# Chapter 7
# Local Navigation: Obstacle Avoidance

A mobile robot must *navigate* from one point to another in its environment. This can be a simple task, for example, if a robot can follow an unobstructed line on the floor of a warehouse (Sect. 3.4), but the task becomes more difficult in unknown and complex environments like a rover exploring the surface of Mars or a submersible exploring an undersea mountain range. Even a self-driving car which travels along a road needs to cope with other cars, obstacles on the road, pedestrian crosswalks, road construction, and so on.

The navigation of a self-driving car can be divided into two tasks: There is the high-level task of finding a path between a starting position and a goal position. Before the development of modern computer systems, finding a path required you to study a map or to ask directions. Now there are smartphone applications which take a starting position and a goal position and compute paths between the two positions. If the application receives real-time data on traffic conditions, it can suggest which path will get you to your goal in the shortest time. The path can be computed offline, or, if you have a GPS system that can determine your current position, the path can be found in real-time and updated to take changing conditions into account.

A self-driving car must also perform the lower-level task of adapting its behavior to the environment: stopping for a pedestrian in a crosswalk, turning at intersections, avoiding obstacles in the road, and so on. While high-level path finding can be done once before the trip (or every few minutes), the low-level task of obstacle avoidance must be performed frequently, because the car never knows when a pedestrian will jump into the road or when the car it is following will suddenly brake.

Section 7.1 looks at the low-level task of obstacle avoidance. Section 7.2 shows how a robot can recognize markings while following a line so that it knows when it has reached its goal. Sections 7.3–7.5 demonstrate a higher-level behavior: finding a path without a map of the environment. This is done by analogy with a colony of ants locating a food source and communicating its location to all members of the colony.



M. Ben-Ari and F. Mondada, *Elements of Robotics*,
https://doi.org/10.1007/978-3-319-62533-1_7

## 7.1 Obstacle Avoidance

The algorithms presented so far have focused on detecting objects and moving towards them. When a robot moves toward a goal it is likely to encounter additional objects called *obstacles* that block the path and prevent the robot from reaching its goal. We assume that the robot is able to detect if there is an unobstructed path to the goal, for example, by detecting a light on the goal. This section describes three algorithms for obstacle avoidance, where the obstacles are walls that block the robot's movement:

- A straightforward wall following algorithm, which unfortunately will not work if there are multiple obstacles in the environment.
- An algorithm that can avoid multiple obstacles, but it must know the general direction of the goal (perhaps from its GPS system). Unfortunately, some obstacles can cause the robot to become trapped in a loop.
- The Pledge algorithm is a small modification of the second one that overcomes this erroneous behavior.

The algorithms will use the abstract conditional expressions wall-ahead and wall-right, which are true if there is a wall close to the front or to the right of the robot. The first algorithm will also use the conditional expression corner-right which is true if the robot is moving around an obstacle and senses a corner to its right. There are several ways of implementing these expressions which we pursue in Activity 7.1.

**Activity 7.1: Conditional expressions for wall following**

- Implement the conditional expression wall-ahead using a horizontal proximity or a touch sensor.
- Implement the conditional expression wall-right. This is easy to do with a sensor mounted on the right of the robot, or with a rotating distance sensor. If you have only a forward-facing proximity sensor, you can have the robot turn slightly to the right, detect the wall, if any, and then turn back again.
- Implement the conditional expression corner-right. This can be implemented as an extension of wall-right. When the value of wall-right changes from true to false, make a short right turn and check if wall-right becomes true again.

### *7.1.1 Wall Following*

Figure 7.1 shows a robot performing wall following by maintaining its position so that the wall is to its right (Algorithm 7.1). If a wall is detected ahead, the robot turns left so that the wall is to its right. If a wall is detected to the right, the robot continues

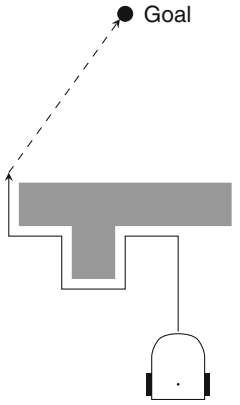


**Fig. 7.1** Wall following

moving along the wall. If a corner is detected, the robot turns right to continue moving around the obstacle. At the same time, the robot continually searches for the goal (black dot). When it detects the goal the robot moves directly towards it.

