log p(m \mid z_{1:t}, x_{1:t}) = \log p(z_{1:t} \mid x_{1:t}, m) + \log p(m) \tag{9.22}$$

The log-likelihood $\log p(z_{1:t} \mid x_{1:t}, m)$ decomposes into a sum all individual measurement log-likelihoods $\log p(z_t \mid x_t, m)$. Further, the log-prior $\log p(m)$ is simply a sum of the type

$$\begin{aligned} \log p(m) &= \sum_i [\log p(\mathbf{m}_i)]^{\mathbf{m}_i} + [\log(1 - p(\mathbf{m}_i))]^{1-\mathbf{m}_i} \\ &= M \log(1 - p(\mathbf{m}_i)) + \sum_i (l_0)^{\mathbf{m}_i} \end{aligned} \tag{9.23}$$

where M denotes the number of grid cells, and l_0 is adopted from (9.7). The term $M \log(1 - p(\mathbf{m}_i))$ is obviously independent of the map. Hence it suffices to optimize the remaining expression and the data log-likelihood:

$$m^* = \operatorname*{argmax}_m \sum_t \log p(z_t \mid x_t, m) + \sum_i (l_0)^{\mathbf{m}_i} \tag{9.24}$$

A hill-climbing algorithm for maximizing this log-probability is provided in Table 9.3. This algorithm starts with the all-free map (line 2). It "flips" the occupancy value of a grid cell when such a flip increases the likelihood of the data (lines 4-6). For this algorithm it is essential that the prior of occupancy $p(\mathbf{m}_i)$ is not too close to 1; otherwise it might return an all-occupied map. As any hill climbing algorithm, this approach is only guaranteed to find a local maximum. In practice, there are usually very few, if any, local maxima.

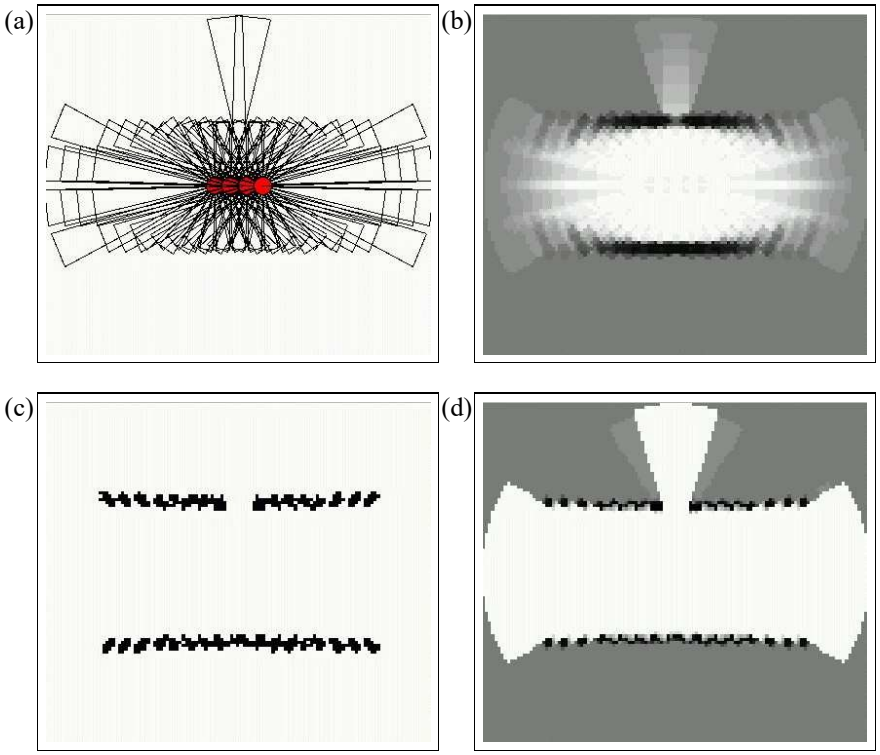

Figure 9.8 (a) sonar range measurements from a noise-free simulation; (b) Results of the standard occupancy mapper, lacking the open door. (c) A maximum a posterior map. (d) The residual uncertainty in this map, obtained by measuring the sensitivity of the map likelihood function with respect to individual grid cells..

Figure 9.8 illustrates the effect of the MAP occupancy grid algorithm. Figure 9.8a depicts a noise-free data set of a robot passing by an open door. Some of the sonar measurements detect the open door, while others are reflected at the door post. The standard occupancy mapping algorithm with inverse models fails to capture the opening, as shown in Figure 9.8b. The mode of the posterior is shown in Figure 9.8c. This map models the open door correctly, hence it is better suited for robot navigation than the standard occupancy grid map algorithm. Figure 9.8d shows the residual uncertainty of this map. This diagram is the result of a cell-wise sensitivity analysis: The magnitude by which flipping a grid cell decreases the log-likelihood function is illustrated by the grayness of a cell. This diagram, similar in appearance to the regular

occupancy grid map, suggests maximum uncertainty for grid cells behind obstacles. It lacks the vertical stripes found in Figure 9.8a.

There exists a number of limitations of the algorithm **MAP_occupancy_grid_mapping**, and it can be improved in multiple ways. The algorithm is a maximum a posterior approach, and as such returns no notion of uncertainty in the residual map. Our sensitivity analysis approximates this uncertainty, but this approximation is overconfident, since sensitivity analysis only inspects the mode locally. Further, the algorithm is a batch algorithm and cannot be executed incrementally. In fact, the MAP algorithm requires that all data is kept in memory. At the computational end, the algorithm can be sped up by initializing it with the result of the regular occupancy grid mapping approach, instead of an empty map. Finally. we note that only a small number of measurements are affected by flipping a grid cell in Line 5 of Table 9.3. While each sum is potentially huge, only a small number of elements has to be inspected when calculating the argmax. We leave the design of an appropriate data a structure to the reader as an exercise.

9.5 SUMMARY

This chapter introduced algorithms for learning occupancy grids. All algorithms in this chapter require exact pose estimates for the robot, hence they do not solve the general mapping problem.

- The standard occupancy mapping algorithm estimates for each grid cell individually the posterior probability of occupancy. It is an adaptation of the binary Bayes filter for static environments.
- Data from multiple sensors can be fused into a single map in two ways: By maintaining a single map using Bayes filters, and by maintaining multiple maps, one for each sensor modality, and extracting the most most pessimistic occupancy value when making navigation decisions. The latter procedure is preferable when different sensors are sensitive to different types of obstacles.
- The standard occupancy grid mapping algorithm relies on inverse measurement models, which reason from effects (measurements) to causes (occupancy). This differs from previous applications of Bayes filters in the context of localization, where the Bayes filter was based on a conventional measurement model that reasons from causes to effects.

- It is possible to learn inverse sensor models from the conventional measurement model, which models the sensor from causes to effects. To do so, one has to generate samples, and learn an inverse model using function approximation.
- The standard occupancy grid mapping algorithm does not maintain dependencies in the estimate of occupancy. This is a result of decomposing the map posterior estimation problem into a large number of single-cell posterior estimation problem.
- The full map posterior is generally not computable, due to the large number of maps that can be defined over a grid. However, it can be maximized. Maximizing it leads to maps that are more consistent with the data. However, the maximization requires the availability of all data, and the resulting maximum a posterior map does not capture the residual uncertainty in the map.

Without a doubt, occupancy grid maps and their various extensions are vastly popular in robotics. This is because they are extremely easy to acquire, and they capture important elements for robot navigation.

10

SIMULTANEOUS LOCALIZATION AND MAPPING

10.1 INTRODUCTION

This and the following chapters address one of the most fundamental problems in robotics, the *simultaneous localization and mapping problem*. This problem is commonly abbreviated as *SLAM*, and is also known as *Concurrent Mapping and Localization*, or CML. SLAM problems arise when the robot does not have access to a map of the environment; nor does it have access to its own poses. Instead, all it is given are measurements $z_{1:t}$ and controls $u_{1:t}$. The term "simultaneous localization and mapping" describes the resulting problem: In SLAM, the robot acquires a map of its environment while simultaneously localizing itself relative to this map. SLAM is significantly more difficult than all robotics problems discussed thus far: It is more difficult than localization in that the map is unknown and has to be estimated along the way. It is more difficult than mapping with known poses, since the poses are unknown and have to be estimated along the way.

From a probabilistic perspective, there are two main forms of the SLAM problem, which are both of equal practical importance. One is known as the *online SLAM problem*: It involves estimating the posterior over the momentary pose along with the map:

$$p(x_t, m \mid z_{1:t}, u_{1:t}) \tag{10.1}$$

Here x_t is the pose at time t, m is the map, and $z_{1:t}$ and $u_{1:t}$ are the measurements and controls, respectively. This problem is called the online SLAM problem since it only involves the estimation of variables that persist at time t. Many algorithms for the

online SLAM problem are incremental: they discard past measurements and controls once they have been processed.

The second SLAM problem is called the *full SLAM problem*. In full SLAM, we seek to calculate a posterior over the entire path $x_{1:t}$ along with the map, instead of just the current pose x_t:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \tag{10.2}$$

This subtle difference in the formulation of the SLAM problem between online and full SLAM has ramifications in the type algorithms that can be brought to bear. In particular, the online SLAM problem is the result of integrating out past poses from the full SLAM problem:

$$\begin{aligned} & p(x_t, m \mid z_{1:t}, u_{1:t}) \\ = & \int \int \cdots \int p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \, dx_1 \, dx_2 \ldots \, dx_{t-1} \end{aligned} \tag{10.3}$$

In online SLAM, these integrations are typically performed one-at-a-time, and they cause interesting changes of the dependency structures in SLAM that we will fully explore in the next chapter.

A second key characteristic of the SLAM problem has to do with the nature of the estimation problem. SLAM problems possess a continuous and a discrete component. The continuous estimation problem pertains to the location of the objects in the map and the robot's own pose variables. Objects may be landmarks in feature-based representation, or they might be object patches detected by range finders. The discrete nature has to do with correspondence: When an object is detected, a SLAM algorithm must reason about the relation of this object to previously detected objects. This reasoning is typically discrete: Either the object is the same as a previously detected one, or it is not.

We already encountered similar continuous-discrete estimation problems in previous chapters. For example, EKF localization 7.5 estimates the robot pose, which is continuous, but to do so it also estimates the correspondences of measurements and landmarks in the map, which are discrete. In this and the subsequent chapters, we will discuss a number of different techniques to deal with the continuous and the discrete aspects of the SLAM problem.

At times, it will be useful to make the correspondence variables explicit, as we did in Chapter 7 on localization. The online SLAM posterior is then given by

$$p(x_t, m, c_t \mid z_{1:t}, u_{1:t}) \tag{10.4}$$

and the full SLAM posterior by

$$p(x_{1:t}, m, c_{1:t} \mid z_{1:t}, u_{1:t}) \tag{10.5}$$

The online posterior is obtained from the full posterior by integrating out past robot poses and summing over all past correspondences:

$$\begin{aligned} & p(x_t, m, c_t \mid z_{1:t}, u_{1:t}) \\ = & \int\int \cdots \int \sum_{c_1} \sum_{c_2} \cdots \sum_{c_{t-1}} p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \; dx_1 \; dx_2 \ldots \; dx_{t-1} \end{aligned} \tag{10.6}$$

In both versions of the SLAM problems—online and full—estimating the full posterior (10.4) or (10.5) is the gold standard of SLAM. The full posterior captures all there is to be known about the map and the pose or the path. In practice, calculating a full posterior is usually infeasible. Problems arise from two sources: (1) the high dimensionality of the continuous parameter space, and (2) the large number of discrete correspondence variables. Many state-of-the-art SLAM algorithm construct maps with tens of thousands of features, or more. Even under known correspondence, the posterior over those maps alone involves probability distributions over spaces with 10^5 or more dimensions. This is in stark contrast to localization problems, in which posteriors were estimated over three-dimensional continuous spaces. Further, in most applications the correspondences are unknown. The number of possible assignment to the vector of all correspondence variables, $c_{1:t}$ grows exponentially in the time t. Thus, practical SLAM algorithms that can cope with the correspondence problem must rely on approximations.

The SLAM problem will be discussed in a number of subsequent chapter. The remainder of this chapter develops an EKF algorithm for the online SLAM problem. Much of this material builds on Chapter 3.3, where the EKF was introduced, and Chapter 7.5, where we applied the EKF to the mobile robot localization problem. We will derive a progression of EKF algorithms that first apply EKFs to SLAM with known correspondences, and then progress to the more general case with unknown correspondences.

10.2 SLAM WITH EXTENDED KALMAN FILTERS

10.2.1 Setup and Assumptions

Historically the earliest, and perhaps the most influential SLAM algorithm is based on the extended Kalman filter, or EKF. In a nutshell, the EKF SLAM algorithm applies the EKF to online SLAM using maximum likelihood data association. In doing so, EKF SLAM is subject to a number of approximations and limiting assumptions:

- **Feature-based maps.** Maps, in the EKF, are composed of point landmarks. For computational reasons, the number of point landmarks is usually small (e.g., smaller than 1,000). Further, the EKF approach tends to work well the less ambiguous the landmarks are. For this reason, EKF SLAM requires significant engineering of feature detectors, sometimes using artificial beacons or landmarks as features.
- **Gaussian noise.** As any EKF algorithm, EKF SLAM makes a Gaussian noise assumption for the robot motion and the perception. The amount of uncertainty in the posterior must be relatively small, since otherwise the linearization in EKFs tend to introduce intolerable errors.
- **Positive measurements.** The EKF SLAM algorithm, just like the EKF localizer discussed in Chapter 7.5, can only process positive sightings of landmarks. It cannot process negative information that arises from the absence of landmarks in a sensor measurements. This is a direct consequence of the Gaussian belief representation and was already discussed in Chapter 7.5.

10.2.2 SLAM with Known Correspondence

The SLAM algorithm for the case with known correspondence addresses the continuous portion of the SLAM problem only. Its development is in many ways parallel to the derivation of the EKF localization algorithm in Chapter 7.5, but with one key difference: In addition to estimating the robot pose x_t, the EKF SLAM algorithm also estimates the coordinates of all landmarks encountered along the way. This makes it necessary to include the landmark coordinates into the state vector.

1: **Algorithm EKF_SLAM_known_correspondences($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, c_t$):**

2: $$F_x = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & \underbrace{0 \cdots 0}_{2N} \end{pmatrix}$$

3: $$\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix}$$

4: $$G_t = I + F_x^T \begin{pmatrix} 0 & 0 & \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & \frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix} F_x$$

5: $$\bar{\Sigma}_t = G_t \; \Sigma_{t-1} \; G_t^T + F_x^T \; R_t \; F_x$$

6: $$Q_t = \begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}$$

7: for all observed features $z_t^i = (r_t^i \;\; \phi_t^i \;\; s_t^i)^T$ do

8: $j = c_t^i$

9: if landmark j never seen before

10: $$\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \\ \bar{\mu}_{j,s} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \\ s_t^i \end{pmatrix} + r_t^i \begin{pmatrix} \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \\ 0 \end{pmatrix}$$

11: endif

12: $$\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{j,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{j,y} - \bar{\mu}_{t,y} \end{pmatrix}$$

13: $$q = \delta^T \delta$$

14: $$\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \operatorname{atan2}(\delta_y, \delta_x) - \bar{\mu}_{t,\theta} \\ \bar{\mu}_{j,s} \end{pmatrix}$$

15: $$F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & \underbrace{0 \cdots 0}_{2j-2} & 0 & 0 & 1 & \underbrace{0 \cdots 0}_{2N-2j} \end{pmatrix}$$

16: $$H_t^i = \frac{1}{q} \begin{pmatrix} \sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & -\sqrt{q}\delta_x & \sqrt{q}\delta_y & 0 \\ \delta_y & \delta_x & -1 & -\delta_y & -\delta_x & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} F_{x,j}$$

17: $$K_t^i = \bar{\Sigma}_t \; H_t^{iT} (H_t^i \; \bar{\Sigma}_t \; H_t^{iT} + Q_t)^{-1}$$

18: endfor

19: $$\mu_t = \bar{\mu}_t + \textstyle\sum_i K_t^i (z_t^i - \hat{z}_t^i)$$

20: $$\Sigma_t = (I - \textstyle\sum_i K_t^i \; H_t^i) \; \bar{\Sigma}_t$$

21: return μ_t, Σ_t

Table 10.1 The extended Kalman fi lter (EKF) algorithm for the simultaneous localization and mapping problem, formulated here for a feature-based map and a robot equipped with sensors for measuring range and bearing. This version assumes knowledge of the exact correspondences.

