9. MOBILE ROBOTS

9.1 Introduction

Mobile robots are autonomous machines that are capable of moving and navigating in known or unknown environments, such as offices, factories and other industrial settings. Their navigation ability, or collision-free path planning, is the key difference between mobile robots and other movable vehicles, such as cars. With navigational ability, the robot can autonomously plan and move along a path from its current location to its desired goal location while avoiding collisions with obstacles. In an industrial environment, these obstacles include both humans and equipment. It is important for the robot to avoid collisions that could cause injuries to humans or cause damage to equipment or itself.

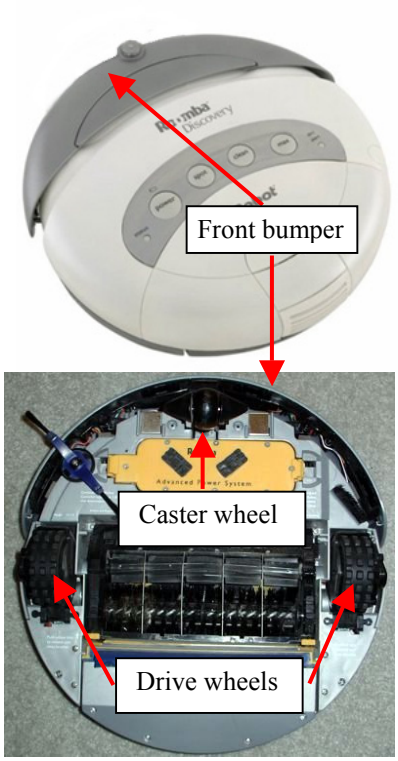


Figure 9.1 Top and bottom views of the Roomba® vacuum cleaning robot [1].

Different types of mobile robots include legged (*i.e.* designs inspired by human, canine and insect kinematics), tracked (*i.e.* similar in design to a military tank, and good for moving over soft ground) and wheeled mobile robots. This chapter focuses on three wheeled mobile robots that move over a 2D plane since they are the most successful, such as the Roomba® [1] vacuum cleaning robot which has sold more than 3 million units (see Figure 9.1). Other applications for wheeled mobile robots include delivery (see Figure 9.2), security and tele-presence. They can also be equipped with a robot arm, see Figure 1.26 for an example. This chapter will also be limited to robots whose cross-section can be modelled as a disk. It will cover the kinematics of mobile robots, the geometric modeling of robots and obstacles, and three path-planning methods: visibility graph, simplified visibility graph, and distance sensor-based.

9.2 Kinematics of Mobile Robots

Figure 9.2 A mobile robot delivering medical specimens in a hospital [2].

Based on their kinematic designs, most mobile robots can be classified as either: holonomic and non-holonomic. In robotics, holonomicity is determined by the relationship between the number of controllable degrees of freedom (DOFs) and total number of DOFs of a robot. If the number controllable DOFs equals the number of total DOFs, the robot is holonomic. If they number less than the total DOFs, the robot is non-holonomic. Movement over a 2D plane involves a total of three DOFs, *i.e.* two orthogonal translations and one rotation. With a holonomic robot, in terms of the robot's coordinate frame, these are: translating left and right (termed "sway"), translating forward and backward (termed "surge") and rotating left and right (termed "yaw"). A robot that cannot control one of these DOF is termed non-holonomic. For example, a robot designed like a car is non-holonomic since it only has only two controllable DOFs: surging (throttle and brake) and yawing (steering). This explains why some manoeuvers, *e.g.* parallel parking, are difficult to perform with a car.

Advantages of holonomic robots are:
- Greater manoeuvrability. This is very important for environments with dynamic obstacles such as humans.
- Suitable for operating in narrow spaces since they require much less space for turning.
- Motion control is simple in comparison to non-holonomic robots.

Advantages of non-holonomic robots are:
- Hardware is simpler and less expensive since fewer motors are used.