**Algorithm 7.1: Simple wall following**

```
1: while not-at-goal
2:     if goal-detected
3:         move towards goal
4:     else if wall-ahead
5:         turn left
6:     else if corner-right
7:         turn right
8:     else if wall-right
9:         move forward
10:    else
11:        move forward
```

Unfortunately, Algorithm 7.1 does not always work correctly. Figure 7.2 shows a configuration with *two* obstacles between the robot and the goal. The robot will never detect the goal so it will move around the first obstacle indefinitely.

**Activity 7.2: Simple wall following**

- Implement Algorithm 7.1 and verify that it demonstrates the behaviors shown in Figs. 7.1, 7.2.

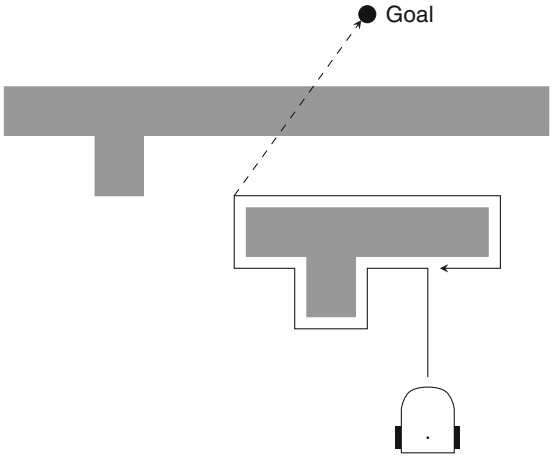


**Fig. 7.2** Simple wall following doesn't always enable the robot to reach the goal

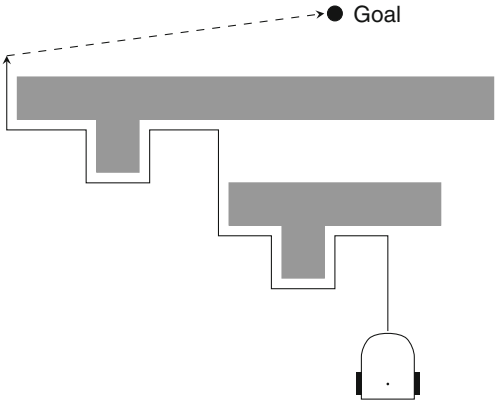


**Fig. 7.3** Wall following with direction

### *7.1.2 Wall Following with Direction*

The problem with Algorithm 7.1 is that it is a local algorithm that only looks at its immediate environment and does not take account of the fact that the higher-level navigation algorithm knows roughly the direction the robot should take to reach the goal. Figure 7.3 shows the behavior of a robot that "knows" that the goal is somewhere to its north so the robot moves at a heading of 0° relative to north. The wall following algorithm is only used if the robot cannot move north.

Algorithm 7.2 is similar to the previous algorithm except for its preference to move north if possible. It uses a variable heading to remember its current heading as

it moves around the obstacle. When heading is again north (a multiple of 360°), the robot moves forward instead of looking for a corner.

**Algorithm 7.2: Wall following**

```
    integer heading ← 0°
 1: while not-at-goal
 2:     if goal-detected
 3:         move towards goal
 4:     else if wall-ahead
 5:         turn left
 6:         heading ← heading + 90°
 7:     else if corner-right
 8:         if heading = multiple of 360°
 9:             move forward
10:         else
11:             turn right
12:             heading ← heading − 90°
13:     else if wall-right
14:         move forward
15:     else
16:         move forward
```

Unfortunately, the algorithm can fail when faced with a *G*-shaped obstacle (Fig. 7.4). After making four left turns, its heading is 360° (also north, a multiple of 360°) and it continues to move forward, encountering and following the wall again and again.

**Activity 7.3: Wall following with direction**

- Implement the wall following algorithm with direction and verify that it demonstrates the behavior shown in Fig. 7.4.