For convenience, let us call the state vector comprising robot pose and the map the *combined state vector*, and denote this vector y_t. The combined vector is given by

$$\begin{aligned} y_t &= \begin{pmatrix} x_t \\ m \end{pmatrix} \\ &= \begin{pmatrix} x & y & \theta & m_{1,x} & m_{1,y} & s_1 & m_{2,x} & m_{2,y} & s_2 & \dots & m_{N,x} & m_{N,y} & s_N \end{pmatrix}^T \end{aligned} \tag{10.7}$$

Here x, y, and θ denote the robot's coordinates at time t, $m_{i,x}, m_{i,y}$ are the coordinates of the i-th landmark, for $i = 1, \dots, N$, and s_i is its signature. The dimension of this state vector is $3N + 3$, where N denotes the number of landmarks in the map. Clearly, for any reasonable number of N, this vector is significantly larger than the pose vector that is being estimated in Chapter 7.5, which introduced the EKF localization algorithm. EKF SLAM calculates the online posterior

$$p(y_t \mid z_{1:t}, u_{1:t}) \tag{10.8}$$

The EKF SLAM algorithm is depicted in Table 10.1—notice the similarity to the EKF localization algorithm in Table 7.2. Lines 2 through 5 apply the motion update, whereas Lines 6 through 20 incorporate the measurement vector. In particular, Lines 3 and 5 manipulate the mean and covariance of the belief in accordance to the motion model. This manipulation only affects those elements of the belief distribution concerned with the robot pose. It leaves all mean and covariance variables for the map unchanged, along with the pose-map covariances. Lines 7 through 18 iterate through all measurements. The test in Line 9 returns true only for landmarks for which we have no initial location estimate. For those, Line 10 initializes the location of such a landmark by the projected location obtained from the corresponding range and bearing measurement. As we shall discuss below, this step is important for the linearization in EKFs; it would not be needed in linear Kalman filters. For each measurement, an "expected" measurement is computed in Line 14, and the corresponding Kalman gain is computed in Line 17. Notice that the Kalman gain is a matrix of size 3 by $3N + 3$. This matrix is usually non-sparse, that is, information is propagated through the entire state estimate. The final filter update then occurs in Lines 19 and 20, where the innovation is folded back into the robot's belief.

The fact that the Kalman gain is fully populated for all state variables—and not just the observed landmark and the robot pose—is important. In SLAM, observing a landmark does not just improve the position estimate of this very landmark, but that of other landmarks as well. This effect is mediated by the robot pose: Observing a landmark improves the robot pose estimate, and as a result it eliminates some of the uncertainty

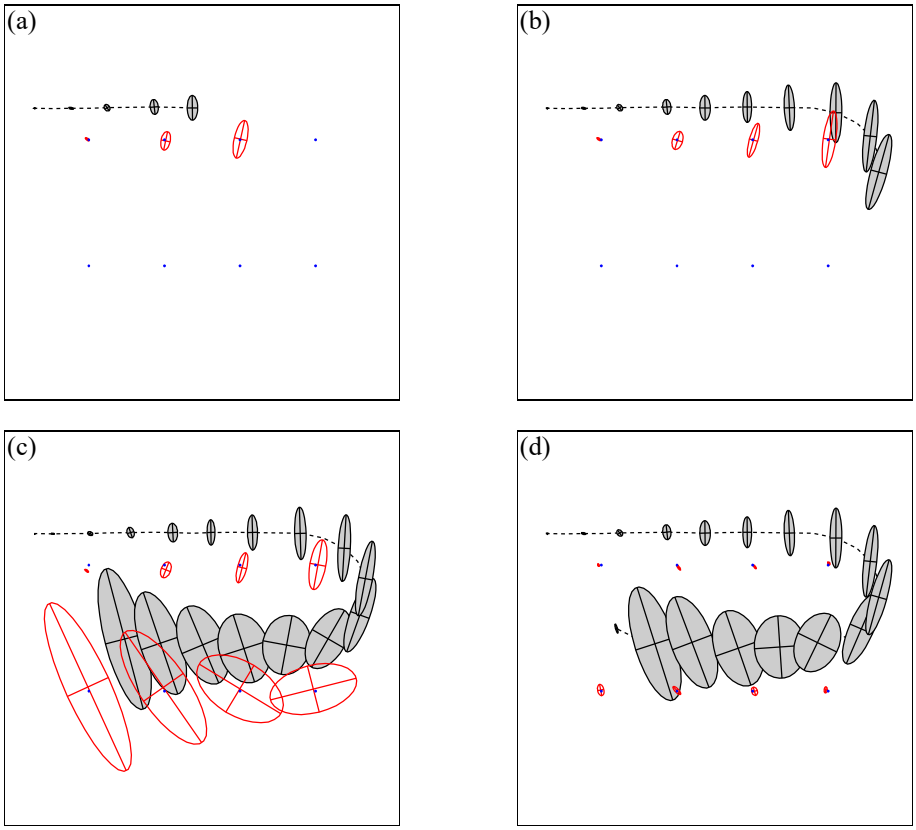

Figure 10.1 EKF applied to the online SLAM problem. The robot's path is a dotted line, and its estimations of its own position are shaded ellipses. Eight distinguishable landmarks of unknown location are shown as small dots, and their location estimations are shown as white ellipses. In (a)–(c) the robot's positional uncertainty is increasing, as is its uncertainty about the landmarks it encounters. In (d) the robot senses the first landmark again, and the uncertainty of *all* landmarks decreases, as does the uncertainty of its current pose.

of landmarks previously seen by the same robot. The amazing effect here is that we do not have to model past poses explicitly—which would put us into the realm of the full SLAM problem and make the EKF a non-realtime algorithm. Instead, this dependence is captured in the Gaussian posterior, more specifically, in the off-diagonal covariance elements of the matrix Σ_t.

Figure 10.1 illustrates the EKF SLAM algorithm for an artificial example. The robot navigates from a start pose which serves as the origin of its coordinate system. As it moves, its own pose uncertainty increases, as indicated by uncertainty ellipses of growing diameter. It also senses nearby landmarks and maps them with an uncertainty that combines the fixed measurement uncertainty with the increasing pose uncertainty. As a result, the uncertainty in the landmark locations grows over time. In fact, it parallels that of the pose uncertainty at the time a landmark is observed. The interesting transition happens in Figure 10.1d: Here the robot observes the landmark it saw in the very beginning of mapping, and whose location is relatively well known. Through this observation, the robot's pose error is reduced, as indicated in Figure 10.1d—notice the very small error ellipse for the final robot pose! Further, this observation also reduces the uncertainty for other landmarks in the map! This phenomenon arises from a correlation that is expressed in the covariance matrix of the Gaussian posterior. Since most of the uncertainty in earlier landmarks is causes by the robot pose, and this very uncertainty persists over time, the location estimates of those landmarks are correlated. When gaining information on the robot's pose, this information spreads to previously observed landmarks. This effect is probably the most important characteristic of the SLAM posterior: Information that helps localize the robot is propagated through map, and as a result improves the localization of other landmarks in the map.

10.2.3 Mathematical Derivation

The derivation of the EKF SLAM algorithm for the case with known correspondences largely parallels that of the EKF localizer in Chapter 7.5. The key difference is the augmented state vector, which now includes the locations of all landmarks in addition to the robot pose.

In SLAM, the initial pose is taken to be to origin of the coordinate system. This definition is somewhat arbitrary, in that it can be replaced by any coordinate. None of the landmark locations are known initially. The following initial mean and covariance express this belief:

$$\mu_0 = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \end{pmatrix}^T \tag{10.9}$$

$$\Sigma_0 = \begin{pmatrix} 0 & 0 & 0 & \infty & \cdots & \infty \\ 0 & 0 & 0 & \infty & \cdots & \infty \\ 0 & 0 & 0 & \infty & \cdots & \infty \\ \infty & \infty & \infty & \infty & \cdots & \infty \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \infty & \infty & \infty & \infty & \cdots & \infty \end{pmatrix} \tag{10.10}$$

The covariance matrix is of size $(3N+3) \times (3N+3)$. It is composed of a small 3×3 matrix of zeros for the robot pose variables. All other covariance values are infinite.

As the robot moves, the state vector changes according to the standard noise-free velocity model (see Equations (5.13) and (7.4)). In SLAM, this motion model is extended to the augmented state vector:

$$y_t = y_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega_t \Delta t + \gamma_t \Delta t \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{10.11}$$

The variables x, y, and θ denote the robot pose in y_{t-1}. Because the motion only affects the robot's pose and all landmarks remain where they are, only the first three elements in the update are non-zero. This enables us to write the same equation more compactly:

$$y_t = y_{t-1} + F_x \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega_t \Delta t + \gamma_t \Delta t \end{pmatrix} \tag{10.12}$$

Here F_x is a matrix that maps the 3-dimensional state vector into a vector of dimension $3N + 3$.

$$F_x = \begin{pmatrix} 1 & 0 & 0 & \underbrace{0 \cdots 0}_{3N \text{ columns}} \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix} \tag{10.13}$$

The full motion model with noise is then as follows

$$y_t = \underbrace{y_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega \Delta t \end{pmatrix}}_{g(u_t, y_{t-1})} + \mathcal{N}(0, F_x^T \; R_t \; F_x) \tag{10.14}$$

where $F_x^T R_t F_x$ extends the covariance matrix to the dimension of the full state vector squared.

As usual in EKFs, the motion function g is approximated using a first degree Taylor expansion

$$g(u_t, y_{t-1}) \quad \approx \quad g(u_t, \mu_{t-1}) \; + \; G_t \; (y_{t-1} - \mu_{t-1}) \tag{10.15}$$

where the Jacobian $G_t = g'(u_t, \mu_{t-1})$ is the derivative of g at u_t, as in Equation (7.8). Obviously, the additive form in (10.14) enables us to decompose this Jacobian into an identity matrix of dimension $(3N + 3) \times (3N + 3)$ (the derivative of y_{t-1}) plus a low-dimensional Jacobian g_t that characterizes the change of the robot pose:

$$G_t \quad = \quad I + F_x^T \; g_t \; F_x \tag{10.16}$$

with

$$g_t \quad = \quad \begin{pmatrix} 0 & 0 & \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & \frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix} \tag{10.17}$$

Plugging these approximations into the standard EKF algorithm gives us Lines 2 through 5 of Table 10.1. Obviously, several of the matrices multiplied in line 5 are sparse, which should be exploited when implementing this algorithm. The result of this update are the mean $\bar{\mu}_t$ and the covariance $\bar{\Sigma}_t$ of the estimate at time t after updating the filter with the control u_t, but before integrating the measurement z_t.

The derivation of the measurement update is similar to the one in Section 7.5. In particular, we are given the following measurement model

$$z_t^i \quad = \quad \underbrace{\begin{pmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ \text{atan2}(m_{j,y} - y, m_{j,x} - x) - \theta \\ m_{j,s} \end{pmatrix}}_{h(y_t, j)} + \mathcal{N}(0, \underbrace{\begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}}_{Q_t}) \tag{10.18}$$

where x, y, and θ denotes the pose of the robot, i is the index of an individual landmark observation in z_t, and $j = c_t^i$ is the index of the observed landmark at time t. This

expression is approximated by the linear function

$$h(y_t, j) \quad \approx \quad h(\bar{\mu}_t, j) \; + \; H_t^i \; (y_t - \bar{\mu}_t) \tag{10.19}$$

Here H_t^i is the derivative of h with respect to the full state vector y_t. Since h depends only on two elements of that state vector, the robot pose x_t and the location of the j-th landmark m_j, the derivative factors into a low-dimensional Jacobian h_t^i and a matrix $F_{x,j}$, which maps h_t^i into a matrix of the dimension of the full state vector:

$$H_t^i \quad = \quad h_t^i \; F_{x,j} \tag{10.20}$$

Here h_t^i is the Jacobian of the function $h(y_t, j)$ at $\bar{\mu}_t$, calculated with respect to the state variables x_t and m_j:

$$h_t^i \quad = \quad \begin{pmatrix} \frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}_t} & \frac{y_t - \bar{\mu}_{t,y}}{\sqrt{q}_t} & 0 & \frac{\bar{\mu}_{t,x} - m_{j,x}}{\sqrt{q}_t} & \frac{\bar{\mu}_{t,y} - y_t}{\sqrt{q}_t} & 0 \\ \frac{\bar{\mu}_{t,y} - y_t}{q_t} & \frac{m_{j,x} - \bar{\mu}_{t,x}}{q_t} & -1 & \frac{y_t - \bar{\mu}_{t,y}}{q_t} & \frac{\bar{\mu}_{t,x} - m_{j,x}}{q_t} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{10.21}$$

The scalar $q_t = (m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2$, and as before, $j = c_t^i$ is the landmark that corresponds to the measurement z_t^i. The matrix $F_{x,j}$ is of dimension $(3N+3) \times 5$. It maps the low-dimensional matrix h_t^i into a matrix of dimension $(3N + 3) \times 3$:

$$F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & \underbrace{0 \cdots 0}_{2j-2} & 0 & 0 & 1 & \underbrace{0 \cdots 0}_{2N-2j} \end{pmatrix} \tag{10.22}$$

These expressions make up for the gist of the Kalman gain calculation in Lines 8 through 17 in our EKF SLAM algorithm in Table 10.1, with one important extension.

When a landmark is observed for the first time, its initial pose estimate in Equation (10.9) leads to a poor linearization. This is because with the default initialization in (10.9), the point about which h is being linearized is $(\hat{\mu}_{j,x} \;\; \hat{\mu}_{j,y} \;\; \hat{\mu}_{j,s})^T = (0 \;\; 0 \;\; 0)^T$, which is a poor estimator of the actual landmark location. A better landmark estimator is given in Line 10 Table 10.1. Here we initialize the landmark estimate $(\hat{\mu}_{j,x} \;\; \hat{\mu}_{j,y} \;\; \hat{\mu}_{j,s})^T$ with the expected position. This expected position is derived from the expected robot pose and the measurement variables for this landmark

$$\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \\ \bar{\mu}_{j,s} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \\ s_t^i \end{pmatrix} + r_t^i \begin{pmatrix} \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \\ 0 \end{pmatrix} \tag{10.23}$$

We note that this initialization is only possible because the measurement function h is bijective. Measurements are two-dimensional, as are landmark locations. In cases where a measurement is of lower dimensionality than the coordinates of a landmark, h is a true projection and it is impossible to calculate a meaningful expectation for $(\bar{\mu}_{j,x} \;\; \bar{\mu}_{j,y} \;\; \bar{\mu}_{j,s})^T$ from a single measurement only. This is, for example, the case in computer vision implementations of SLAM, since cameras often calculate the angle to a landmark but not the range. SLAM is then usually performed by integrating multiple sightings to and apply triangulation to determine an appropriate initial location estimate.

Finally, we note that the EKF algorithm requires memory that is quadratic in N, the number of landmarks in the map. Its update time is also quadratic in N. The quadratic update complexity stems from the matrix multiplications that take place at various locations in the EKF.

10.3 EKF SLAM WITH UNKNOWN CORRESPONDENCES

10.3.1 The General EKF SLAM Algorithm

The EKF SLAM algorithm with known correspondences is now extended into the general EKF SLAM algorithm, which uses an incremental maximum likelihood (ML) estimator to determine correspondences. Table 10.2 depicts the algorithm. The input to this algorithm now lacks a correspondence variable c_t. Instead, it includes the momentary size of the map, N_{t-1}.

1: **Algorithm EKF_SLAM($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, N_{t-1}$):**

2: $N_t = N_{t-1}$

3: $F_x = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix}$

4: $\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \omega_t\Delta t \end{pmatrix}$

5: $G_t = I + F_x^T \begin{pmatrix} 0 & 0 & \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & \frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & 0 \end{pmatrix} F_x$

6: $\bar{\Sigma}_t = G_t\ \Sigma_{t-1}\ G_t^T + F_x^T\ R_t\ F_x$

7: $Q_t = \begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}$

8: for all observed features $z_t^i = (r_t^i\ \ \phi_t^i\ \ s_t^i)^T$ do

9: $\begin{pmatrix} \bar{\mu}_{N_t+1,x} \\ \bar{\mu}_{N_t+1,y} \\ \bar{\mu}_{N_t+1,s} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \\ s_t^i \end{pmatrix} + r_t^i \begin{pmatrix} \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \\ 0 \end{pmatrix}$

10: for $k = 1$ to N_t+1 do

11: $\delta_k = \begin{pmatrix} \delta_{k,x} \\ \delta_{k,y} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{k,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{k,y} - \bar{\mu}_{t,y} \end{pmatrix}$

12: $q_k = \delta_k^T \delta_k$

13: $\hat{z}_t^k = \begin{pmatrix} \sqrt{q_k} \\ \text{atan2}(\delta_{k,y}, \delta_{k,x}) - \bar{\mu}_{t,\theta} \\ \bar{\mu}_{k,s} \end{pmatrix}$

14: $F_{x,k} = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix}$

15: $H_t^k = \frac{1}{q_k} \begin{pmatrix} \sqrt{q}_k\delta_{k,x} & -\sqrt{q}_k\delta_{k,y} & 0 & -\sqrt{q}_k\delta_{k,x} & \sqrt{q}_k\delta_{k,y} & 0 \\ \delta_{k,y} & \delta_{k,x} & -1 & -\delta_{k,y} & -\delta_{k,x} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} F_{x,k}$

16: $\Psi_k = H_t^k\ \bar{\Sigma}_t\ [H_t^k]^T + Q_t$

17: $\pi_k = (z_t^i - \hat{z}_t^k)^T\ \Psi_k^{-1}\ (z_t^i - \hat{z}_t^k)$

18: endfor

19: $\pi_{N_t+1} = \alpha$

20: $j(i) = \underset{k}{\text{argmin}}\ \pi_k$

21: $N_t = \max\{N_t, j(i)\}$

22: $K_t^i = \bar{\Sigma}_t\ [H_t^{j(i)}]^T \Psi_{j(i)}^{-1}$

23: endfor

24: $\mu_t = \bar{\mu}_t + \sum_i K_t^i\ (z_t^i - \hat{z}_t^{j(i)})$

25: $\Sigma_t = (I - \sum_i K_t^i\ H_t^{j(i)})\ \bar{\Sigma}_t$

26: return μ_t, Σ_t

Table 10.2 The EKF SLAM algorithm with ML correspondences, shown here with outlier rejection.