- Standard wheels may be used. These have better traction than the omni-directional wheels used with holonomic robots, are less expensive and consume less energy.

The following assumptions will be used when analyzing the kinematics of mobile robots:

1. The robot's structure is rigid;
2. The robot moves on 2D plane;
3. Its wheels roll without slipping; and
4. Its wheels are non-deformable.

9.2.1 Holonomic robots

Figure 9.3 shows an example of the most popular design for holonomic robots. Its geometry is depicted in Figure 9.4. With this design, three omni-directional wheels are driven by three motors. The rotation axes of the wheels intersect at a single point. An omni-directional wheel allows free motion perpendicular to its regular rolling direction. The wheel design shown in Figure 9.5 uses a combination of two offset disks, each with a series of free-spinning rollers, to achieve this omni-directional behaviour. Note that one disadvanage of omni-directional wheels is that they have poor traction compared to conventional wheels.

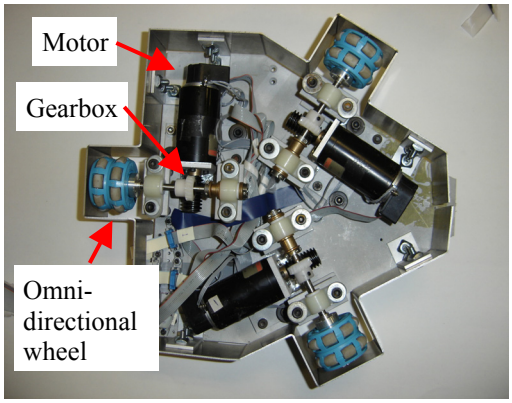


Figure 9.3 Example of a holonomic robot. This robot was designed and built at McMaster University by Lingqi Zeng, Adam Trischler and Gary Bone.

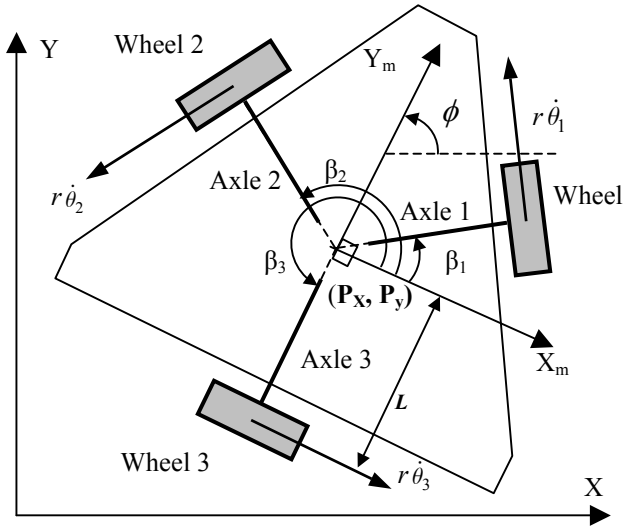


Figure 9.4 Holonomic robot geometry.

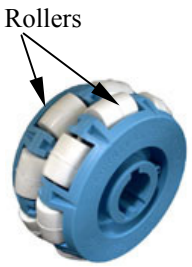


Figure 9.5 An omni-directional wheel [3].

Assuming the robot is moving with the velocity vector, $\dot{\mathbf{p}}_m = \begin{bmatrix} \dot{x}_m & \dot{y}_m & \dot{\phi} \end{bmatrix}$; the radius of the wheel is r and the distance between the geometric centre of the robot and a wheel is L, the ith wheel's angular velocity, $\dot{\theta}_i$, is given by:

$$\dot{\theta}_i = \left(-\dot{x}_m \sin\beta_i + \dot{y}_m \cos\beta_i + L\dot{\phi}\right)/r \qquad (9.1)$$