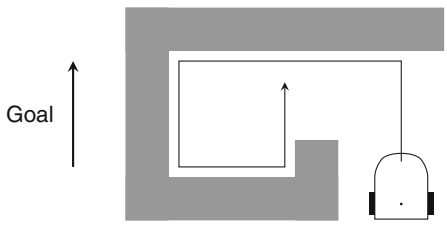


**Fig. 7.4** Why wall following with direction doesn't always work

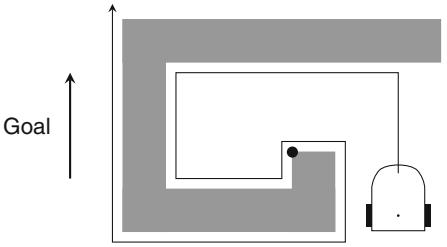


**Fig. 7.5** Pledge algorithm for wall following

- Run the simple wall following algorithm (Algorithm 7.1) with a *G*-shaped obstacle. What happens? Does this affect our claim that this algorithm is not suitable for obstacle avoidance?

### *7.1.3 The Pledge Algorithm*

The Pledge algorithm modifies line 8 of the wall following algorithm to:

if heading $= 0°$

The robot moves forward only when its cumulative heading is equal to 0° and not when it is moving north—a heading that is a multiple of 360°. The robot now avoids the "G"-shaped obstacle (Fig. 7.5): when its encounters the corner (black dot), it is moving north, but its heading is 360° after four left turns. Although 360° is a multiple of 360°, it is not equal to 0°. Therefore, the robot continues to follow the wall until four right turns subtract 360 so that the total heading is 0°.

**Activity 7.4: Pledge algorithm**

- Implement the Pledge algorithm and verify that it demonstrates the behavior shown in Fig. 7.5.

## 7.2 Following a Line with a Code

Let us return to the task of finding a path to a goal. If the path is marked by a line on the ground, line following algorithms (Sect. 3.4) can guide a robot within the

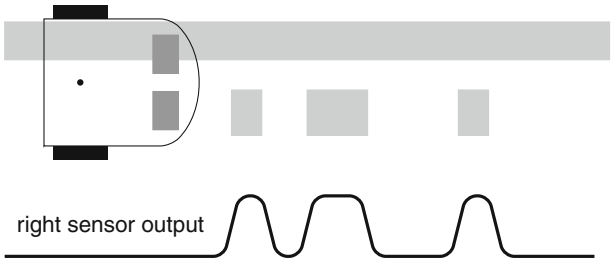


**Fig. 7.6** A robot following a line using its left sensor and reading a code with its right sensor

environment, but line following is not navigation. To navigate from one position to another we also need a localization algorithm so that the robot knows when it has reached its goals. We do not need a continuous localization algorithm like the ones in Chap. 8, we only need to know positions on the line that facilitate fulfilling the task. This is similar to navigating when driving: you only need to know about interchanges, intersections, major landmarks, and so on, in order to know where you are. Between such positions you can just follow the road.

Navigation without continuous localization can be implemented by reading a code placed on the floor next to the line. Figure 7.6 shows a robot with two ground sensors: the left one senses the line and the right one senses the code. Below the robot is a graph of the signal returned by the right sensor.

**Activity 7.5: Line following while reading a code**

- Implement line following with code reading as shown in Fig. 7.6.
- Write a program that causes a robot to follow a path.
- Place marks that encode values next to the path. The robot should display these values (using light, sound or a screen) when it moves over the codes.

**Activity 7.6: Circular line following while reading a code**

- Implement a clock using two robots, one for the minutes and one for the hours (Fig. 7.7).
- An alternate implementation would be to have the two robots move at different speeds so that one completes a revolution in one hour and the other completes a revolution in one day. Discuss the difference between the implementations.