The motion update in Lines 3 through 6 is equivalent to the one in **EKF_SLAM_known_correspondences** in Table 10.1. The measurement update loop, however, is different. Starting in Line 8, it first creates the hypothesis of a new landmark with index $N_t + 1$; this index is one larger than the landmarks in the map at this point in time. The new landmark's location is initialized in Line 9, by calculating its expected location given the estimate of the robot pose and the range and bearing in the measurement. Line 9 also assigns the observed signature value to this new landmark. Next, various update quantities are then computed in Lines 10 through 18 for all N_t+1 possible landmarks, including the "new" landmark. Line 19 sets the threshold for the creation of a new landmark: A new landmark is created if the Mahalanobis distance to all existing landmarks in the map exceeds the value α. The ML correspondence is then selected in Line 20. If the measurement is associated with a previously unseen landmark, the landmark counter is incremented in Line 21, and various vectors and matrices are enlarged accordingly—this somewhat tedious step is not made explicit in Table 10.2. The update of the EKF finally takes place in Lines 24 and 25. The algorithm **EKF_SLAM** returns the new number of landmarks N_t along with the mean μ_t and the covariance Σ_t.

The derivation of this EKF SLAM follows directly from previous derivations. In particular, the initialization in Line 9 is identical to the one in Line 10 in **EKF_SLAM_known_correspondences**, Table 10.1. Lines 10 through 18 parallel Lines 12 through 17 in **EKF_SLAM_known_correspondences**, with the added variable π_k needed for calculating the ML correspondence. The selection of the ML correspondence in Line 20, and the definition of the Mahalanobis distance in line 17, is analogous to the ML correspondence discussed in Chapter 7.6; in particular, the algorithm **EKF_localization** in Table 7.3 on page 175 used an analogous equation do determine the most likely landmark. (Line 14). The measurement updates in Lines 24 and 25 of Table 10.2 are also analogous to those in the EKF algorithm with known correspondences, assuming that the participating vectors and matrices are of the appropriate dimension in case the map has just been extended.

Our example implementation of **EKF_SLAM** can be made more efficient by restricting the landmarks considered in Lines 10 through 18 to those that are near the robot. Further, many of the values and matrices calculated in this inner loop can safely be cached away when looping through more than one feature measurement vector z_t^i. in practice, a good management of features in the map and a tight optimization of this loop can greatly reduce the running speed.

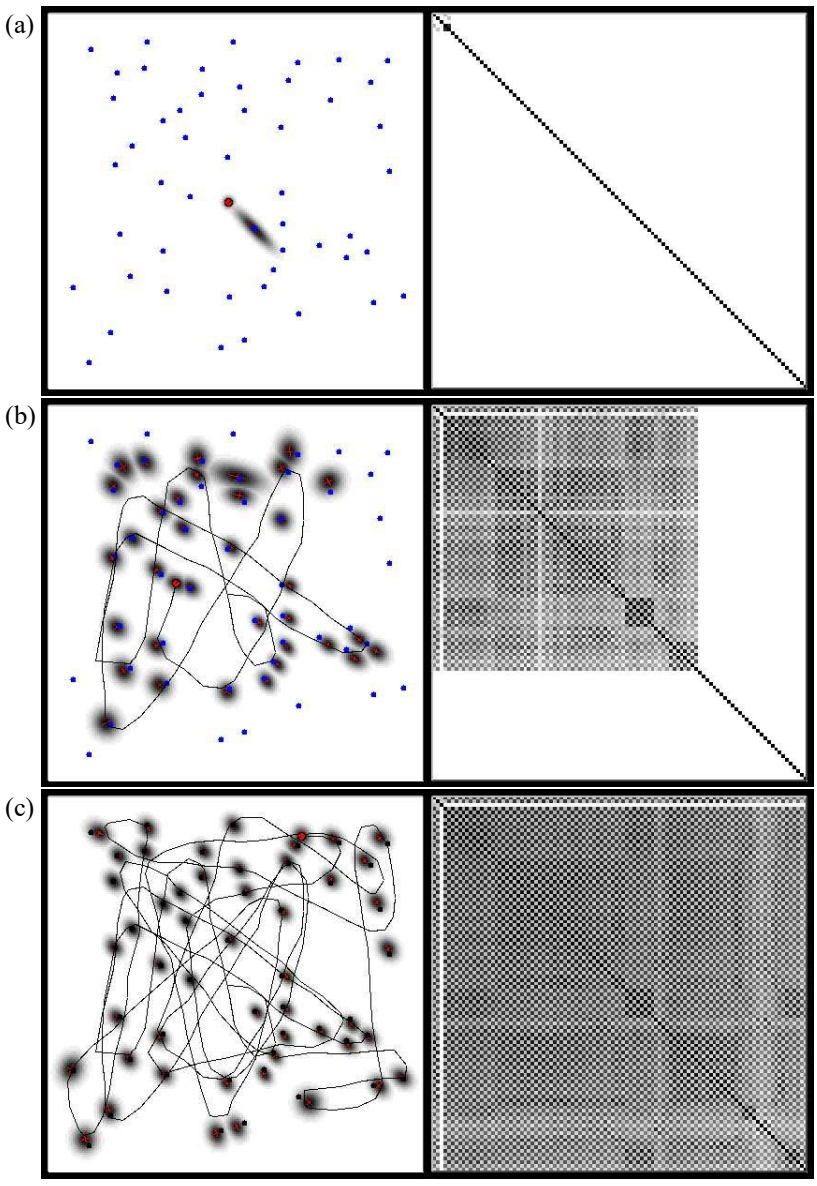

Figure 10.2 EKF SLAM with known data association in a simulated environment. The map is shown on the left, with the gray-level corresponding to the uncertainty of each landmark. The matrix on the right is the correlation matrix, which is the normalized covariance matrix of the posterior estimate. After some time, all x- and all y-coordinate estimates become fully correlated.

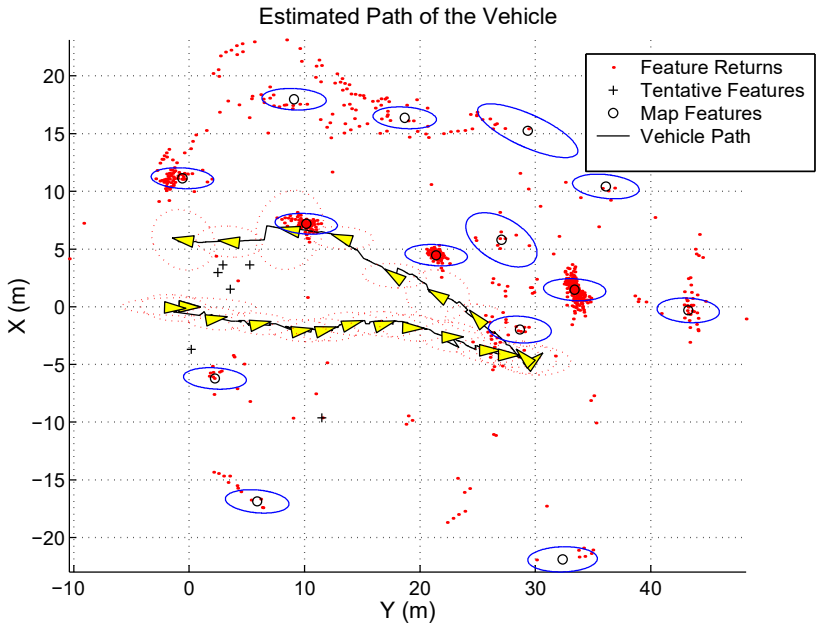

Figure 10.3 Example of Kalman fi lter estimation of the map and the vehicle pose. Image courtesy of Stefan Williams and Hugh Durrant-Whyte from the Australian Centre for Field Robotics, University of Sydney, Australia.

10.3.2 Examples

Figure 10.2 shows the EKF SLAM algorithm—here with known correspondence—applied in simulation. The left panel of each of the three diagrams plots the posterior distributions, marginalized for the individual landmarks and the robot pose. The right side depicts the correlation matrix for the augmented state vector y_t; the correlation is the normalized covariance. As is easily seen from the result in Figure 10.2c, over time all x- and y-coordinate estimates become fully correlated. This means the map becomes known in relative terms, up to an uncertain global location that cannot be reconciled. This highlights an important characteristic of the SLAM problem: The absolute coordinates of a map relative to the coordinate system defined by the initial robot pose can only be determined in approximation, whereas the relative coordinates can be determined asymptotically with certainty.

In practice, EKF SLAM has been applied successfully to a large range of navigation problems, involving airborne, underwater, indoor, and various other vehicles. Fig-

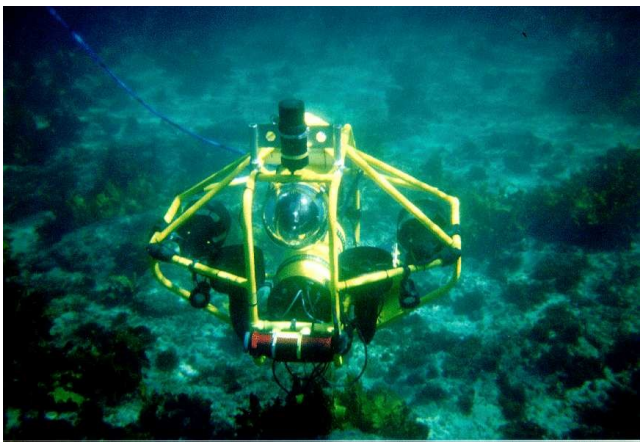

Figure 10.4 Underwater vehicle Oberon, developed at the University of Sydney. Image courtesy of Stefan Williams and Hugh Durrant-Whyte from the Australian Centre for Field Robotics, University of Sydney, Australia.

ure 10.3 shows an example result obtained using the underwater robot Oberon, developed at the University of Sydney, Australia, and shown in Figure 10.4. This vehicle is equipped with a pencil sonar, a sonar that can scan at very high resolutions an detect obstacles up to 50 meters away. To facilitate the mapping problem, researchers have deposited long, small vertical objects in the water, which can be extracted from the sonar scans with relative ease. In this specific experiment, there is a row of such objects, spaced approximately 10 meters apart. In addition, a more distant cliff offers additional point features that can be detected using the pencil sonar. In the experiment shown in Figure 10.3, the robot moves by these landmarks, then turns around and moves back. While doing so, it measures and integrates detected landmarks into its map. using the Kalman filter approach described in this chapter.

The map shown in Figure 10.3 shows the robot's path, marked by the triangles connected by a line. Around each triangle one can see an ellipse, which corresponds to the covariance matrix of the Kalman filter estimate projected into the robot's x-y position. The ellipse shows the variance; the larger it is, the less certain the robot is about its current pose. Various small dots in Figure 10.3 show landmark sightings, obtained by searching the sonar scan for small and highly reflective objects. The majority of these sightings is rejected, using a mechanism described further below, in the section that follows. However, some are believed to correspond to a landmark and added to the map. At the end of the run, the robot has classified 14 such objects as landmarks, each of which is plotted with the projected uncertainty ellipse in Figure 10.3. These

landmarks include the artificial landmarks put out by the researchers, but they also include various other terrain features in the vicinity of the robot. The residual pose uncertainty is small. In this example, the robot successfully finds and maps all of the artificial landmarks in this highly noisy domain.

10.3.3 Feature Selection and Map Management

Making EKF SLAM robust in practice often requires additional techniques for managing maps. Many of them pertain to the fact that the Gaussian noise assumption is unrealistic, and many spurious measurements occur in the far tail end of the noise distribution. Such spurious measurements can cause the creation of fake landmarks in the map which, in turn, negatively affect the localization of the robot.

Many state-of-the-art techniques possess mechanisms to deal with outliers in the measurement space. Such outliers are defined as spurious landmark sightings outside the uncertainty range of any landmark in the map. The most simple technique to compensate such outliers is to maintain a *provisional landmark list*. Instead of augmenting the map by a new landmark once a measurement indicates the existence of a new landmark, such a new landmark is first added to a provisional list of landmarks. This list is just like the map, but landmarks on this list are *not* used to adjust the robot pose (the corresponding gradients in the measurement equations are set to zero). Once a landmark has consistently been observed and its uncertainty ellipse has shrunk, it is transitioned into the regular map.

In practical implementations, this mechanism tends to reduce the number of landmarks in the map by a significant factor, while still retaining all physical landmarks with high probability. A further step, also commonly found in state-of-the-art implementations, is to maintain a posterior likelihood of the existence of a landmark. Such a probability may implemented as log-odds ratio and be denoted o_j for the j-th landmark in the map. Whenever the j-th landmark is m_j observed, o_j is incremented by a fixed value. Not observing m_j when it would be in the perceptual range of the robot's sensors leads to a decrement of o_j. Since it can never be known with certainty whether a landmark is within a robot's perceptual range, the decrement may factor in the probability of such an event. Landmarks are removed from the map when the value o_j drops below a threshold. Such techniques lead to much leaner maps in the face of non-Gaussian measurement noise.

As noted previously, the maximum likelihood approach to data association has a clear limitation. This limitation arises from the fact that the maximum likelihood approach

deviates from the idea of full posterior estimation in probabilistic robotic. Instead of maintaining a joint posterior over augmented states and data associations, it reduces the data association problem to a deterministic determination, which is treated as if the maximum likelihood association was always correct. This limitation makes EKF brittle with regards to landmark confusion, which in turn can lead to wrong results. In practice, researchers often remedy the problem by choosing one of the following two methods, both of which reduce the chances of confusing landmarks:

- **Spatial arrangement.** The further apart landmarks are, the smaller the chance to accidentally confuse them. It is therefore common practice to choose landmarks that are sufficiently far away from each other to that the probability of confusing one with another is negligible. This introduces an interesting trade-off: a large number of landmarks increases the danger of confusing them. Too few landmarks makes it more difficult to localize the robot, which in turn also increases the chances of confusing landmarks. Little is currently known about the optimal density of landmarks, and researchers often use intuition when selecting specific landmarks.
- **Signatures.** When selecting appropriate landmarks, it is essential to to maximize the perceptual distinctiveness of landmarks. For example, doors might possess different colors, or corridors might have different widths. The resulting signatures are essential for successful SLAM.

With these additions, the EKF SLAM algorithm has indeed been applied successfully to a wide range of practical mapping problems, involving robotic vehicles in the air, on the ground, in the deep see, and in abandoned mines.

A key limitation of EKF SLAM lies in the necessity to select appropriate landmarks. By doing so, most of the sensor data is usually discarded. Further, the quadratic update time of the EKF limits this algorithm to relatively scarce maps with less than 1,000 features. In practice, one often seeks maps with 10^6 features or more, in which case the EKF ceases to be applicable.

The relatively low dimensionality of the map tends to create a harder data association problem. This is easily verified: When you open your eyes and look at the full room you are in, you probably have no difficulty to recognize where you are! however, if you are only told the location of a small number of landmarks—e.g., the location of all light sources—the decision is much harder. As a result, data association in EKF SLAM is more difficult than in some of the SLAM algorithms discussed in subsequent chapter, and capable of handling orders of magnitude more features. This culminates into the principal dilemma of the EKF SLAM algorithm: While incremental maxi-

mum likelihood data association might work well with dense maps with hundreds of millions of features, it tends to be brittle with scarce maps. However, EKFs require sparse maps because of the quadratic update complexity. In subsequent chapters, we will discuss SLAM algorithms that are more efficient and can handle much large maps. We will also discuss more robust data association techniques. For its many limitation, the value of the EKF SLAM algorithm presented in this chapter is mostly historical.