This kinematic relationship is illustrated in Figure 9.6 for the case when $\dot{\phi} = 0$. The rollers have a speed of $\dot{x}_m \cos\beta_i + \dot{y}_m \sin\beta_i$. Since it is unnecessary to control the rollers' speed, we have:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{\theta}_1 & \dot{\theta}_2 & \dot{\theta}_3 \end{bmatrix}^T = \mathbf{J}_m^{-1}\dot{\mathbf{p}}_\mathbf{m} \qquad (9.2)$$

That is:

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} = \frac{1}{r}\begin{bmatrix} -\sin\beta_1 & \cos\beta_1 & L \\ -\sin\beta_2 & \cos\beta_2 & L \\ -\sin\beta_3 & \cos\beta_3 & L \end{bmatrix}\begin{bmatrix} \dot{x}_m \\ \dot{y}_m \\ \dot{\phi} \end{bmatrix} \qquad (9.3)$$

where $\dot{\theta}_1, \dot{\theta}_2$ and $\dot{\theta}_3$ are the corresponding angular velocities of the three

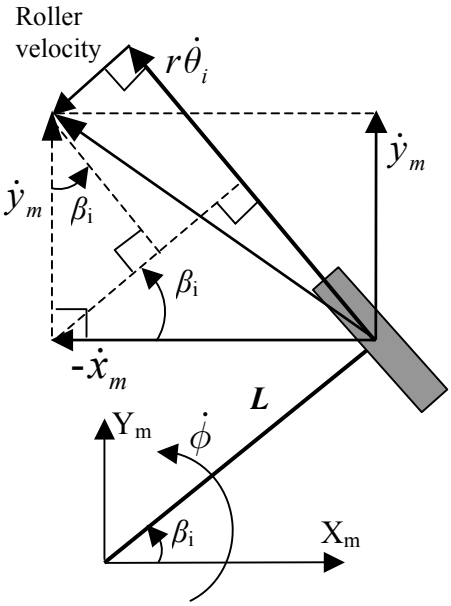


Figure 9.6 Kinematics of a wheel when $\dot{\phi} = 0$.

wheels in Figure 9.4. Equation 9.3 describes the inverse velocity kinematics in terms of the robot's coodinate frame.

For the kinematics in world coordinates, the required transformation matrix is :

$$\mathbf{R}=\text{Rot}(Z_m, 90^o-\phi)=\begin{bmatrix}\sin\phi & -\cos\phi & 0\\ \cos\phi & \sin\phi & 0\\ 0 & 0 & 1\end{bmatrix} \tag{9.4}$$

The inverse velocity kinematics equation of the holonomic robot is then:

$$\begin{bmatrix}\dot\theta_1\\ \dot\theta_2\\ \dot\theta_3\end{bmatrix}=\mathbf{J}_m^{-1}\begin{bmatrix}\dot x_m\\ \dot y_m\\ \dot\phi\end{bmatrix}=\mathbf{J}_m^{-1}\mathbf{R}\begin{bmatrix}\dot x\\ \dot y\\ \dot\phi\end{bmatrix}=\frac{1}{r}\begin{bmatrix}\cos(\phi+\beta_1) & \sin(\phi+\beta_1) & L\\ \cos(\phi+\beta_2) & \sin(\phi+\beta_2) & L\\ \cos(\phi+\beta_3) & \sin(\phi+\beta_3) & L\end{bmatrix}\begin{bmatrix}\dot x\\ \dot y\\ \dot\phi\end{bmatrix} \tag{9.5}$$

Therefore its forward velocity kinematics equation is:

$$\begin{bmatrix}\dot x\\ \dot y\\ \dot\phi\end{bmatrix}=r\begin{bmatrix}\cos(\phi+\beta_1) & \sin(\phi+\beta_1) & L\\ \cos(\phi+\beta_2) & \sin(\phi+\beta_2) & L\\ \cos(\phi+\beta_3) & \sin(\phi+\beta_3) & L\end{bmatrix}^{-1}\begin{bmatrix}\dot\theta_1\\ \dot\theta_2\\ \dot\theta_3\end{bmatrix} \tag{9.6}$$

The typical robot design (e.g. Figure 9.3) uses the symmetric configuration: $\beta_1=30^o$, $\beta_2=150^o$ and $\beta_3=270^o$.