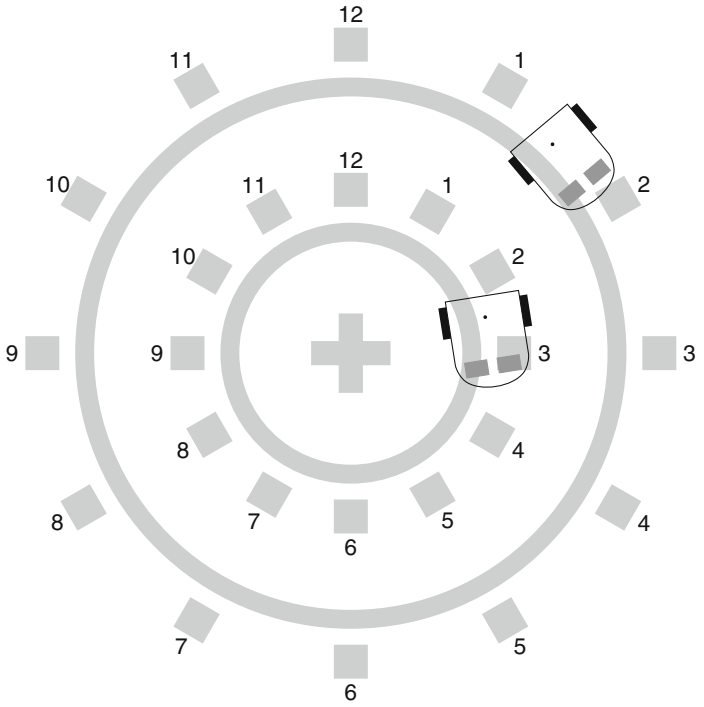


**Fig. 7.7** A robotic clock: one robot indicates the hour and the other indicates the minute

## 7.3 Ants Searching for a Food Source

Let us now return to the high-level algorithm of finding a path. If there is a line and a mechanism for localization like a code, the approach of the previous section can be used. However, even if a line does not exist, a robot may be able to *create* its own line. The interesting aspect of this method is that the robot does not need to know its location in the environment, for example, using a GPS; instead, it uses landmarks in the environment itself for navigation. The algorithm will be presented within the real-world context of ants searching for food:

> There exists a nest of ants. The ants search randomly for a source of food. When an ant finds food it returns directly to the nest by using landmarks and its memory of the path it took from the nest. During the return journey to the nest with the food, the ant deposits chemicals called *pheromones*. As more and more ants find the food source and return to the nest, the trail accumulates more pheromones than the other areas that the ants visit. Eventually, the amount of pheromones on the trail will be so strong that the ants can follow a direct path from the nest to the food source.

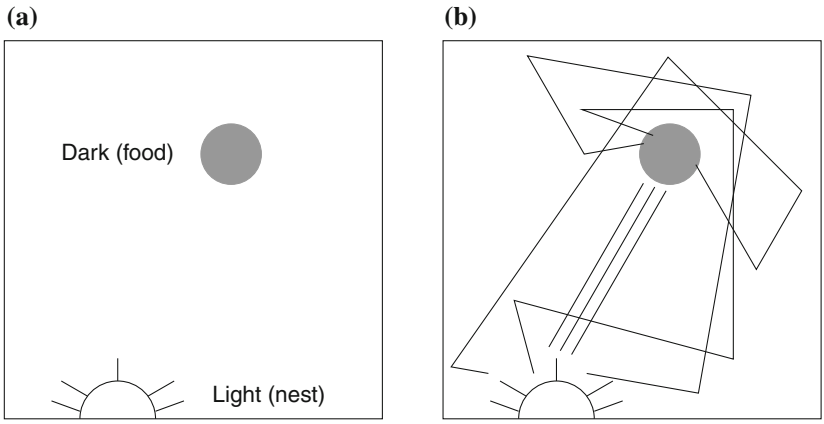


**Fig. 7.8** **a** The ants' nest and the food source. **b** Pheromones create a trail

Figure 7.8a shows the ants' nest in the lower left corner represented as a light that enables the ants to easily find their way back to the nest. The dark spot is the food source. Figure 7.8b shows three random trails that eventually discover the food source; then the ants return directly to the nest, leaving three straight lines of pheromones. This concentration can be subsequently used to find the food source directly.

The ant-like behavior can be implemented by a robot. Assume that there is a fixed area within which the robot can move. As in Fig. 7.8a there is a food source and a nest. The food source will be represented by a dark spot that can be easily detected by a ground sensor on the robot. The proximity sensors of the robot are used to detect the walls of the area. Activity 7.7 suggests two methods of representing the nest that depend on what additional sensors your robot has.