10.4 SUMMARY

This chapter described the general SLAM problem, and introduced the EKF approach.

- The SLAM problem is defined as a concurrent localization and mapping problem, in which a robot seeks to acquire a map of the environment while simultaneously seeking to localize itself relative to this map.
- The SLAM problem comes in two versions: online and global. Both problems involve the estimation of the map. The online problem seeks to estimate the momentary robot pose, whereas the global problem seeks to determine all poses. Both problem are of equal importance in practice, and have found equal coverage in the literature.
- The EKF SLAM algorithm is arguably the earliest SLAM algorithm. It applies the extended Kalman filter to the online SLAM problem. With known correspondences, the resulting algorithm is incremental. Updates require time quadratic in the number of landmarks in the map.
- When correspondences are unknown, the EKF SLAM algorithm applies an incremental maximum likelihood estimator to the correspondence problem. The resulting algorithm works well when landmarks are sufficiently distinct.
- Additional techniques were discussed for managing maps. Two common strategies for identifying outliers include provisional list for landmarks that are not yet observed sufficiently often, and a landmark evidence counter that calculates the posterior evidence of the existence of a landmark.
- EKF SLAM has been applied with considerable success in a number of robotic mapping problems. Its main drawbacks are the need for sufficiently distinct landmarks, and the computational complexity required for updating the filter.

In practice, EKF SLAM has been applied with some success. When landmarks are sufficiently distinct, the approach approximates the posterior well. The advantage of

calculating a full posterior are manifold: It captures all residual uncertainty and enables the robot to reason about its control taking its true uncertainty into account. However, the EKF SLAM algorithm suffers from its enormous update complexity, and the limitation to sparse maps. This, in turn, makes the data association problem a difficult one, and EKF SLAM tends to work poorly in situations where landmarks are highly ambiguous. Further brittleness is due to the fact that the EKF SLAM algorithm relies on an incremental maximum likelihood data association technique. This technique makes is impossible to revise part data associations, and can induce failure when the ML data association is incorrect.

The EKF SLAM algorithm applies to the online SLAM problem; it is inapplicable to the full SLAM problem. In the full SLAM problem, the addition of a new pose to the state vector at each time step would make both the state vector and the covariance grow without bounds. Updating the covariance would therefore require an ever-increasing amount of time, and the approach would quickly run out of computational time no matter how fast the processor.

10.5 BIBLIOGRAPHICAL REMARKS

Place them here!

10.6 PROJECTS

1. Develop an incremental algorithm for posterior estimation of poses and maps (with known data association) that does not rely on linearizing the motion model and the perceptual model. Our suggestion: Replace the Kalman filter by particle filters.

2. The basic Kalman filter approach is unable to cope with the data association problem in a sound statistical way. Develop an algorithm (and a statistical framework) for posterior estimation with unknown data association, and characterize its advantages and disadvantages. We suggest to use a mixture of Gaussians representation.

3. Based on the previous problem, develop an approximate method for posterior estimation with unknown data association, where the time needed for each incremental update step does not grow over time (assuming a fixed number of landmarks).

4. Develop a Kalman filter algorithm which uses local occupancy grid maps as its basic components, instead of landmarks. Among other things, problems that have to be solved are how to relate local grids to each other, and how to deal with the ever-growing number of local grids.

5. Develop and algorithm that learns its own features for Kalman filter mapping. There could be two different variants: One which chooses features so as to minimize the uncertainty (entropy) in the posterior, another which is given access to the correct map and seeks to maximize the probability of the correct map under the posterior.

11

THE EXTENDED INFORMATION FORM ALGORITHM

11.1 INTRODUCTION

The EKF SLAM algorithm described in the previous chapter is subject to a number of limitations, as discussed there. In this chapter, we introduce an alternative SLAM algorithm, called the *extended information form SLAM algorithm*, of *EIF SLAM*. EIF SLAM has in common with the EKF that it represents the posterior by a Gaussian. Unlike EKF SLAM, which solves the online SLAM problem, EIF SLAM solves the full SLAM problem. The reader may recall that the online SLAM posterior is defined over the map and the most recent robot pose, whereas the full SLAM posterior is defined over the map and the entire robot path. Another key difference is that EIF SLAM represents the posterior Gaussian in its information form, that is, via the information (or precision) matrix and the information state vector. Clearly, the information and the moments representation carry the same information; however, as we shall see, the update mechanisms of the information form are substantially different from that of the EKF.

EIF SLAM is not an incremental algorithm; this is because it calculates posteriors over a robot path. Instead, EIF SLAM processes an entire data set at a time, to generate the full SLAM posterior. This approach is fundamentally different from EKF SLAM, which is incremental and enables a robot to update its map forever. EIF SLAM is best suited to problems where one seeks a map from a data set of fixed size, and can afford to hold the data in memory up to the time where the map is built.

Because EIF SLAM has all data available during mapping, it can apply improved linearization and data association techniques. Whereas in EKF SLAM, the linearization and the correspondence for a measurement at time t are calculated based on the data up to time t, in EIF SLAM *all* data can be used to linearize and to calculate correspon-

dences. Further, EIF SLAM iterates the construction of the map, the calculation of correspondence variables, and the linearization of the measurement models and motion models, so as to obtain the best estimate of all of those quantities. In doing so, EIF SLAM tends to produce maps that are superior in accuracy to that of EKFs. EIF SLAM is less brittle to erroneous data association, and it can cope with higher rotational uncertainties in the data than EKF SLAM. And finally, EIF SLAM is applicable to maps many orders of magnitude larger than those that can be accommodated by the EKF.

This chapter first describe the intuition behind EIF SLAM and its basic updates steps. We then derive the various update steps mathematically and prove its correctness relative to specific linear approximations. We then discuss actual implementations, in which posteriors are calculated over maps

11.2 INTUITIVE DESCRIPTION

The basic intuition behind EIF SLAM is remarkably simple. Suppose we are given a set of measurements $z_{1:t}$ with associated correspondence variables $c_{1:t}$, and a set of controls $u_{1:t}$. The first step of EIF SLAM will be to use those data to construct an information matrix and an information vector defined over the joint space of robot poses $x_{1:t}$ and the map $m = \{m_j\}$. We will denote the information matrix by Ω and the information vector by ξ. As we shall see below, each measurement and each control leads to a *local* update of Ω and ξ. In fact, the rule for incorporating a control or a measurement into Ω and ξ is a local addition, paying tribute to the important fact that information is an additive quantity.

Make Figure: a robot path, landmarks, and an information matrix.

Figure **??** illustrates the process of constructing the information matrix. Each control u_t provides information about the relative value of the the robot pose at time $t-1$ and the pose at time t. We can think of this information as a constraint between x_{t-1} and x_t, or a "spring" in a spring-mass model of the world. The control u_t is incorporated into Ω and ξ by adding a values between the rows and columns connecting x_{t-1} and x_t. The magnitude of these values corresponds to the stiffness of the constraint—or the residual uncertainty between the relative poses caused by the noise in the motion model. This is illustrated in Figure **??**, which shows the link between two robot poses, and the corresponding element in the information matrix.

Similarly, consider a measurement z_t^i. This measurement provides information between the location of the landmark $j = c_t^i$ and the robot pose x_t at time t. Again, this information can be thought of as a constraint. The respective update in the EIF amounts to adding values between the elements linking the pose x_t and the map feature m_j. As before, the magnitude of these values reflects the residual uncertainty due to the measurement noise; the less noisy the sensor, the larger the value added to Ω and ξ.

After incorporating all measurements $z_{1:t}$ and controls $u_{1:t}$, we obtain an information matrix Ω whose off-diagonal elements are all zero with two exceptions: Between any two consecutive poses x_{t-1} and x_t will be a non-zero value that represents the information link introduced by the control u_t. Also non-zero will be any element between a map feature m_j and a pose x_t, if m_j was observed when the robot was at x_t. All elements between pairs of different landmarks remain zero. This reflects the fact that we never received information pertaining to their relative location—all we receive in SLAM are measurements that constrain the location of a landmark relative to a robot pose. Thus, the information matrix is sparse; all but a linear number of elements are zero.

Of course, the information representation does not give us a map; neither does it tell us the estimated path of the robot. Both obtained via $\mu = \Omega^{-1}\xi$ (see Equation (3.72) on page 56). This operation involves the inversion of a sparse matrix. This raises the question on how efficiently we can recover the map, and the posterior Σ. It is the central question in EIF SLAM: EIF have been applied to maps with hundreds of millions of features, and inverting a matrix of this size can be a challenge.

The answer to the complexity question depends on the topology of the world. If each feature is seen only once, the graph represented by the constraints in Ω is linear. Thus, Ω can be reordered so that it becomes a band-diagonal matrix, that is, all non-zero values occur near its diagonal. The standard variable elimination technique for matrix inversion will then invert Ω in linear time, by a single sweep through Ω. This intuition carries over to cycle-free world that is traversed once, so that each feature is seen for a short, consecutive period of time.

The more common case, however, involves features that are observed multiple times, with large time delays in between. This might be the case because the robot goes back and forth through a corridor, or because the world possesses cycles, and the robot traverses a full cycle. In either situation, there will exist features m_j that are seen at drastically different time steps x_{t_1} and x_{t_2}, with $t_2 \gg t_1$. In our constraint graph, this introduces a cyclic dependence: x_{t_1} and x_{t_2} are linked through the sequence of controls $u_{t_1+1}, u_{t_1+2}, \ldots, u_{t_2}$ and through the joint observation links between x_{t_1} and m_j, and x_{t_2} and m_j, respectively. Such links make our linear matrix inversion trick

inapplicable, and matrix inversion becomes more complex. Since most worlds possess cycles, this is the case of interest.

The EIF SLAM algorithm now employs an important factorization trick, which we can think of as propagating information trough the information matrix (in fact, it is a generalization of the well-known variable elimination algorithm for matrix inversion). Suppose we would like to remove a feature m_j from the information matrix Ω and the information state ξ. In our spring mass model, this is equivalent to removing the node and all springs attached to this node. As we shall see below, this is possible by a remarkably simple operation: We can remove all those springs between m_j and the poses at which m_j was observed, by introducing new springs between any pair of such poses. More formally, let $\tau(j)$ be the set of poses at which m_j was observed (that is: $x_t \in \tau(j) \iff \exists i : c_t^i = j$). Then we already know that the feature m_j is only linked to poses x_t in $\tau(j)$; by construction, m_j is *not* linked to any other pose, or to any landmark in the map. We can now set all links between m_j and the poses $\tau(j)$ to zero by introducing a new link between any two poses $x_t, x_{t'} \in \tau(j)$. Similarly, the information vector values for all poses $\tau(j)$ are also updated. An important characteristic of that this operation is local: It only involves only a small number of constraints. After removing all links to m_j, we can safely remove m_j from the information matrix and vector. The resulting information matrix is smaller—it lacks an entry for m_j. However, it is equivalent for the remaining variables, in the sense that the posterior defined by this information matrix is mathematically equivalent to the original posterior before removing m_j. This equivalence is intuitive: We simply have replaced springs connecting m_j to various poses in our spring mass model by a set of springs directly linking these poses. in doing so, the total force asserted by these spring remains equivalent, with the only exception that m_j is now disconnected.

Make Figure: reducing the graph by shifting edges around

The virtue of this reduction step is that we can gradually transform our optimization problem into a smaller one. By removing each feature m_j from Ω and ξ, we ultimately arrive at a much smaller information form $\tilde{\Omega}$ and $\tilde{\xi}$ defined only over the robot pose variables. The reduction can be carried out in time linear in the size of the map; in fact, it generalizes the variable elimination technique for matrix inversion to the information form, in which we also maintain an information state. The posterior over the robot path is now recovered as $\tilde{\Sigma} = \tilde{\Omega}^{-1}$ and $\tilde{\mu} = \tilde{\Sigma}\xi$. Unfortunately, our reduction step does not eliminate cycles in the posterior, so the remaining matrix inversion problem may still require more than linear time.

As a last step, EIF SLAM recovers the landmark locations. Conceptually, this is achieved by building a new information matrices Ω_j and information vectors ξ_j for

1: **Algorithm EIF_initialize($u_{1:t}$):**

2: $$\begin{pmatrix} \mu_{0,x} \\ \mu_{0,y} \\ \mu_{0,\theta} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

3: *for all controls* $u_t = (v_t \ \ \omega_t)^T$ *do*

4: $$\begin{pmatrix} \mu_{t,x} \\ \mu_{t,y} \\ \mu_{t,\theta} \end{pmatrix} = \begin{pmatrix} \mu_{t-1,x} \\ \mu_{t-1,y} \\ \mu_{t-1,\theta} \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \ \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \ \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix}$$

5: *endfor*

6: *return* $\mu_{0:t}$

Table 11.1 Initialization of the mean pose vector $\mu_{1:t}$ in the EIF SLAM algorithm.

each m_j. Both are defined over the variable m_j and the poses $\tau(j)$ at which m_j were observed. It contains the original links between m_j and $\tau(j)$, but the poses $\tau(j)$ are set to the values in $\tilde{\mu}$, without uncertainty. From this information form, it is now very simple to calculate the location of m_j, using the common matrix inversion trick. Clearly, Ω_j contains only elements that connect to m_j; hence the inversion takes time linear in the number of poses in $\tau(j)$.

It should be apparent why the information representation is such a natural representation. The full SLAM problem is solved by locally adding information into a large matrix, for each measurement z_t^i and each control u_t. To turn information into an estimate of the map and the robot path, information between poses and features is gradually shifted to information between pairs of poses. The resulting structure only constraints the robot poses, which are then calculated using matrix inversion. Once the poses are recovered, the feature locations are calculated one-after-another, based on the original feature-to-pose information.

11.3 THE EIF SLAM ALGORITHM

We will now make the various computational step of the EIF SLAM precise. The full EIF SLAM algorithm will be described in a number of steps. The main difficulty in implementing the simple additive information algorithm pertains to the conversion of

Algorithm EIF_construct($u_{1:t}, z_{1:t}, c_{1:t}, \mu_{0:t}$)**:**

set $\Omega = 0, \xi = 0$

add $\begin{pmatrix} \infty & 0 & 0 \\ 0 & \infty & 0 \\ 0 & 0 & \infty \end{pmatrix}$ *to* Ω *at* x_0

for all controls $u_t = (v_t \;\; \omega_t)^T$ *do*

$$\hat{x}_t = \mu_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \omega_t\Delta t \end{pmatrix}$$

$$G_t = \begin{pmatrix} 1 & 0 & \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 1 & \frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & 1 \end{pmatrix}$$

add $\begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1} \begin{pmatrix} 1 & -G_t \end{pmatrix}$ *to* Ω *at* x_t *and* x_{t-1}

add $\begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1} [\hat{x}_t + G_t \; \mu_{t-1}]$ *to* ξ *at* x_t *and* x_{t-1}

endfor

for all measurements z_t *do*

$$Q_t = \begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}$$

for all observed features $z_t^i = (r_t^i \;\; \phi_t^i \;\; s_t^i)^T$ *do*

$j = c_t^i$

$$\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \mu_{j,x} - \mu_{t,x} \\ \mu_{j,y} - \mu_{t,y} \end{pmatrix}$$

$q = \delta^T\delta$

$$\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \text{atan2}(\delta_y, \delta_x) - \mu_{t,\theta} \end{pmatrix}$$

$$H_t^i = \frac{1}{q}\begin{pmatrix} \sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & -\sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & \delta_x & -1 & -\delta_y & -\delta_x \end{pmatrix}$$

add $H_t^{iT}\, Q_t^{-1}\, H_t^i$ *to* Ω *at* x_t *and* m_j

add $H_t^{iT}\, Q_t^{-1}\, [\, z_t^i - \hat{z}_t^i - H_t^i \begin{pmatrix} \mu_{t,x} \\ \mu_{t,y} \\ \mu_{t,\theta} \\ \mu_{j,x} \\ \mu_{j,y} \end{pmatrix}]$ *to* ξ *at* x_t *and* m_j

endfor

endfor

return Ω, ξ

Table 11.2 Calculation of Ω and ξ in the EIF algorithm.