9.2.2 Non-holonomic robots

Figure 9.7 shows the typical geometry of a non-holonomic robot. Its inverse velocity kinematics can be obtained from equation 9.3. From the figure, for wheels 1 and 2: $\beta_1=0^o$ and $\beta_2=180^o$. The third wheel is a free-spinning caster. Since there is no driven third wheel the last row of $\mathbf{J}_m^{-1}$ should be deleted. Therefore equation 9.3 simplifies to:

$$\begin{bmatrix}\dot\theta_1\\ \dot\theta_2\end{bmatrix}=\mathbf{J}_m^{-1}\begin{bmatrix}\dot x_m\\ \dot y_m\\ \dot\phi_m\end{bmatrix}=\frac{1}{r}\begin{bmatrix}-\sin(0) & \cos(0) & L\\ -\sin(\pi) & \cos(\pi) & L\end{bmatrix}\begin{bmatrix}\dot x_m\\ \dot y_m\\ \dot\phi_m\end{bmatrix}=\frac{1}{r}\begin{bmatrix}0 & 1 & L\\ 0 & -1 & L\end{bmatrix}\begin{bmatrix}\dot x_m\\ \dot y_m\\ \dot\phi_m\end{bmatrix} \tag{9.7}$$

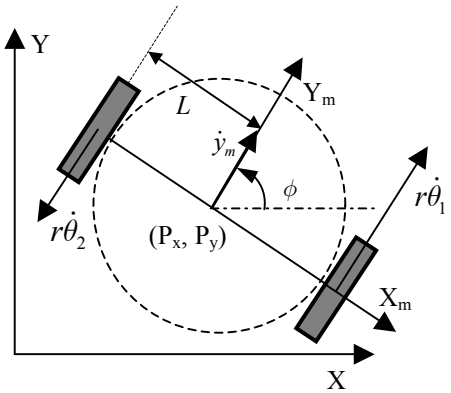


Figure 9.7 Geometry of a typical non-holonomic robot.

From equation 9.7, and the geometry shown in Figure 9.7, we can see that the robot cannot move in the X_m or sway direction (*i.e.* this DOF cannot be controlled) so it is by definition a non-holonomic robot. The Roomba® robot employs this design.

Similarly, the inverse velocity kinematics of the robot in world coordinates can be obtained by substituting $\beta_1=0^o$ and $\beta_2=180^o$ into (9.5) as follows:

$$\begin{bmatrix}\dot\theta_1\\ \dot\theta_2\end{bmatrix}=\mathbf{J}_m^{-1}\mathbf{R}\begin{bmatrix}\dot x\\ \dot y\\ \dot\phi\end{bmatrix}=\frac{1}{r}\begin{bmatrix}\cos(\phi+0) & \sin(\phi+0) & L\\ \cos(\phi+\pi) & \sin(\phi+\pi) & L\end{bmatrix}\begin{bmatrix}\dot x\\ \dot y\\ \dot\phi\end{bmatrix}=\frac{1}{r}\begin{bmatrix}\cos\phi & \sin\phi & L\\ -\cos\phi & -\sin\phi & L\end{bmatrix}\begin{bmatrix}\dot x\\ \dot y\\ \dot\phi\end{bmatrix} \tag{9.9}$$

Since the inverse Jacobian matrix is not a square matrix we can't obtain the Jacobian matrix from it by matrix inversion. However, the Jacobian matrix can be obtained from the robot's geometry and basic kinematics. From the geometric relationship drawn in Figure 9.7, we have:

$$\begin{bmatrix}\dot x\\ \dot y\\ \dot\phi\end{bmatrix}=\begin{bmatrix}\cos\phi & 0\\ \sin\phi & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}\dot y_m\\ \dot\phi\end{bmatrix} \tag{9.9}$$