**Activity 7.7: Locating the nest**

- Implement a program that causes the robot to move to the nest no matter where it is placed in the area.
- Accelerometers: Mount the area on a slope such that one corner, the nest, is at the lowest point of the area.
- Light sensor: The nest is represented by a light source that can be detected by the light sensor regardless of the position and heading of the robot. If the light sensor is fixed and can detect light only from a certain direction, the robot will have to rotate to locate the light source.

Simulate the pheromones by covering the area with a sheet of white paper and attaching a black marker to the robot so that it draws a line wherever it moves. A

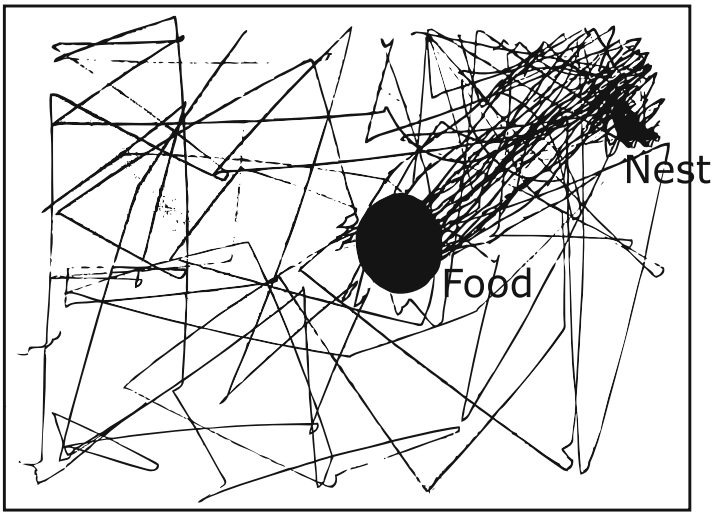


**Fig. 7.9** A robot simulating pheromones of ants

ground sensor detects the marks in the area. Figure 7.9 shows the lines resulting from the behavior of a robot running the algorithm. Activity 7.8 asks you to explore the ability of the robot to sense areas which have a high density of lines.

**Activity 7.8: Sensing areas of high density**

- In Sect. 3.4.3 we noted that sensors don't sense a single geometrical point but rather have an aperture that reads a relatively large area, perhaps even as much as a square centimeter (Fig. 3.6). Experiment with your ground sensor to see how the readings returned by the sensor depend on the width of the line. Can you come to any conclusion about the optimal width of the marker? If it is too thin the trail won't be detected and if it is too thick the markings of the random movement might be taken to be the trail.
- Represent the food source as a relatively large totally black spot and make sure that it gives a minimal reading of the ground sensor.
- Figure 7.9 shows that the trail between the food source and the nest has a high density. Experiment with various numbers of lines and define an effective threshold between the trail and areas of random motion outside the trail. See if you can get the robot to make darker lines by varying its motion or by moving back and forth along the trail.

## 7.4 A Probabilistic Model of the Ants' Behavior

A *model* is an abstraction of a system that shows how parameters impact phenomena. Models are used, for example, to study traffic patterns in order to predict the effect of new roads or traffic lights. To understand how the path from the nest to the food is generated, this section presents a simplified model the behavior of the ants.

The fundamental characteristic of the ants' behavior is that they do not have a map of their environment, so they must move randomly in order to search for the food source. Therefore, a model of their behavior must be probabilistic. Let us assume that the environment is a rectangular area that is a grid of cells. Figure 7.10 shows an area split into $6 \times 8 = 48$ cells.

**Coordinates in a grid of cells**

Throughout the book, the coordinates of a cell in a grid are given as (*row*, *column*). Rows are numbered from top to bottom and columns from left to right like matrices in mathematics, however, the numbering starts from 0, as in the array data type in computer science.

Without any information on how ants choose their movements, we assume that they can move in any direction with the same probability, so the probability $p$ of an ant being in any cell is 1 divided by the number of cells, here $p = 1/48 = 0.021$.