1: **Algorithm EIF_reduce(Ω, ξ):**
2: $\tilde{\Omega} = \Omega$
3: $\tilde{\xi} = \xi$
4: *for each feature* j *do*
5: *let* $\tau(j)$ *be the set of all poses* x_t *at which feature* j *was observed*
6: *subtract* $\tilde{\Omega}_{\tau(j),j}\ \tilde{\Omega}_{j,j}^{-1}\ \xi_j$ *from* $\tilde{\xi}$ *at* $x_{\tau(j)}$ *and* m_j
7: *subtract* $\tilde{\Omega}_{\tau(j),j}\ \tilde{\Omega}_{j,j}^{-1}\ \tilde{\Omega}_{j,\tau(j)}$ *from* $\tilde{\Omega}$ *at* $x_{\tau(j)}$ *and* m_j
8: *remove the rows and columns corresponding to feature* j *from* $\tilde{\Omega}$ *and* $\tilde{\xi}$
9: *endfor*
10: *return* $\tilde{\Omega}, \tilde{\xi}$

Table 11.3 Algorithm for reducing the size of the information representation of the posterior.

a conditional probability of the form $p(z_t^i \mid x_t, m)$ and $p(x_t \mid u_t, x_{t-1})$ into a link in the information matrix. The information matrix elements are all linear; hence this step involves linearizing $p(z_t^i \mid x_t, m)$ and $p(x_t \mid u_t, x_{t-1})$. In EKF SLAM, this linearization was found by calculating a Jacobian at the estimated mean poses $\mu_{0:t}$. To build our initial information matrix Ω and ξ, we need an initial estimate $\mu_{0:t}$ for all poses $x_{0:t}$.

There exist a number of solutions to the problem of finding an initial mean μ suitable for linearization. For example, we can run an EKF SLAM and use its estimate for linearization. For the sake of this chapter, we will use an even simpler technique: Our initial estimate will simply be provided by chaining together the motion model $p(x_t \mid u_t, x_{t-1})$. Such an algorithm is outlined in Table 11.1, and called there **EIF_initialize**. This algorithm takes the controls $u_{1:t}$ as input, and outputs sequence of pose estimates $\mu_{0:t}$. It initializes the first pose by zero, and then calculates subsequent poses by recursively applying the velocity motion model. Since we are only interested in the mean poses vector $\mu_{0:t}$, **EIF_initialize** only use the deterministic part of the motion model. It also does not consider any measurement in its estimation.

Once an initial $\mu_{0:t}$ is available, the EIF SLAM algorithm constructs the full SLAM information matrix Ω and the corresponding information vector ξ. This is achieved by the algorithm **EIF_construct**, depicted in Table 11.2. This algorithm contains a good amount of mathematical notation, much of which shall become clear in our mathematical derivation of the algorithm further below. **EIF_construct** accepts as an input the

1: **Algorithm EIF_solve($\tilde{\Omega}, \tilde{\xi}, \Omega, \xi$):**
2: $\Sigma_{0:t} = \tilde{\Omega}^{-1}$
3: $\mu_{0:t} = \Sigma_{0:t}\ \tilde{\xi}$
4: *for each feature* j *do*
5: *let* $\tau(j)$ *be the set of all poses* x_t *at which feature* j *was observed*
6: $\mu_j = \Omega_{j,j}^{-1}\ (\xi_j + \Omega_{j,\tau(j)}\ \tilde{\mu}_{\tau(j)})$
7: *endfor*
8: *return* $\mu, \Sigma_{0:t}$

Table 11.4 Algorithm for updating the posterior μ.

set of controls, $u_{1:t}$, the measurements $z_{1:t}$ and associated correspondence variables $c_{1:t}$, and the mean pose estimates $\mu_{0:t}$. It then gradually constructs the information matrix Ω and the information vector ξ by locally adding submatrices in accordance with the information obtained from each measurement and each control. In particular, Line 2 in **EIF_construct** initializes the information elements. The "infinite" information entry in Line 3 fixes the initial pose x_0 to $(0\ \ 0\ \ 0)^T$. It is necessary, since otherwise the resulting matrix becomes singular, reflecting the fact that from relative information alone we cannot recover absolute estimates.

Controls are integrated in Lines 4 through 9 of **EIF_construct**. The pose $\hat{x}$ and the Jacobian G_t calculated in Lines 5 and 6 represent the linear approximation of the nonlinear measurement function g. As obvious from these equations, this linearization step utilizes the pose estimates $\mu_{0:t-1}$, with $\mu_0 = (0\ \ 0\ \ 0)^T$. This leads to the updates for Ω, and ξ, calculated in Lines 7, and 8, respectively. Both terms are added into the corresponding rows and columns of Ω and ξ. This addition realizes the inclusion of a new constraint into the SLAM posterior, very much along the lines of the intuitive description in the previous section.

Measurements are integrated in Lines 10 through 21 of **EIF_construct**. The matrix Q_t calculated in Line 11 is the familiar measurement noise covariance. Lines 13 through 17 compute the Taylor expansion of the measurement function, here stated for the feature-based measurement model defined in Chapter 6.6. This calculation culminates into the computation of the measurement update in Lines 18 and 19. The matrix that is being address to Ω in Line 18 is of dimension 5×5. To add it, we decomposes it into a matrix of dimension 3×3 for the pose x_t, a matrix of dimension 2×2 for the feature m_j, and two matrices of dimension 3×2 and 2×3 for the link between x_t and m_j.

Those are added to Ω at the corresponding rows and columns. Similarly, the vector added to the information vector ξ is of vertical dimension 5. It is also chopped into two vectors of size 3 and 2, and added to the elements corresponding to x_t and m_j, respectively. The result of **EIF_construct** is an information vector ξ and a matrix Ω. Ω is sparse, containing only non-zero submatrices along the main diagonal, between subsequent poses, and between poses and features in the map. If the number map features outnumber the number of poses by a factor of 10, some 99% of the elements of Ω are zero. The running time of this algorithm is linear in t, the number of time steps at which data was accrued.

The next step of the EIF SLAM algorithm pertains to reducing the dimensionality of the information matrix/vector. This is achieved through the algorithm **EIF_reduce** in Table 11.3. This algorithm takes as input Ω and ξ defined over the full space of map features and poses, and outputs a reduced matrix $\tilde{\Omega}$ and vectors $\tilde{\xi}$ defined over the space of all poses. This transformation is achieved by removing features m_j one-at-a-time, in Lines 4 through 9 of **EIF_reduce**. The book keeping of the exact indexes of each item in $\tilde{\Omega}$ and $\tilde{\xi}$ is a bit tedious, hence Table 11.3 only provides an intuitive account. Line 5 calculates the set of poses $\tau(j)$ at which the robot observed feature j. It then extracts two submatrices from the present $\tilde{\Omega}$: $\tilde{\Omega}_{j,j}$, which is the quadratic submatrix between m_j and m_j, and $\tilde{\Omega}_{\tau(j),j}$, which is composed of the off-diagonal elements between m_j and the pose variables $\tau(j)$. It also extracts from the information state vector $\tilde{\xi}$ the elements corresponding to the j-th feature, denoted here as ξ_j. It then subtracts information from $\tilde{\Omega}$ and $\tilde{\xi}$ as stated in Lines 6 and 7. After this operation, the rows and columns for the feature m_j are zero. These rows and columns are then removed, reducing the dimension on $\tilde{\Omega}$ and $\tilde{\xi}$ accordingly. This process is iterated until all features have been removed, and only pose variables remain in $\tilde{\Omega}$ and $\tilde{\xi}$. The complexity of **EIF_reduce** is once again linear in t.

The last step in the EIF SLAM algorithm computes the mean and covariance for all poses in the robot path, and a mean location estimate for all features in the map. This is achieved through **EIF_solve** in Table 11.4. Lines 2 and 3 compute the path estimates $\mu_{0:t}$, by inverting thee reduced information matrix $\tilde{\Omega}$ and multiplying the resulting covariance with the information vector. When $\tilde{\Omega}$ involves cycles, the inversion becomes the computationally most expensive step in the EIF SLAM algorithm. Subsequently, **EIF_solve** computes the location of each feature in Lines 4 through 7. The return value of **EIF_solve** contains the mean for the robot path and all features in the map, but only the covariance for the robot path.

The quality of the solution calculated by the EIF SLAM algorithm depends on the goodness of the initial mean estimates, calculated by **EIF_initialize**. The x- and y-components of these estimates effect the respective models in a linear way, hence the

1: **Algorithm EIF_SLAM_known_correspondence($u_{1:t}, z_{1:t}, c_{1:t}$):**
2: $\mu_{0:t} = \textbf{EIF\_initialize}(u_{1:t})$
3: *repeat*
4: $\Omega, \xi = \textbf{EIF\_construct}(u_{1:t}, z_{1:t}, c_{1:t}, \mu_{0:t})$
5: $\tilde{\Omega}, \tilde{\xi} = \textbf{EIF\_reduce}(\Omega, \xi)$
6: $\mu, \Sigma_{0:t} = \textbf{EIF\_solve}(\tilde{\Omega}, \tilde{\xi}, \Omega, \xi)$
7: *until convergence*
8: *return* μ

Table 11.5 The EIF SLAM algorithm for the full SLAM problem with known correspondence.

linearization does not depend on they value. Not so for the orientation variables in $\mu_{0:t}$. Errors in these initial estimates affect the accuracy of the Taylor approximation, which in turn affects the result.

To accommodate errors in the initial $\mu_{0:t}$, the procedures **EIF_construct**, **EIF_reduce**, and **EIF_solve** are run multiple times over the same data set. Each iteration takes as an input a estimated mean vector $\mu_{0:t}$ for all poses, and outputs a new, improves estimate. The iteration of the EIF SLAM optimization are only necessary when the initial pose estimates have high error (e.g., more than 20 degrees orientation error). A small number of iterations (e.g., 3) is usually sufficient.

Table 11.5 summarizes the resulting algorithm. It initializes the means, then repeats the construction step, the reduction step, and the solution step. Typically, two or three iterations suffice for convergence. The resulting mean μ comprises the best estimates of the robot's path and the map.

11.4 MATHEMATICAL DERIVATION

The derivation of the EIF SLAM algorithm begins with a derivation of a recursive formula for calculating the full SLAM posterior, represented in information form. We then investigate each term in this posterior, and derive from them the additive SLAM updates through Taylor expansions. From that, we will derive the necessary equations for recovering the path and the map.

11.4.1 The Full SLAM Posterior

As in the discussion of EKF SLAM, it will be beneficial to introduce a variable for the augmented state of the full SLAM problem. We will use y to denote state variables that combine one or more poses x with the map m. In particular, we define $y_{0:t}$ to be a vector composed of the path $x_{0:t}$ and the map m, whereas y_t is composed of the momentary pose at time t and the map m:

$$y_{0:t} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_t \\ m \end{pmatrix} \quad \text{and} \quad y_t = \begin{pmatrix} x_t \\ m \end{pmatrix} \tag{11.1}$$

The posterior in the full SLAM problem is $p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t})$, where $z_{1:t}$ are the familiar measurements with correspondences $c_{1:t}$, and $u_{1:t}$ are the controls. Bayes rule enables us to factor this posterior:

$$\begin{aligned} & p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ &= \eta\, p(z_t \mid y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t})\, p(y_{0:t} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) \end{aligned} \tag{11.2}$$

where η is the familiar normalizer. The first probability on the right-hand side can be reduced by dropping irrelevant conditioning variables:

$$p(z_t \mid y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t}) = p(z_t \mid y_t, c_t) \tag{11.3}$$

Similarly, we can factor the second probability by partitioning $y_{0:t}$ into x_t and $y_{0:t-1}$, and obtain

$$\begin{aligned} & p(y_{0:t} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) \\ &= p(x_t \mid y_{1:t-1}, z_{1:t-1}, u_{1:t}, c_{1:t})\, p(y_{1:t-1} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) \\ &= p(x_t \mid x_{t-1}, u_t)\, p(y_{1:t-1} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) \end{aligned} \tag{11.4}$$

Putting these expressions back into (11.2) gives us the recursive definition of the full SLAM posterior:

$$p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t})$$

$$= \eta\, p(z_t \mid y_t, c_t)\, p(x_t \mid x_{t-1}, u_t)\, p(y_{1:t-1} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) \quad (11.5)$$

The closed form expression is obtained through induction over t. Here $p(y_0)$ is the prior over the map m and the initial pose x_0.

$$\begin{aligned} p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) &= \eta\, p(y_0) \prod_t p(z_t \mid y_t, c_t)\, p(x_t \mid x_{t-1}, u_t) \quad (11.6) \\ &= \eta\, p(y_0) \prod_t \left[p(x_t \mid x_{t-1}, u_t) \prod_i p(z_t^i \mid y_t, c_t^i) \right] \end{aligned}$$

Here, as before, z_t^i is the i-th measurement in the measurement vector z_t at time t. The prior $p(y_0)$ factors into two independent priors, $p(x_0)$ and $p(m)$. In SLAM, we have no initial knowledge about the map m. This allows us to replace $p(y_0)$ by $p(x_0)$, subsuming the factor $p(m)$ into the normalizer η.

The information from represents probabilities in logarithmic form. The log-SLAM posterior follows directly from the previous equation:

$$\begin{aligned} &\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \quad (11.7) \\ &= \text{const.} + \log p(x_0) \sum_t \left[\log p(x_t \mid x_{t-1}, u_t) \sum_i \log p(z_t^i \mid y_t, c_t^i) \right] \end{aligned}$$

This posterior has a simple form: It is a sum of terms. These terms include a prior, one term for each control u_t, and one term for each measurement z_t^i. As we shall see, EIF SLAM simply replaces those additive terms with quadratic constraints.

11.4.2 Taylor Expansion

The key approximation of EIF SLAM is the same as in the EKF: The motion and measurement models are approximated using linear functions with Gaussian error distributions. Just as in Chapter 10, we assume the outcome of robot motion is distributed normally according to $\mathcal{N}(g(u_t, x_{t-1}), R_t)$, where g is the deterministic motion function, and R_t is the covariance of the motion error. Similarly, measurements z_t^i are generated according to $\mathcal{N}(h(y_t, c_t^i), Q_t)$, where h is the familiar measurement function and Q_t is the measurement error covariance. In equations, we have

$$p(x_t \mid x_{t-1}, u_t) = \eta\, \exp\left\{ -\tfrac{1}{2} (x_t - g(u_t, x_{t-1}))^T\, R_t^{-1}\, (x_t - g(u_t, x_{t-1})) \right\}$$

$$p(z_t^i \mid y_t, c_t^i) \quad = \quad \eta \, \exp\left\{-\tfrac{1}{2}(z_t^i - h(y_t, c_t^i))^T \, Q_t^{-1} \, (z_t^i - h(y_t, c_t^i))\right\} \tag{11.8}$$

The prior $p(x_0)$ in (11.7) is also easily expressed by a Gaussian-type distribution. This prior anchors the initial pose x_0 to the origin of the global coordinate system: $x_0 = (0 \;\; 0 \;\; 0)^T$:

$$p(x_0) \quad = \quad \eta \, \exp\left\{-\tfrac{1}{2} \, x_0^T \, \Omega_0 \, x_0\right\} \tag{11.9}$$

with

$$\Omega_0 \quad = \quad \begin{pmatrix} \infty & 0 & 0 \\ 0 & \infty & 0 \\ 0 & 0 & \infty \end{pmatrix} \tag{11.10}$$

It shall, at this point, not concern us that the value of ∞ cannot be implemented, as we can easily substitute ∞ with a large positive number. This leads to the following quadratic form of the log-SLAM posterior in (11.7):

$$\begin{aligned} &\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ &\quad = \quad \text{const.} - \tfrac{1}{2} \Bigg[x_0^T \, \Omega_0 \, x_0 + \sum_t (x_t - g(u_t, x_{t-1}))^T \, R_t^{-1} \, (x_t - g(u_t, x_{t-1})) \\ &\qquad + \sum_i (z_t^i - h(y_t, c_t^i))^T \, Q_t^{-1} \, (z_t^i - h(y_t, c_t^i)) \Bigg] \end{aligned} \tag{11.11}$$

This representation highlights an essential characteristic of the full SLAM posterior in the information form: It is composed of a number of quadratic terms, one for the prior, and one for each control and each measurement.

However, these terms are quadratic in the functions g and h, not in the variables we week to estimate (poses and the map). This is alleviated by linearizing g and h via Taylor expansion, analogously to Equations (10.15) and (10.19) in the derivation of the EKF:

$$\begin{aligned} g(u_t, x_{t-1}) &\approx g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \\ h(y_t, c_t^i) &\approx h(\mu_t, c_t^i) + H_t^i \, (y_t - \mu_t) \end{aligned} \tag{11.12}$$

Where μ_t is the current estimate of the state vector y_t, and $H_t^i = h_t^i \; F_{x,j}$ as defined already in Equation (10.20).

