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*.

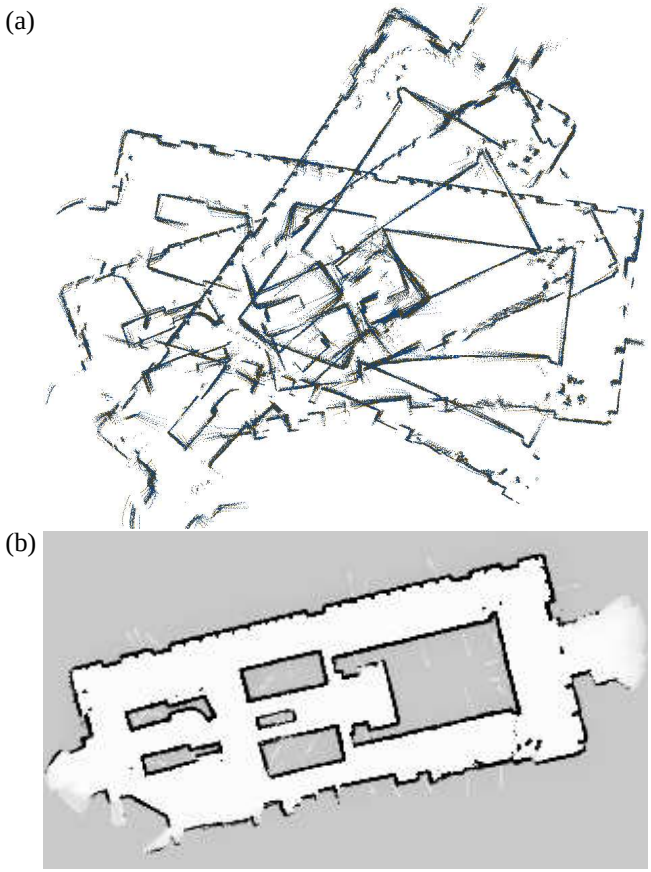

Figure 9.1 (a) Raw range data, position indexed by odometry. (b) Map

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 filter 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

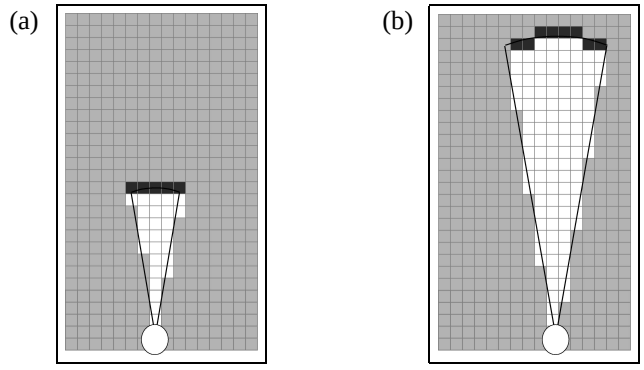

Figure 9.2 Two examples of our inverse measurement model **inverse_range_sensor_model** for two different measurement ranges. The darkness of each grid cell corresponds to the likelihood of occupancy.

$l_{\text{occ}} > l_0$ 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.