The probability that the ant is in the cell with the food source is $p$, the same as for any other cell. According to our specification of the ant's behavior, once it enters this cell and identifies the cell as the food source, it returns directly to the nest. In Fig. 7.10 the food source is in cell (3, 4), so an ant visiting that cell must return to the nest at cell (0, 7), passing through cells (2, 5) and (1, 6). What is the probability that the ant is in any of these three cells? There are two possibilities: either the ant is in the

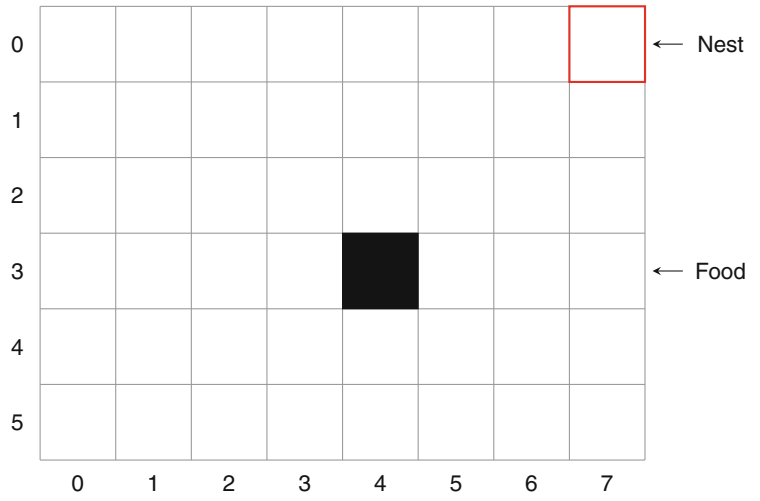


**Fig. 7.10** Representation of the environment as a grid of cells

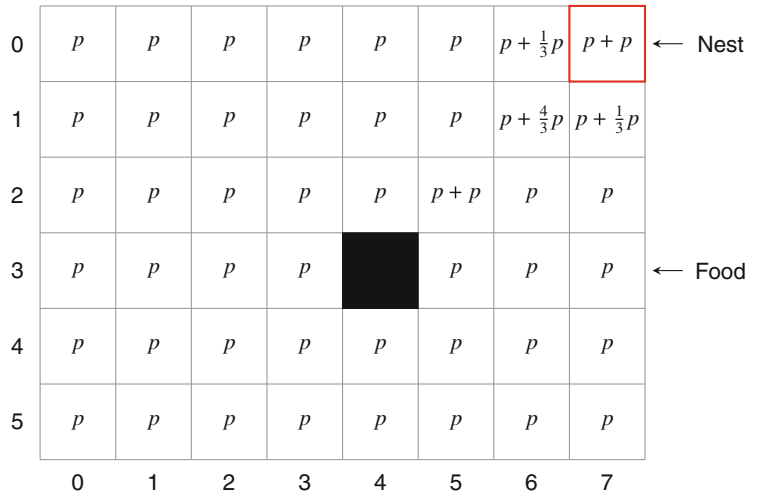


**Fig. 7.11** Probabilities for the location of the ant

cell because it randomly moved there with probability $p$, or the ant is there because it moved to the food source randomly with probability $p$ and then *with probability* 1 is moved towards the nest. Therefore, the total probability of being in any of those cells is $p + p \times 1 = p + p = 2p$.[1] If our robot is drawing lines as it moves, the cells on the diagonal should be twice as dark as the other cells.

Once the ant has reached the nest, it will move randomly again, that is, it will select a random neighbor to move to. In general a cell has eight neighbors (above and below, left and right, four on the diagonals), so the probability is $p/8$ that it will be in any one of these neighbors. The nest, however, is in the corner with only three neighbors, so the probability is $p/3$ that it will move to any one of them. Figure 7.11 shows the probability of the location of the ant after finding the food source, returning to the nest and making one additional random move. When implemented by a robot with a marker, the cells with higher probability will become darker (Fig. 7.12).

What can we conclude from this model?

- Although the ants move randomly, their behavior of returning to the nest after finding the food source causes the probability of being on the diagonal to be higher than anywhere else in the environment.
- Since the ants drop pheromones (black marks) at every cell they visit, it follows that the marks on the diagonal path between the food source and the nest will be darker than the marks on other cells. Eventually, the markings on this path will be sufficiently dark so that the robot can follow it to the food source without performing a random exploration.