11.4.3 Constructing the Information Form

This linear approximation turns the log-likelihood (11.11) into a function that is quadratic in $y_{0:t}$. In particular, we obtain

$$\begin{aligned}\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \;\; &= \;\; \text{const.} - \tfrac{1}{2} \\ &\Big\{ x_0^T \; \Omega_0 \; x_0 + \sum_t [x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]^T \\ &\qquad\qquad R_t^{-1} \; [x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})] \\ &+ \sum_i [z_t^i - h(\mu_t, c_t^i) - H_t^i(y_t - \mu_t)]^T \; Q_t^{-1} \; [z_t^i - h(\mu_t, c_t^i) - H_t^i(y_t - \mu_t)] \Big\}\end{aligned} \tag{11.13}$$

This function is indeed a quadratic, but it shall prove convenient to reorder its terms.

$$\begin{aligned}\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \;\; &= \;\; \text{const.} \\ &-\tfrac{1}{2} \underbrace{x_0^T \; \Omega_0 \; x_0}_{\text{quadratic in } x_0} - \tfrac{1}{2} \sum_t \underbrace{x_{t-1:t}^T \begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1} \; (1 \;\; -G_t) \; x_{t-1:t}}_{\text{quadratic in } x_{t-1:t}} \\ &+ \underbrace{x_{t-1:t}^T \begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1} \; [g(u_t, \mu_{t-1}) + G_t \; \mu_{t-1}]}_{\text{linear in } x_{t-1:t}} \\ &-\tfrac{1}{2} \sum_i \underbrace{y_t^T \; H_t^{iT} \; Q_t^{-1} \; H_t^i \; y_t}_{\text{quadratic in } y_t} + \underbrace{y_t^T \; H_t^{iT} \; Q_t^{-1} \; [z_t^i - h(\mu_t, c_t^i) - H_t^i \mu_t]}_{\text{linear in } y_t}\end{aligned} \tag{11.14}$$

Here $x_{t-1:t}$ denotes the vector concatenating x_{t-1} and x_t; hence we can write $(x_t - G_t \; x_{t-1})^T = x_{t-1:t}^T \; (1 \;\; -G_t)$.

If we collect all quadratic terms into the matrix Ω, and all linear terms into a vector ξ, we see that expression (11.14) is of the form

$$\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \;\; = \;\; \text{const.} - \tfrac{1}{2} \; y_{0:t}^T \; \Omega \; y_{0:t} + y_{0:t}^T \; \xi \tag{11.15}$$

We can read off these terms directly from (11.14), and verify that they are indeed implemented in the algorithm **EIF_construct** in Table 11.2:

- **Prior.** The initial pose prior manifests itself by a quadratic term Ω_0 over the initial pose variable x_0 in the information matrix. Assuming appropriate extension of the matrix Ω_0 to match the dimension of $y_{0:t}$, we have

$$\Omega \longleftarrow \Omega_0 \tag{11.16}$$

  This initialization is performed in Lines 2 and 3 of the algorithm **EIF_construct**. **EIF_construct**.

- **Controls.** From (11.14), we see that each control u_t adds to Ω and ξ the following terms, assuming that the matrices are rearranged so as to be of matching dimensions:

$$\Omega \longleftarrow \Omega + \begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1} \begin{pmatrix} 1 & -G_t \end{pmatrix} \tag{11.17}$$

$$\xi \longleftarrow \xi + \begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1} \left[g(u_t, \mu_{t-1}) + G_t \, \mu_{t-1} \right] \tag{11.18}$$

  This is realized in Lines 4 through 9 in **EIF_construct**.

- **Measurements.** According to Equation (11.14), each measurement z_t^i transforms Ω and ξ by adding the following terms, once again assuming appropriate adjustment of the matrix dimensions:

$$\Omega \longleftarrow \Omega + H_t^{iT} \, Q_t^{-1} \, H_t^i \tag{11.19}$$

$$\xi \longleftarrow \xi + H_t^{iT} \, Q_t^{-1} \left[z_t^i - h(\mu_t, c_t^i) - H_t^i \mu_t \right] \tag{11.20}$$

  This update occurs in Lines 10 through 21 in **EIF_construct**.

This proves the correctness of the construction algorithm **EIF_construct**, relative to our linear Taylor expansion approximation.

We also note that the steps above only affect elements on the (generalized) main diagonal of the matrix, or elements that involve at least one pose. Thus, all between-feature elements are zero in the resulting information matrix.

Marginals of a multivariate Gaussian. Let the probability distribution $p(x, y)$ over the random vectors x and y be a Gaussian represented in the information form

$$\Omega = \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \qquad \text{and} \qquad \xi = \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} \tag{11.21}$$

If Ω_{yy} is invertible, the marginal $p(x)$ is a Gaussian whose information representation is

$$\bar{\Omega}_{xx} = \Omega_{xx} - \Omega_{xy}\,\Omega_{yy}^{-1}\,\Omega_{yx} \qquad \text{and} \qquad \bar{\xi}_x = \xi_x - \Omega_{xy}\,\Omega_{yy}^{-1}\,\xi_y \tag{11.22}$$

Proof. The marginal for a Gaussian in its moments representation

$$\Sigma = \begin{pmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{pmatrix} \qquad \text{and} \qquad \mu = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}$$

is $\mathcal{N}(\mu_x, \Sigma_{xx})$. By definition, the information matrix of this Gaussian is therefore Σ_{xx}^{-1}, and the information vector is $\Sigma_{xx}^{-1}\,\mu_x$. We show $\Sigma_{xx}^{-1} = \bar{\Omega}_{xx}$ via the Inversion Lemma from Table 3.2 on page 44. Let $P = (0\ \ 1)^T$, and let $[\infty]$ be a matrix of the same size as Ω_{yy} but whose entries are all infinite (and with $[\infty]^{-1} = 0$. This gives us

$$(\Omega + P[\infty]P^T)^{-1} = \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & [\infty] \end{pmatrix}^{-1} \overset{(*)}{=} \begin{pmatrix} \Sigma_{xx}^{-1} & 0 \\ 0 & 0 \end{pmatrix}$$

The same expression can also be expanded by the inversion lemma into:

$$\begin{aligned}
&(\Omega + P[\infty]P^T)^{-1} \\
&\quad = \Omega - \Omega\,P([\infty]^{-1} + P^T\,\Omega\,P)^{-1}\,P^T\,\Omega \\
&\quad = \Omega - \Omega\,P(0 + P^T\,\Omega\,P)^{-1}\,P^T\,\Omega \\
&\quad = \Omega - \Omega\,P(\Omega_{yy})^{-1}\,P^T\,\Omega \\
&\quad = \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} - \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & \Omega_{yy}^{-1} \end{pmatrix} \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \\
&\quad \overset{(*)}{=} \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} - \begin{pmatrix} 0 & \Omega_{xy}\,\Omega_{yy}^{-1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \\
&\quad = \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} - \begin{pmatrix} \Omega_{xy}\,\Omega_{yy}^{-1}\,\Omega_{yx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} = \begin{pmatrix} \bar{\Omega}_{xx} & 0 \\ 0 & 0 \end{pmatrix}
\end{aligned}$$

The remaining statement, $\Sigma_{xx}^{-1}\,\mu_x = \bar{\xi}_x$, is obtained analogously, exploiting the fact that $\mu = \Omega^{-1}\xi$ (see Equation(3.72)) and the equality of the two expressions marked "$(*)$"above:

$$\begin{aligned}
\begin{pmatrix} \Sigma_{xx}^{-1}\,\mu_x \\ 0 \end{pmatrix} &= \begin{pmatrix} \Sigma_{xx}^{-1} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix} = \begin{pmatrix} \Sigma_{xx}^{-1} & 0 \\ 0 & 0 \end{pmatrix} \Omega^{-1} \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} \\
&\overset{(*)}{=} \left[\Omega - \begin{pmatrix} 0 & \Omega_{xy}\,\Omega_{yy}^{-1} \\ 0 & 1 \end{pmatrix} \Omega\right] \Omega^{-1} \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} \\
&= \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} - \begin{pmatrix} 0 & \Omega_{xy}\,\Omega_{yy}^{-1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} = \begin{pmatrix} \bar{\xi}_x \\ 0 \end{pmatrix}
\end{aligned}$$

Table 11.6 Lemma for marginalizing Gaussians in information form.

Conditionals of a multivariate Gaussian. Let the probability distribution $p(x, y)$ over the random vectors x and y be a Gaussian represented in the information form

$$\Omega = \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \qquad \text{and} \qquad \xi = \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} \tag{11.23}$$

The conditional $p(x \mid y)$ is a Gaussian with information matrix Ω_{xx} and information vector $\xi_x + \Omega_{xy}\, y$.

Proof. The result follows trivially from the definition of a Gaussian in information form:

$$\begin{aligned} & p(x \mid y) \\ &= \eta\ \exp\left\{-\tfrac{1}{2}\begin{pmatrix} x \\ y \end{pmatrix}^T \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} x \\ y \end{pmatrix}^T \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix}\right\} \\ &= \eta\ \exp\left\{-\tfrac{1}{2}x^T\Omega_{xx}x + x^T\Omega_{xy}y - \tfrac{1}{2} + y^T\Omega_{yy}y + x^T\xi_x + y^T\xi_y\right\} \\ &= \eta\ \exp\{-\tfrac{1}{2}x^T\Omega_{xx}x + x^T(\Omega_{xy}y + \xi_x) \underbrace{-\tfrac{1}{2} + y^T\Omega_{yy}y + y^T\xi_y}_{\text{const.}}\} \\ &= \eta\ \exp\{-\tfrac{1}{2}x^T\Omega_{xx}x + x^T(\Omega_{xy}y + \xi_x)\} \end{aligned} \tag{11.24}$$

Table 11.7 Lemma for marginalizing Gaussians in information form.

11.4.4 Reducing the Information Form

The reduction step **EIF_reduce** is based on a factorization of the full SLAM posterior.

$$\begin{aligned} & p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ &= p(x_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t})\ p(m \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}) \end{aligned} \tag{11.25}$$

Here $p(x_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \sim \mathcal{N}(\Omega, \xi)$ is the posterior over paths alone, with the map integrated out:

$$p(x_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = \int p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t})\ dm \tag{11.26}$$

As we will show shortly, this probability is caculated in **EIF_reduce** in Table 11.3:

$$p(x_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \sim \mathcal{N}(\tilde{\xi}, \tilde{\Omega}) \tag{11.27}$$

In general, such an integration will be intractable, due to the large number of variables in m. For Gaussians, this integral can be calculates in closed form. The key insight is given in Table 11.6, which states and proves the Marginalization Lemma for Gaussians.

Let us subdivide the matrix Ω and the vector ξ into submatrices, for the robot path $x_{0:t}$ and the map m:

$$\Omega = \begin{pmatrix} \Omega_{x_{0:t},x_{0:t}} & \Omega_{x_{0:t},m} \\ \Omega_{m,x_{0:t}} & \Omega_{m,m} \end{pmatrix} \tag{11.28}$$

$$\xi = \begin{pmatrix} \xi_{x_{0:t}} \\ \xi_m \end{pmatrix} \tag{11.29}$$

The probability (11.27) is then obtained as

$$\tilde{\Omega} = \Omega_{x_{0:t},x_{0:t}} - \Omega_{x_{0:t},m}\, \Omega_{m,m}^{-1}\, \Omega_{m,x_{0:t}} \tag{11.30}$$

$$\tilde{\xi} = \xi_{x_{0:t}} - \Omega_{x_{0:t},m}\, \Omega_{m,m}^{-1}\, \xi_m \tag{11.31}$$

The matrix $\Omega_{m,m}$ is block-diagonal. This follows from the way Ω is constructed, in particular the absence of any links between pairs of landmarks. This makes the inversion efficient:

$$\Omega_{m,m}^{-1} = \sum_j F_j^T\, \Omega_{j,j}^{-1}\, F_j \tag{11.32}$$

where $\Omega_{j,j} = F_j \Omega F_j^T$ is the sub-matrix of Ω that corresponds to the j-th feature in the map, that is

$$F_j = \begin{pmatrix} 0 \cdots 0 & 1\ 0\ 0 & 0 \cdots 0 \\ 0 \cdots 0 & 0\ 1\ 0 & 0 \cdots 0 \\ 0 \cdots 0 & \underbrace{0\ 0\ 1}_{j-\text{th feature}} & 0 \cdots 0 \end{pmatrix} \tag{11.33}$$

This makes it possible to to decompose the implement Equations (11.30) and (11.31) into a sequential update:

$$\tilde{\Omega} = \Omega_{x_{0:t},x_{0:t}} - \sum_j \Omega_{x_{0:t},j}\, \Omega_{j,j}^{-1}\, \Omega_{j,x_{0:t}} \tag{11.34}$$

$$\tilde{\xi} \quad = \quad \xi_{x_{0:t}} - \sum_j \Omega_{x_{0:t},j} \; \Omega_{j,j}^{-1} \; \xi_j \tag{11.35}$$

The matrix $\Omega_{x_{0:t},j}$ is non-zero only for elements in $\tau(j)$, the set of poses at which landmark j was observed. This essentially proves the correctness of the reduction algorithm **EIF_reduce** in Table 11.3. The operation performed on Ω in this algorithm implments an incomplete variable elimination algorithm for matrix inversion, applied to the landmark variables.

11.4.5 Recovering the Path and the Map

The algorithm **EIF_solve** in Table 11.4 calculates the mean and variance of the Gaussian $\mathcal{N}(\tilde{\xi}, \tilde{\Omega})$, using the standard equations, see Equations (3.67) and (3.72) on page 55:

$$\tilde{\Sigma} \quad = \quad \tilde{\Omega}^{-1} \tag{11.36}$$

$$\tilde{\mu} \quad = \quad \tilde{\Sigma} \, \tilde{\xi} \tag{11.37}$$

In particular, this operation provides us with the mean of the posterior on the robot path; it does not give us the locations of the features in the map.

It remains to recover the second factor of Equation (11.25):

$$p(m \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}) \tag{11.38}$$

The conditioning lemma, stated and proved in Table 11.7, shows that this probability distribution is Gaussian with the parameters

$$\Sigma_m \quad = \quad \Omega_{m,m}^{-1} \tag{11.39}$$

$$\mu_m \quad = \quad \Sigma_m(\xi_m + \Omega_{m,x_{0:t}}\tilde{\xi}) \tag{11.40}$$

Here ξ_m and $\Omega_{m,m}$ are the subvector of ξ, and the submatrix of Ω, respectively, restricted to the map variables. The matrix $\Omega_{m,x_{0:t}}$ is the off-diagonal submatrix of Ω that connects the robot path to the map. As noted before, $\Omega_{m,m}$ is block-diagonal,

hence we can decompose

$$p(m \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}) = \prod_j p(m_j \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}) \tag{11.41}$$

where each $p(m_j \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t})$ is distributed according to

$$\Sigma_j = \Omega_{j,j}^{-1} \tag{11.42}$$

$$\mu_j = \Sigma_j(\xi_j + \Omega_{j,x_{0:t}}\tilde{\mu}) = \Sigma_j(\xi_j + \Omega_{j,\tau(j)}\tilde{\mu}_{\tau(j)}) \tag{11.43}$$

he last transformation exploited the fact that the submatrix $\Omega_{j,x_{0:t}}$ is zero except for those pose variables $\tau(j)$ from which the j-th feature was observed.

It is important to notice that this is a Gaussian $p(m \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t})$ conditioned on the true path $x_{0:t}$. In practice, we do not know the path, hence one might want to know the posterior $p(m \mid z_{1:t}, u_{1:t}, c_{1:t})$ without the path in the conditioning set. This Gaussian cannot be factored in the moments representation, as locations of different features are correlated through the uncertainty over the robot pose. For this reason, **EIF_solve** returns the mean estimate of the posterior but only the covariance over the robot path. Luckily, we never need the full Gaussian in moments form—which would involve a fully populated covariance matrix of massive dimensions—- as all essential questions pertaining to the SLAM problem can be answered at least in approximation without knowledge of Σ.

11.5 DATA ASSOCIATION IN THE EIF

Data association in EIFs is realized through correspdoncence variables, just as in EKF SLAM. Just like EKF SLAM, the EIF searches for a single best correspondence vector, instead of calculating an entire distribution over correspondences. Thus, finding a correspondence vector is a search problem. However, correspondences in EIF SLAM are defined slightly differentely: they are defined over pairs of features in the map, rather than associations of measurements to features. Specifrically, we say $c(j,k) = 1$ iff m_j and m_k correspond to the same physical feature in the world. Otherwise, $c(j,k) = 0$. This feature-correspondence is in fact logically equivalent to the correspondence defined in the previous section, but it simplifies the statement of the basic algorithm.