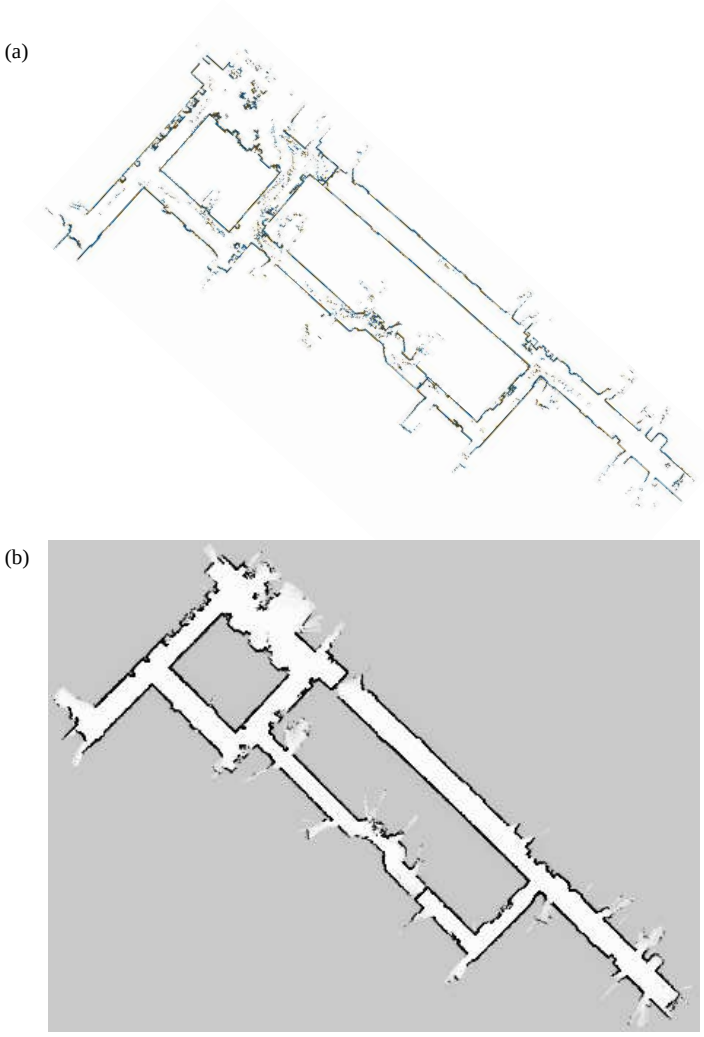

Figure 9.4 (a) Raw laser range data with correct(ed) pose information. Each dot corresponds to a detection of an obstacle. Most obstacles are static (walls etc.), but some were people that walked in the vicinity of the robot. (b) Occupancy grid map built from the data. The grey-scale indicates the posterior 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.

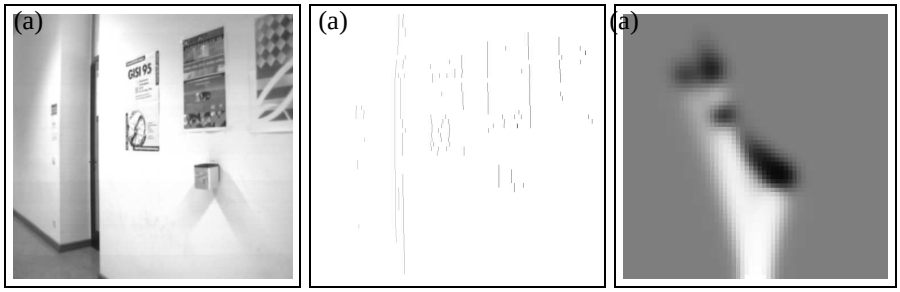

Figure 9.5 Estimation of occupancy maps using stereo vision: (a) camera image, (b) sparse disparity map, (c) occupancy map by projecting the disparity image onto the 2-D plane and convolving the result with a Gaussian.

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.

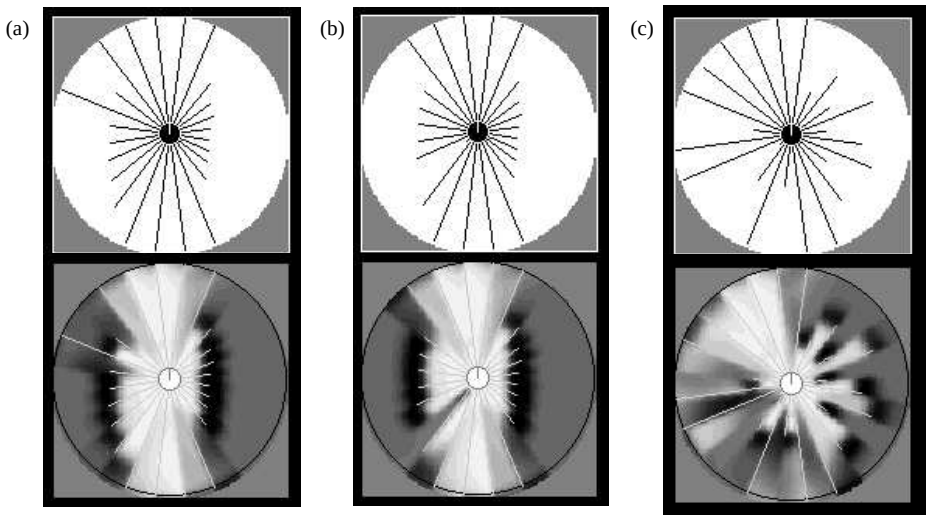

Figure 9.6 Sensor interpretation: Three sample sonar scans (top row) and local occupancy maps (bottom row), as generated by the neural network. Bright regions indicate free-space, and dark regions indicate walls and obstacles (enlarged by a robot diameter).

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.