[1] After the probabilities are updated they must be normalized as explained in Appendix B.2. For another example of normalization, see Sect. 8.4.

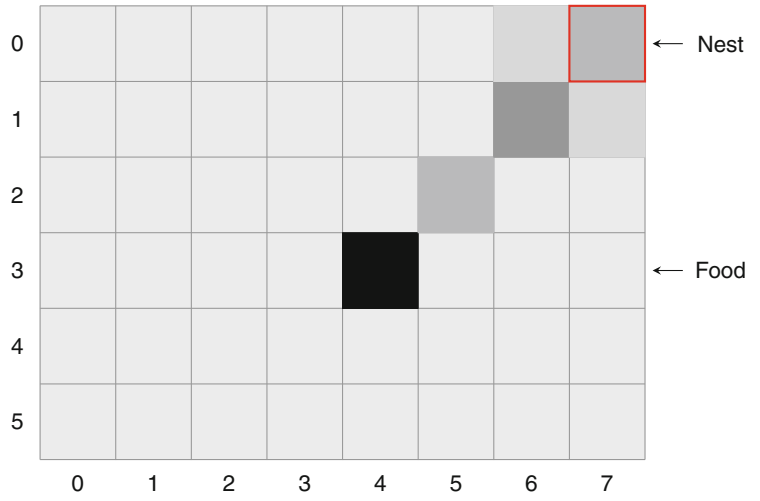


**Fig. 7.12** Probabilities for the location of a robot with a marker

- Since the robot visits the nest often, the cells in the immediate vicinity of the nest will have a probability somewhere between the uniform probability and the high probability of the trail. Therefore, it is important to emphasize the trail using methods such as those explored in Activity 7.8.

## 7.5 A Finite State Machine for the Path Finding Algorithm

An FSM for path finding by the ants is shown in Fig. 7.13. To save space the labels of the transitions use abbreviations which are explained in Table 7.1. Here is a detailed description of the behavior specified by this FSM in each state:

**search:** In this state the robot randomly searches for dark areas. It is the initial state and the transition true ⇝ fwd specifies that initially (and unconditionally) the robot is moving forwards and a timer is set to a random period. When the timer expires (timeout), the robot makes random turn, moves forwards and resets the timer. This random motion will continue until the robot encounters the wall of the area or a gray marking on the surface of the area. If it encounters a wall it makes a random turn *away* from the wall; we assume that the sensor faces directly ahead so the random turn must be in some direction to the side or the rear of the robot. Once the robot has detected a gray marking, it makes the transition to the follow state.

**follow:** The two self-transitions above and to the right of this state are transitions that implement line following (Sect. 3.4). There are three other transitions: Should a timeout occur without detecting gray, the robot is no longer following a line and must return to the search state. If the robot encounters a wall, we want it to turn away,