Our technique for searching the space of correspondences is greedy, just as in the EKF. Each step in the search of the best correspondence value leads to an improvement, as

measured by the appropriate log-likelihood function. However, because EIFs consider all data at the same time, it is possible to devise correspondence techniques that are considerably more powerful than the incremental approach in the EKF. In particular:

1. At any point in the search, EIFs can consider the correspondence of any set of landmarks. There is no requirement to process the observed landmarks sequentially.
2. Correspondence search can be interleaved with the calculation of the map. Assuming that two observed landmarks correspond to the same physical landmark in the world affects the resulting map. By incorporating such a correspondence hypothesis into the map, other correspondence hypotheses will subsequently look more or less likely.
3. Data association decisions in EIFs can be *undone*. The goodness of a data association depends on the value of other data association decisions. What once appears to be a good choice may, at some later time in the search

In the remainder of this chapter, we will describe one specific correspondence search algorithms that exploits the first two properties (but not the third). Our data associaiton algorithm will still be greedy, and it will sequentially seaerch the space of possible correspondences to arrive at a plausibale map. However, like all greedy algorithms, our approach is subject to local maxima; the true space of correspondences is of course exponential in the number of features in the map, and searching through all of them is infeasible in most environments. Hence, we will be content with a hill climbing algorithm.

11.5.1 The EIF SLAM Algorithm With Unknown Correspondence

The key component of our algorithm is a likelihood test for correspondence. Specifially, EIF correspondence is based on a simple test: What is the probability that two different features in the map, m_j and m_k, correspond to the same physical feature in the world? If this probability exceeds a threshold, we will accept this hypothesis and merge both features in the map.

The correspond test is depicted in Table 11.8: The input to the test are two feature indexes, j and k, for which we seek to compute the probability that those two feagtures corespond to the same feature in the physical world. To calculate this probability,

1: **Algorithm EIF_correspondence_test($\Omega, \xi, \mu, \Sigma_{0:t}, j, k$):**

2: $\Omega_{[j,k]} = \Omega_{jk,jk} - \Omega_{jk,\tau(j,k)} \; \Sigma_{\tau(j,k),\tau(j,k)} \; \Omega_{\tau(j,k),jk}$

3: $\xi_{[j,k]} = \Omega_{[j,k]} \; \mu_{j,k}$

4: $\Omega_{\Delta j,k} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}^T \Omega_{[j,k]} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

5: $\xi_{\Delta j,k} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}^T \xi_{[j,k]}$

6: $\mu_{\Delta j,k} = \Omega^{-1}_{\Delta j,k} \; \xi_{\Delta j,k}$

7: *return* $|2\pi \; \Omega^{-1}_{\Delta j,k}|^{-\frac{1}{2}} \; \exp\left\{-\frac{1}{2} \; \mu^T_{\Delta j,k} \; \Omega^{-1}_{\Delta j,k} \; \mu_{\Delta j,k}\right\}$

Table 11.8 The EIF SLAM test for correspondence: It accepts as input an information representation of the SLAM posterior, along with the result of the **EIF_solve** step. It then outputs the posterior probability that m_j corresponds to m_k.

our algorithm utilizes a number of quantities: The information representation of the SLAM posterior, as manifest by Ω and ξ, and the result of the procedure **EIF_solve**, which is the mean vector μ and the path covariance $\Sigma_{0:t}$.

The correspondence test then proceeds in the following way: First, it computes the marginalized posterir over the two target features. This posterior is represented by the information matrix $\Omega_{[j,k]}$ and vector $\xi_{[j,k]}$ computed in Lines 2 and 3 in Table 11.8. This step of the computation utilizes various sub-elements of the information form Ω, ξ, the mean feature locations as specifcied through μ, and the path covariance $\Sigma_{0:t}$. Next, it calculates the parameters of a new Gaussian random variable, whose value is the difference between m_j and m_k. Denoting the difference variable $\Delta_{j,k} = m_j - m_k$, the informaiton parameters $\Omega_{\Delta j,k}, \xi_{\Delta j,k}$ are calculated in Lines 4 and 5, and the corresponding expectation for the difference is computed in Line 6. Line 7 return the desired probability: the probability that the difference between m_j and m_k is 0.

The correspondence test provides us now with an algorithm for performing data association search in EIF SLAM. Table 11.9 shows such an algorithm. It initializes the correspondence variables with unique values. The four steps that follow (Lines 3-7) are the same as in our EIF SLAM algorithm with known correspondence, stated in Table 11.5. However, this general SLAM algorithm then engages in the data association search. Specifically, for each pair of different features in the map, it calculates

1: **Algorithm EIF_SLAM($u_{1:t}, z_{1:t}$):**
2: *initialze all* c_t^i *with a unique value*
3: $\mu_{0:t} = \textbf{EIF\_initialize}(u_{1:t})$
4: $\Omega, \xi = \textbf{EIF\_construct}(u_{1:t}, z_{1:t}, c_{1:t}, \mu_{0:t})$
5: $\tilde{\Omega}, \tilde{\xi} = \textbf{EIF\_reduce}(\Omega, \xi)$
6: $\mu, \Sigma_{0:t} = \textbf{EIF\_solve}(\tilde{\Omega}, \tilde{\xi}, \Omega, \xi)$
7: *repeat*
8: *for each pair of non-corresponding features* m_j, m_k *do*
9: *calculate* $\pi_{j=k} = \textbf{EIF\_correspondence\_test}(\Omega, \xi, \mu, \Sigma_{0:t}, j, k)$
10: *if* $\pi_{j=k} > \chi$ *then*
11: *for all* $c_t^i = k$ *set* $c_t^i = j$
12: $\Omega, \xi = \textbf{EIF\_construct}(u_{1:t}, z_{1:t}, c_{1:t}, \mu_{0:t})$
13: $\tilde{\Omega}, \tilde{\xi} = \textbf{EIF\_reduce}(\Omega, \xi)$
14: $\mu, \Sigma_{0:t} = \textbf{EIF\_solve}(\tilde{\Omega}, \tilde{\xi}, \Omega, \xi)$
15: *endif*
16: *endfor*
17: *until no more pair* m_j, m_k *found with* $\pi_{j=k} < \chi$
18: *return* μ

Table 11.9 The EIF SLAM algorithm for the full SLAM problem with unknown correspondence. The inner loop of this version can be made more effi cient by selective probing feature pairs m_j, m_k, and by collecting multiple correspondences before solving the resulting collapsed set of equations.

the probability of correspondence (Line 9 in Table 11.9). If this probability exceeds a threshold χ, the correspondence vectors are set to the same value (Line 11). The EIF SLAM algorithm them interates the construction, reduction, and solution of the SLAM posterior (Lines 12 through 14). As a result, subsequent correspondence tests factor in previous correspondence decisionsm though a newly constructed map. The map construction is terminated when no further features are found in its inner loop.

Clearly, the alorithm **EIF_SLAM** is not particularly efficient. In particular, it tests all feature pairs for correspondence, not just nearby ones. Further, it reconstructs the map whenevr a single correspondence is found; rather than processing sets of corresponding features in batch. Such modifications, however, are relatively straiughtforward. A good implementation of **EIF_SLAM** will clearly be more refined than our basic implementation discussed here.

11.5.2 Mathematical Derivation

We essentially restrict our derivatoion to showing the correctness of the correspondence test in Table **??**. Our first goal shall be to define a posterior probability distribution over a variable $\Delta_{j,k} = m_j - m_k$, the *difference* between the location of feature m_j and feature m_k. Two features m_j and m_k are equivalent iff their location is the same. Hence, by calculating the posteiror probability of $\Delta_{j,k} =$, we obtain the desired correspondence probability.

We obtain the posterior for $\Delta_{j,k}$ by first calculating the joint over m_j and m_k:

$$\begin{aligned} & p(m_j, m_k \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ &= \int p(m_j, m_k \mid x_{1:t}, z_{1:t}, c_{1:t})\; p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t})\; dx_{1:t} \end{aligned} \tag{11.44}$$

We will denote the information form of this marginal posterior by $\xi_{[j,k]}$ and $\Omega_{[j,k]}$. Note the use of the squared brackets, which distinguish these values from the submatrices of the joint information form.

The distribution (11.44)is obtained from the joint posterior over $y_{0:t}$, by applying the marginalization lemma. Specifically, here Ω and ξ represent the joint posterior over the full state vector $y_{0:t}$, and $\tau(j)$ and $\tau(k)$ denote the sets of poses at which the robot observed feature j, and feature k, respectively. Further, from our EIF solution we are given the mean pose vector $\tilde{\mu}$. To apply the marginalization lemma (Table 11.6), we shall leverage the result of the algorithm **EIF_solve**. Specifically, **EIF_solve** provides us already with a mean for the features m_j and m_k. We simply restate the computation here for the joint feature pair:

$$\mu_{[j,k]} \;=\; \Omega^{-1}_{jk,jk}\; (\xi_{jk} + \Omega_{jk,\tau(j,k)} \mu_{\tau(j,k)}) \tag{11.45}$$

Here $\tau(j,k) = \tau(j) \cup \tau(k)$ denotes the set of poses at which the robot observed m_j or m_k.

For the joint posterior, we also need a covariance. This covariance is *not* computed in **EIF_solve**, simply because the joint covariance over multiple features requires space quadratic in the number of features. However, for pairs of feautures the covariance of the joint is easily recovered. Specicially, let $\Sigma_{\tau(j,k),\tau(j,k)}$ be the submatrix of the covariance $\Sigma_{0:t}$ restricted to all poses in $\tau(j,k)$. Here the covariance $\Sigma_{0:t}$ is calculated in Line 2 of the algorithm **EIF_solve**. Then the marginalization lemma provides us

with the marginal information matrix for the posterior over $(m_j \;\; m_k)^T$:

$$\Omega_{[j,k]} \;=\; \Omega_{jk,jk} - \Omega_{jk,\tau(j,k)}\, \Sigma_{\tau(j,k),\tau(j,k)}\, \Omega_{\tau(j,k),jk} \tag{11.46}$$

The information form representation for the desired posterior is now completed by the following information vector:

$$\xi_{[j,k]} \;=\; \Omega_{[j,k]}\, \mu_{[j,k]} \tag{11.47}$$

Hence we have for the joint

$$\begin{aligned} & p(m_j, m_k \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ &= \eta\, \exp\left\{ -\tfrac{1}{2} \begin{pmatrix} m_j \\ m_k \end{pmatrix}^T \Omega_{[j,k]} \begin{pmatrix} m_j \\ m_k \end{pmatrix} + \begin{pmatrix} m_j \\ m_k \end{pmatrix}^T \xi_{[j,k]} \right\} \end{aligned} \tag{11.48}$$

These equations are identical to Lines 2 and 3 in Table 11.8.

The nice thing about our representation is that it immediately lets us define the desired correspondence probability. For that, let us consider the random variable

$$\Delta_{j,k} \;=\; m_j \;-\; m_k \tag{11.49}$$

$$\;=\; \begin{pmatrix} 1 \\ -1 \end{pmatrix}^T \begin{pmatrix} m_j \\ m_k \end{pmatrix}$$

$$\;=\; \begin{pmatrix} m_j \\ m_k \end{pmatrix}^T \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{11.50}$$

Plugging this into the definition of a Gaussian in informtation representation, we obtain:

$$\begin{aligned} & p(\Delta_{j,k} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ &= \eta\, \exp\left\{ -\tfrac{1}{2}\, \Delta_{j,k}^T \underbrace{\begin{pmatrix} 1 \\ -1 \end{pmatrix}^T \Omega_{[j,k]} \begin{pmatrix} 1 \\ -1 \end{pmatrix}}_{=:\, \Omega_{\Delta_{j,k}}} \Delta_{j,k} + \Delta_{j,k}^T \underbrace{\begin{pmatrix} 1 \\ -1 \end{pmatrix}^T \xi_{[j,k]}}_{=:\, \xi_{\Delta_{j,k}}} \right\} \end{aligned}$$

$$= \eta \exp\left\{-\tfrac{1}{2}\, \Delta_{j,k}^T\, \Omega_{\Delta j,k}\, + \Delta_{j,k}^T\, \xi_{\Delta j,k}\right\}^T \tag{11.51}$$

which is Gausian with information matrix $\Omega_{\Delta j,k}$ and information vector $\xi_{\Delta j,k}$ as defined above. To calculate the probability that this Gaussian assumes the value of $\Delta_{j,k} = 0$, it shall be useful to rewrite this Gaussian in moments form:

$$\begin{aligned} & p(\Delta_{j,k} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ & = |2\pi\, \Omega_{\Delta j,k}^{-1}|^{-\frac{1}{2}}\, \exp\left\{-\tfrac{1}{2}\, (\Delta_{j,k} - \mu_{\Delta j,k})^T\, \Omega_{\Delta j,k}^{-1}\, (\Delta_{j,k} - \mu_{\Delta j,k})\right\} \end{aligned} \tag{11.52}$$

where the mean is given by the obvious expression:

$$\mu_{\Delta j,k} = \Omega_{\Delta j,k}^{-1}\, \xi_{\Delta j,k} \tag{11.53}$$

These steps are found in Lines 4 through 6 in Table 11.8.

The desired probability for $\Delta_{j,k} = 0$ is the result of plugging 0 into this distribution, and reading off the resulting probability:

$$p(\Delta_{j,k} = 0 \mid z_{1:t}, u_{1:t}, c_{1:t}) = |2\pi\, \Omega_{\Delta j,k}^{-1}|^{-\frac{1}{2}}\, \exp\left\{-\tfrac{1}{2}\, \mu_{\Delta j,k}^T\, \Omega_{\Delta j,k}^{-1}\, \mu_{\Delta j,k}\right\} \tag{11.54}$$

This expression is the probability that two features in the map, m_j and m_k, correspond to the same features in the map. This calculation is implemented in Line 7 in Table 11.8.

11.6 EFFICIENCY CONSIDERATION

Practical implementations of EIF SLAM rely on a number of additional insights and techniques for improving efficiency. Possibly the biggest deficiency of EIF SLAM, as discussed thus far, is due to the fact that in the very beginnning, we assume that all observed features constitute different landmarks. Our algorithm them unifies them one-by-one. For any reasonable number of features, such an approach will be unbearably slow. Further, it will neglect the imprtant constraint that at any point in time, the same feature can only be observed once, but not twice.

Existing implementations of the EIF SLAM idea exploit such opportunities. Spoecifically,

1. Features that are immediately identified to correspond with high likelihood are often unified form the very beginning, before running the full EIF SLAM solution. For exmaple, it is quite common to compule short segments into *local maps*, e.g., local occupancy grid maps. EIF inference is then performed onlhy between those local occupancy grid maps, where the match of two maps is taken as a probabilistic constraint between the relative poses of these maps. Such a hierarchical technique reduces the complexity of SLAM by orders of magnitude, while still retaining some of the key elements of the EIF solution, specifically the ability to perform data association over large data sets.

2. Many robots are equipped with sensors that observe large number of features at a time. For exampl, laser range finders observe dozens of features. For any such scan, one commonly assumes that different measurements indeed correspond to different features in the environment, by virtue of the fact that each scan points in a different direction. It therefore follows that $i \neq j \longrightarrow c_t^i \neq c_t^j$, that is, at no point in time correspond to different measurements to the same feature.

   Our pairwise data association technique above is unable to incorporate this constraint. Specifically, it may assign two measurements z_t^i and z_t^j to the same feature z_s^k for some $s \neq t$. To overcome this problem, it is common to assiciate entire measurement vectors z_t and z_s at the same time. This involves a calculation of a joint over all features in z_t and z_s. Such a calculation generalizes our pairwaise calculation and is mathematically straightforward.

3. The EIF algorithm stated in this chapter does not make use of its ability to *undo* data asociations. Once a data association decision is made, it cannot be reverted further down in the search. mathematically, it is relatively straightforward to undo past data association decisions in the information framework. in particular, one can change the correspondence variables of any two measurements in arbitrary ways in our algothm above. However, it is more difficult to test whether a data associaiton should be undone, as there is no (obvious) test for testing wether two previously associated features should be distinct. A simple implementation involves undoing a data association in question, rebyuilding the map, and testing whether our criterion above still calls for correspondence. Such an approach can be computationally involved, as it provides no means of detecting which data association to test. Mechnicsms for detecting unlikely associations are outside the scope of this book, but should be considered when implementing this approach.

Figure 11.1 The Groundhog robot is a 1,500 pound custom-built vehicle equipped with onboard computing, laser range sensing, gas and sinkage sensors, and video recording equipment. The robot has been built to map abandoned mines.

4. Finally, the EIF algorihtm does not consider negative information. In practice, not seeing a feature can be as informative as seeing one. However, the simple formulaiton above does not perform the necessary geometric computations.

   In practice, whether or not we can exploit negative information depends on the nature of the sensor model, and the model of our features in the world. For example, we might have to compute probabilities of occlusion, which might be tricky for certain type sensors (e.g., range and bearing sensors for landmarks). However, contemporary implementations indeed consider negative information, but often by replacing proper probabiliustic calculations through approximations. One such example will be given in the next section.

11.7 EMPIRICAL IMPLEMENTATION

In the remainder of tihs chapter, we will highlight empirical results for an EIF SLAM implementation. The vehicle used in our experiment is Figure 11.1; it is a robot designed to map abandoned mines.