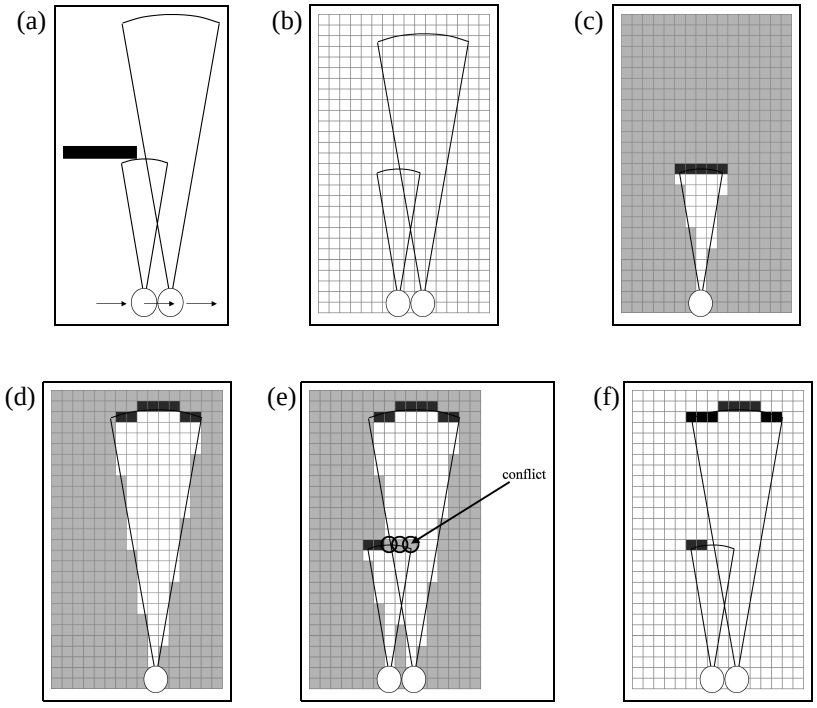

Figure 9.7 The problem with the standard occupancy grid mapping algorithm in Section 9.2: For the environment shown in Figure (a), a passing robot might receive the (noise-free) measurement shown in (b). The factorial approach maps these beams into probabilistic maps separately for each grid cell and 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: $m_i = \underset{k=0,1}{\operatorname{argmax}} (l_0)^k + \sum_t \log$
$\mathbf{measurement\_model}(z_t, x_t, m \text{ with } \mathbf{m}_i = k)$
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.

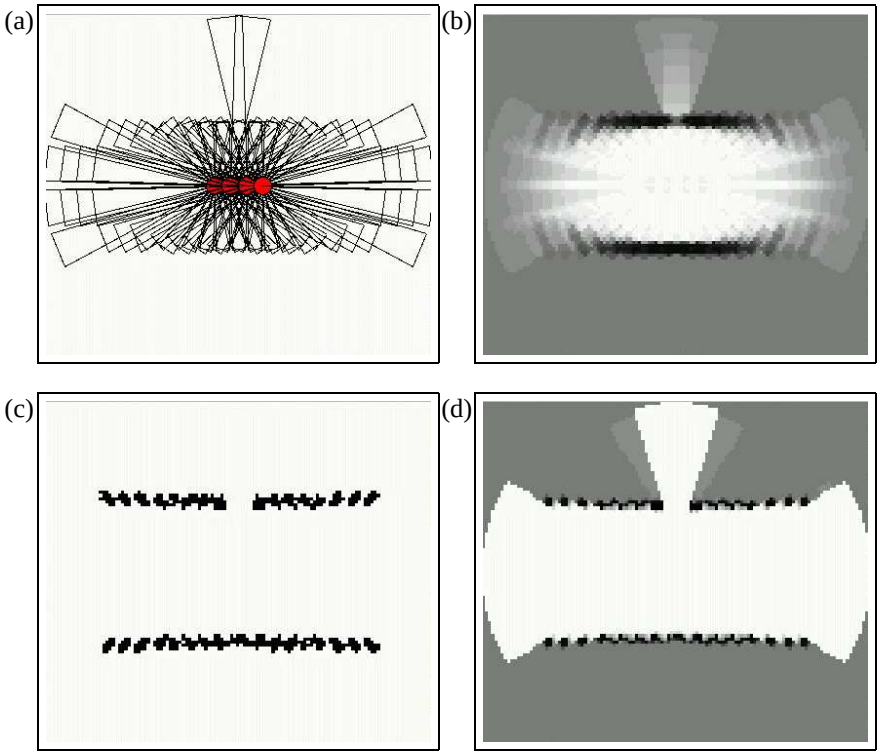

Figure 9.8 (a) sonar range measurements from a noise-free simulation; (b) Results of the standard occupancy mapper, lacking the open door. (c) A maximum a posterior map. (d) The residual uncertainty in this map, obtained by measuring the sensitivity of the map likelihood function with respect to individual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