From Figure 9.7 and basic kinematics, we have:

$$\begin{bmatrix} \dot{y}_m \\ \dot{\phi} \end{bmatrix} = r \begin{bmatrix} 1/2 & -1/2 \\ 1/L & 1/L \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{9.10}$$

Substituting (9.10) into (9.9) gives:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = r \begin{bmatrix} \cos\phi & 0 \\ \sin\phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & -1/2 \\ 1/L & 1/L \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = r \begin{bmatrix} \cos\phi/2 & -\cos\phi/2 \\ \sin\phi/2 & -\sin\phi/2 \\ 1/L & 1/L \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{9.11}$$

Equation (9.11) is the forward velocity kinematics of this non-holonomic robot in world coordinates.

9.3 Geometric Modelling of Robots and Obstacles

Before a path planning method can be created it is necessary to model the robot and the obstacles in its environment. Since the robot will be moving across a planar surface (i.e. the floor) it is only necessary to model the geometry of the robot and obstacles by approximating their projections onto the plane of motion. The geometric models do not need to be exact but they must enclose the true geometry in order to prevent collisions. Polygons are typically used to model the obstacles. The robot is often modelled as a disk. An example showing a robot and three obstacles is given in Figure 9.7a. An important problem when path planning is determining which paths from the robot's current position to its goal position are traversable. For the path shown in Figure 9.7a the answer is difficult for a computer to obtain. The problem becomes easier if we shrink the disk model of the robot down to a point and simultaneously expand every obstacle. To expand each obstacle, every point along its perimeter is expanded by the radius of the disk. If the expanded obstacles intersect with one another, then the point representing the robot cannot traverse between those obstacles, and it becomes easy to determine that this path is not free for the robot. See Figures 9.7b and 9.7c for a depiction of this process.

The disadvantage of using a disk model of a robot is that the expanded obstacles have complex shapes. A simplified, conservative solution is to model the disk-shaped robot by an enclosing square. The diameter of the disc becomes the length of each side of the square. This concept is illustrated in Figure 9.8. Note that the expanded obstacles are much simpler shapes than in Figure 9.7. This simplified expansion method will be used as the first step in the path planning methods that follow.

9.4 Path Planning by Visibility Graphing

In this section we examine the problem of finding the shortest path between the robot's starting location and its goal location amidst static polygonal obstacles, using a method known as visibility graphing. This method assumes that the location and shape of all obstacles are known in advance. It also assumes that the robot and goal locations are also known[1]. A visibility graph is a network of paths that consists of all possible piecewise linear paths from the starting location of the robot to the goal location. One of these paths is the shortest piecewise linear path to the goal location. A visibility graph is created by connecting line segments between all vertices of the expanded polygonal obstacles, and these vertices and the robot's start and goal locations, provided that the lines are not intercepted by the obstacles. In other words, all vertices that are "visible," or have a free line-of-sight from any given vertex, will be connected to that vertex. Note that the edges of the polygonal obstacles also serve as line segments in the visibility graph. By computing the lengths of each possible path, it is possible to search for the shortest path for the robot to take. This path is optimal in the sense that the shortest path should take the minimum time for the robot to traverse. Two examples of visibility graphs are shown in Figure 9.9. The thin solid lines are the segments of the visibility graph. The thick dotted lines represent the shortest path between the start and goal found by searching through these segments. Computer algorithms for creating the visibility graph and for finding the shortest path are beyond the scope of this course, but may be found in [7] and [8].

[1] We will discuss methods for measuring the robot's location in-class.