The type map collected by the robot is shown in Figure 11.2. This map is an occupancy grid map, using effectively pairwise scan matching for recovering the robot's poses. Pairwise scan matching can be thought of as a version of EIF SLAM, but correspondence is only established between immediately consecutive scans. The result of this approach leads to an obvious deficiency of the map shown in Figure 11.2.

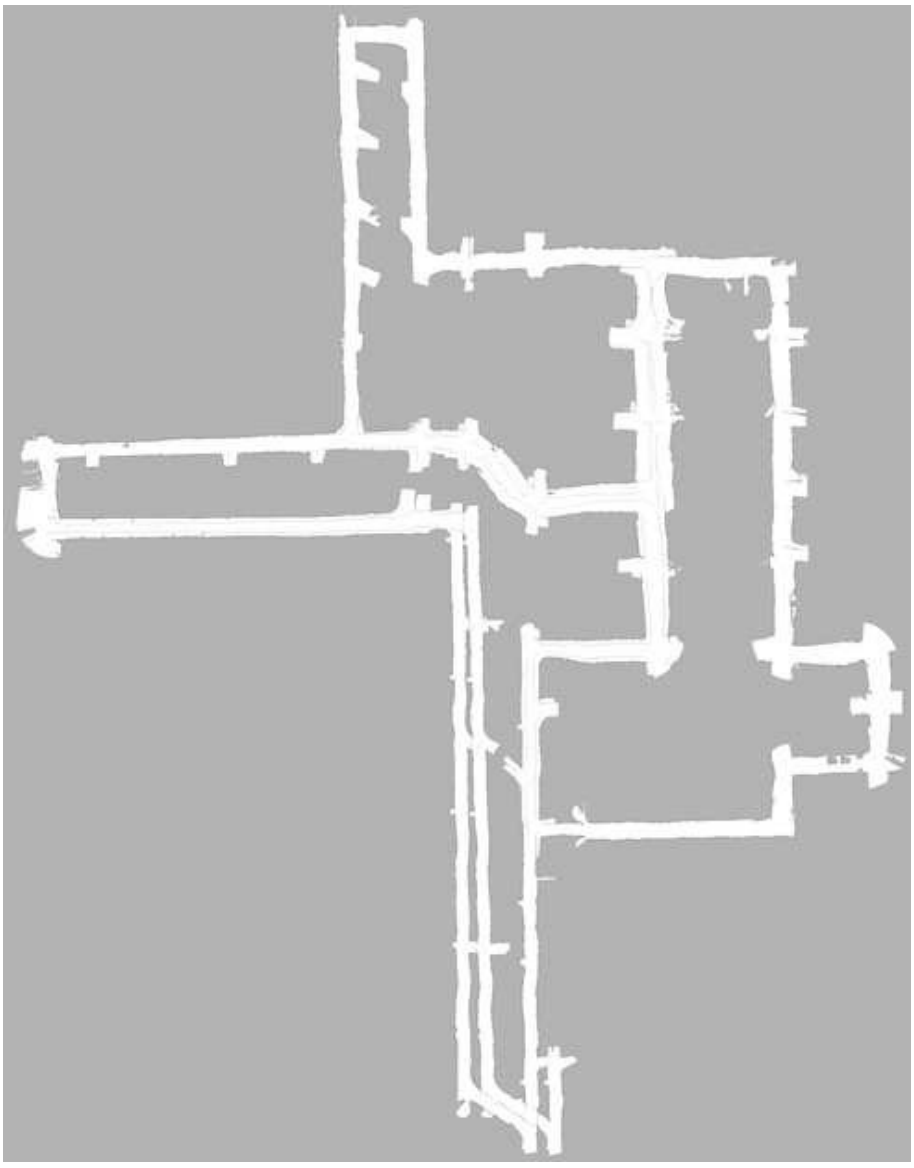

Figure 11.2 Map if a mine, acquired by pairwise scan matching. The diameter of this environment is approximately 250 meters. The map is obvisouly inconsistent, in that several hallways show up more than once.

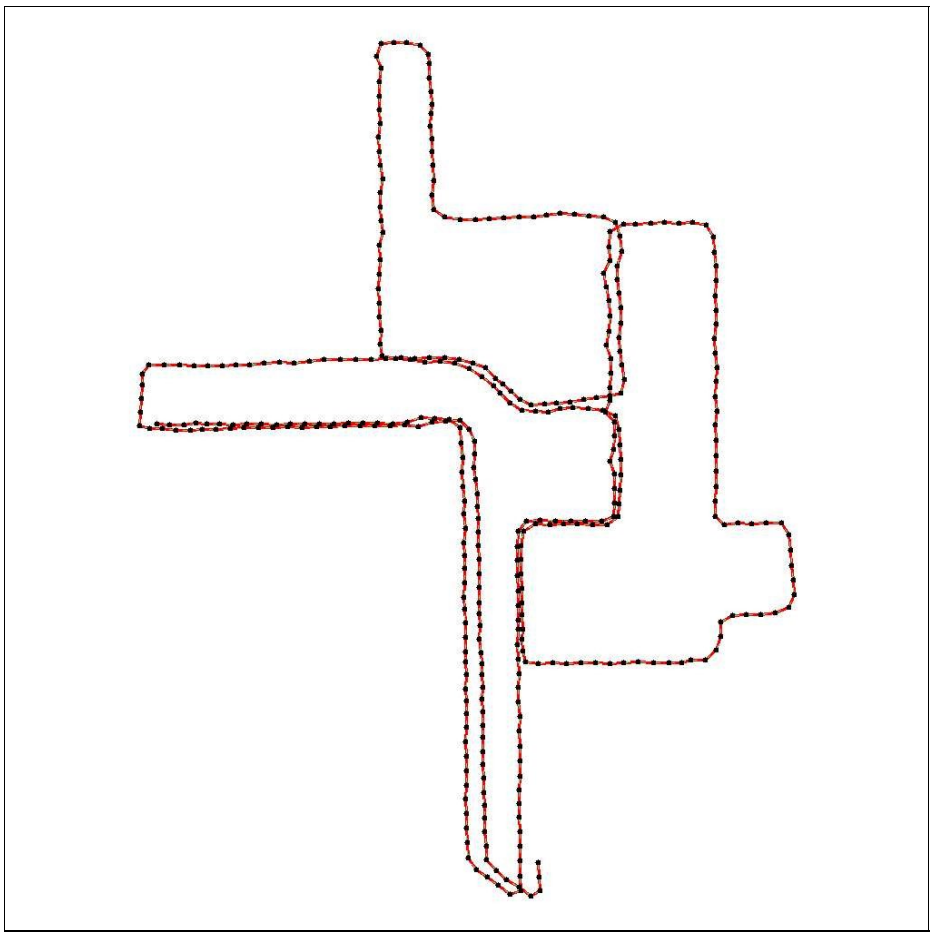

Figure 11.3 Map if a mine, acquired by pairwise scan matching. The diameter of this environment is approximately 250 meters. The map is obvisouly inconsistent, in that several hallways show up more than once.

To apply the EIF SLAM algorithm, our software decomposes the map into small local submaps, one for each five meters of robot travel. Within these five meters, the maps are sufficiently accurate, as general drift is small and hence the scan matching data association technique performs essentially flawlessly. Each submap's coordinates become a pose node in the EIF SLAM. Adjacent submaps are linked through the relative motion constraints between them. The resulting structure is shown in Figure 11.3.

Next, we apply the recursive data association search. The correspondence test is now implemented using a correlation analysis for two overlaying maps, and the Gaussian matching constraints are recovered by approximating this match function through a Gaussian. Figure 11.4 illustrates the process of data association: The circles each correspond to a new constraint that is imposed when contracting the information form of the EIF. This figure illustrates the iterative nature of the search: Certain correspondences are only discovered when others have been propagated, and others are dissolved in the process of the search. The final model is stable, in that additional search for new data association induces no further changes. Displayed as a grid map, it yields the 2-D map shown in Figure 11.5. While this map is far from being perfect (largely due to a crude implementation of the local map matching constraints), it nevertheless is superior to the one found through incremental scan matching.

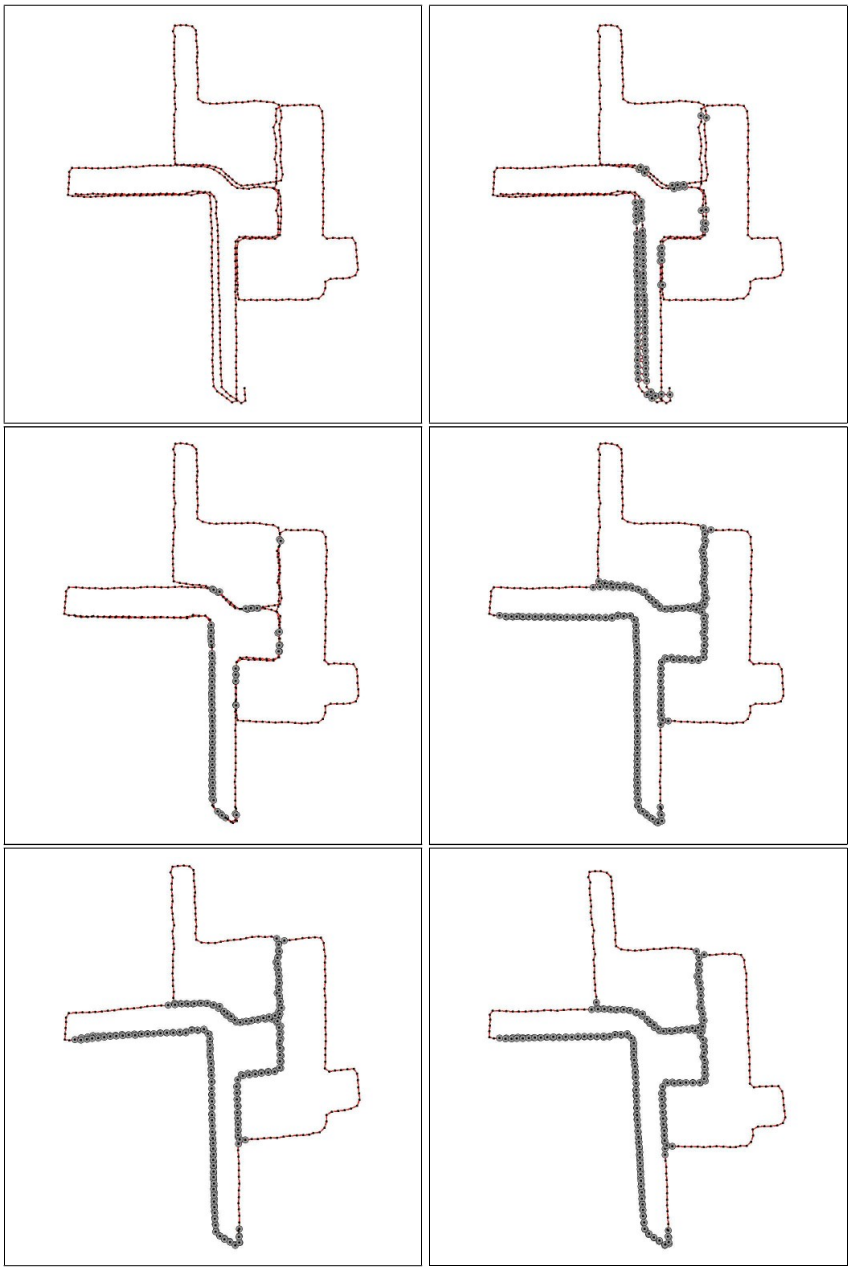

Figure 11.4 Data association search.

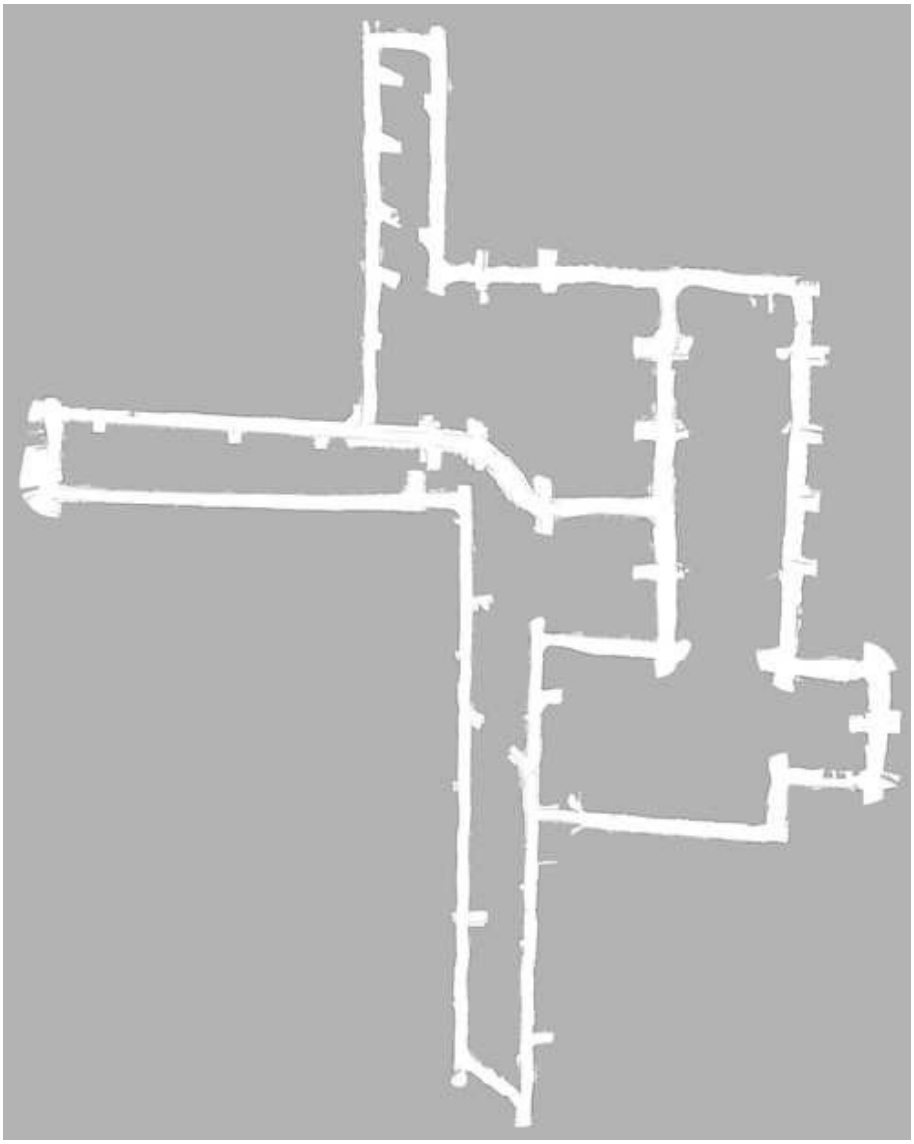

Figure 11.5 Final map, after optimizing for data associations.

11.8 SUMMARY

This chapter introduced the extended information filter (EIF) approach to to the full SLAM problem.

- the EIF SLAM algorithm addresses the full SLAM problem. It calculates posteriors over the full robot path along with the map. Therefore, EIF SLAM is a batch algorithm, not an online algorithm like EKF SLAM.
- The key insight of the EIF SLAM algorithm is that the structure of information is sparse. Specifically,
  - Measurements provide information of a feature relative to the robot's pose at the time of measurement. In information space, they form constraints between these pairs of variables.
  - Similarly, motion provides information between two subsequent poses. In information space, each motion command forms a constraint between subsequent pose variables.

  EIF SLAM simply records all this information, through links that are defined between poses and features, and pairs of subsequent poses. However, this information representation does not provide estimates of the map, or the robot path.
- The EIF SLAM algorithm recovers maps through an iterative procedure which involves three steps: Construction of a linear information form through Taylor extension; reduction of this form; and solving the resulting optimization problem. These three steps effectively resolve the information, and produce a consistent probabilistic posterior over the path and the map. Since EIF SLAM is run batch, we can repeat the linearization step and achieve gradually improved results.
- Data association in EIF SLAM is performed by calculating the probability that two features have identical world coordinates. Since EIF SLAM is a batch algorithm, this can be done for any pair of features, at any time. This led to an iterative greedy search algorithm over all data association variables, which recursively identifies pairs of features in the map that likely correspond.
- Practical implementations of the EIF SLAM often use additional tricks, to keep the computation low and to avoid false data associations. Specifically, practical implementations tend to reduce the data complexity by extracting local maps and using each map as the basic entity; they tend to match multiple features at-a-time, and they tend to consider negative information in the data association.
- We briefly provided results for a variant of EIF SLAM that follows the decomposition idea, and that uses occupancy grid maps for representing sets of range

scans. Despite these approximations, we find that the EIF data association and inference techniques yield favorable results in a large-scale mapping problem.