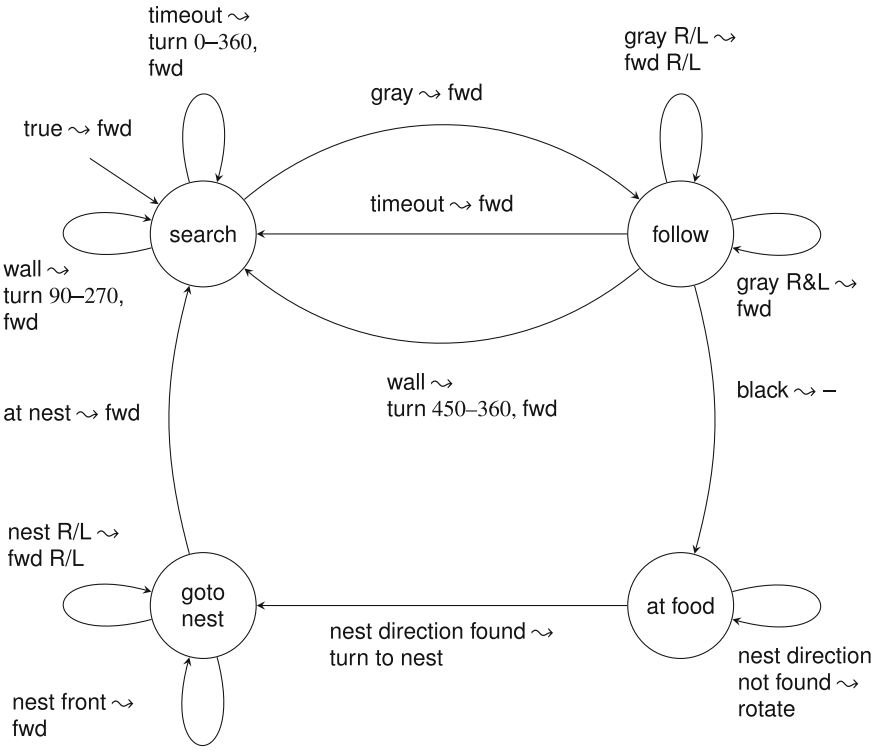


**Fig. 7.13** State machine for drawing a path between a food source and a nest see Table 7.1 for explanations of the abbreviations

**Table 7.1** Abbreviations in the state machine

| Abbreviation | Explanation |
|---|---|
| fwd | Set motor forwards |
| fwd R/L | Set motor forwards and to the right/left |
| | **fwd and fwd R/L also set the timer to a random period** |
| Wall | Wall detected |
| Timeout | Timer period expired |
| Gray R/L/R&L | Gray detected by right/left/both sensors |
| Nest front/R/L | Nest detected in front/right/left |
| Black | Black detected |
| Nest direction | Direction from food to nest found or not found |
| Turn $\theta_1 - \theta_2$ | Turn randomly in the range $\theta_1 - \theta_2$ |
| Rotate | The robot (or its sensor) rotates |

but first we ask it to make a full 360° turn to check if there is a gray marking in its vicinity. Therefore, the transition includes the action turn 450–360. Since the nest is next to a wall, this condition is also true when the robot returns to the nest. If the robot senses a high-density marking (black), it concludes that it has reached the food source and takes the transition to the state at food.

**at food:** Finally, the robot has discovered the food source. It must now return to the nest. We specified that the nest can be detected (Activity 7.7), but the robot's sensor does not necessarily face the direction of the nest. Therefore, the robot (or its sensor) must rotate until it finds the direction to the nest. When it does so, it turns towards the nest and takes the transition to the state goto nest.

**goto nest:** This state is similar to the follow state in that the robot moves forward towards the nest, turning right or left as needed to move in the direction of the nest. When it reaches the nest it returns to the search state.

Look again at Fig. 7.9 which shows an actual experiment with a robot running this algorithm. We see that there is a high density of lines between the nest and food source, but there is also a relatively high density of lines in the vicinity of the nest, not necessarily in the direction of the food source. This can cause to robot to return to random searching instead of going directly to the food source.

## 7.6 Summary

The obstacle avoidance algorithms use wall following algorithms that have been known since ancient times in the context of navigating a maze. When used for obstacle avoidance, various anomalies can cause the algorithms to fail, in particular, the $G$-shaped obstacle can trap a wall following algorithm. The Pledge algorithm overcomes this difficulty.

A colony of ants can determine a path between their nest and a food source without knowing their location and without a map by reinforcing random behavior that has a positive outcome.

## 7.7 Further Reading

There is a large literature on mazes that can be found by following the references in the Wikipedia article for *Maze*. The Pledge algorithm was discovered by 12-year-old John Pledge; our presentation is based on [1, Chap. 4]. A project based on ants following pheromones is described in [2].

## References

1. Abelson, H., diSessa, A.: Turtle Geometry: The Computer as a Medium for Exploring Mathematics. MIT Press, Cambridge (1986)
2. Mayet, R., Roberz, J., Schmickl, T., Crailsheim, K.: Antbots: A feasible visual emulation of pheromone trails for swarm robots. In: Dorigo, M., Birattari, M., Di Caro, G.A., Doursat, R., Engelbrecht, A.P., Floreano, D., Gambardella, L.M., Groß, R., Şahin, E., Sayama, H., Stützle, T. (eds.) Proceedings of Swarm Intelligence: 7th International Conference, ANTS 2010, Brussels, Belgium, 8–10 Sep. 2010, pp. 84–94. Springer Berlin Heidelberg (2010)