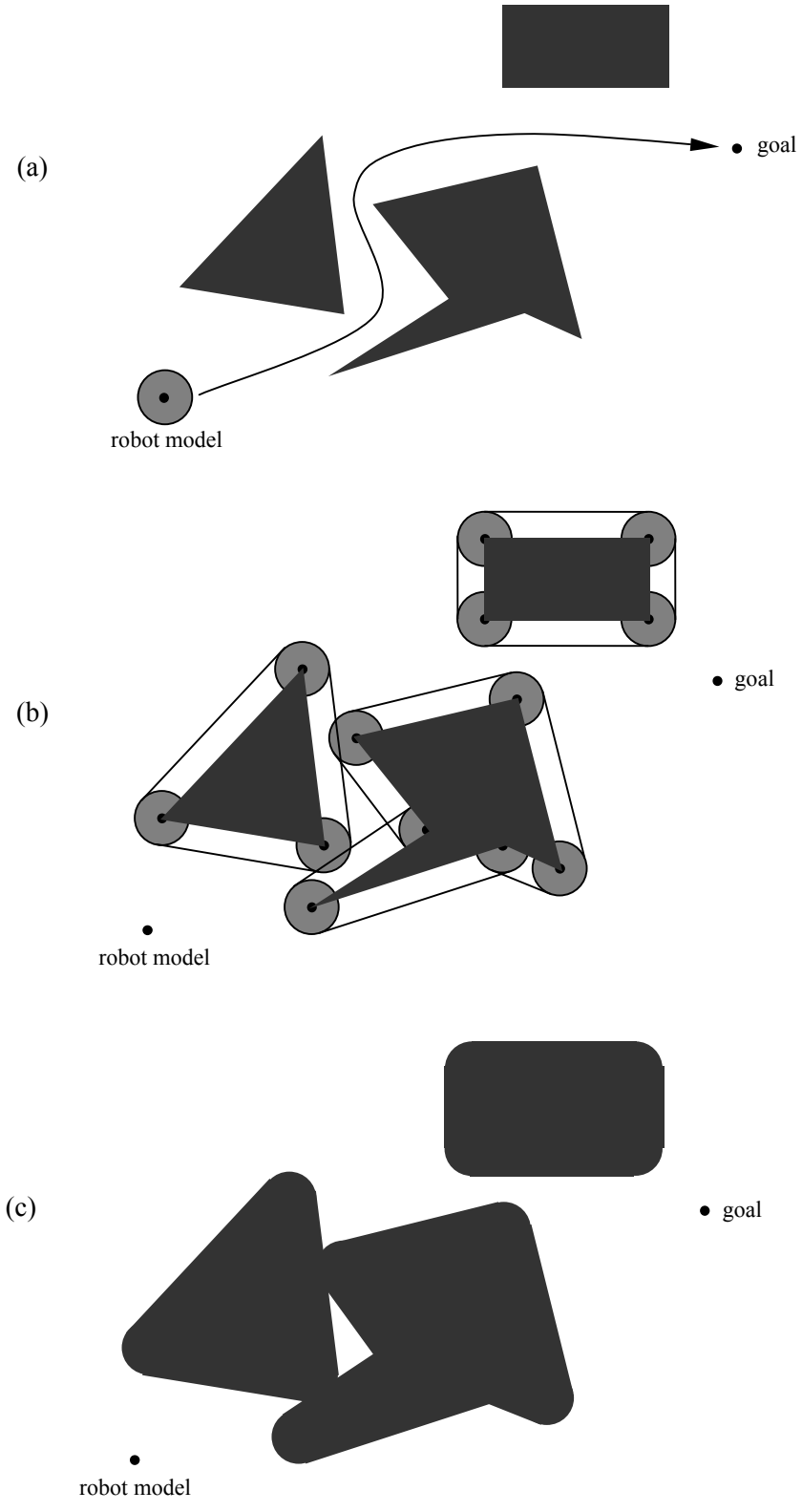


Figure 9.7 Process of expanding polygon obstacles for a robot modelled as a disk: (a) a non-free path to the goal; (b) expanding the obstacles; and (c) expanded obstacles.

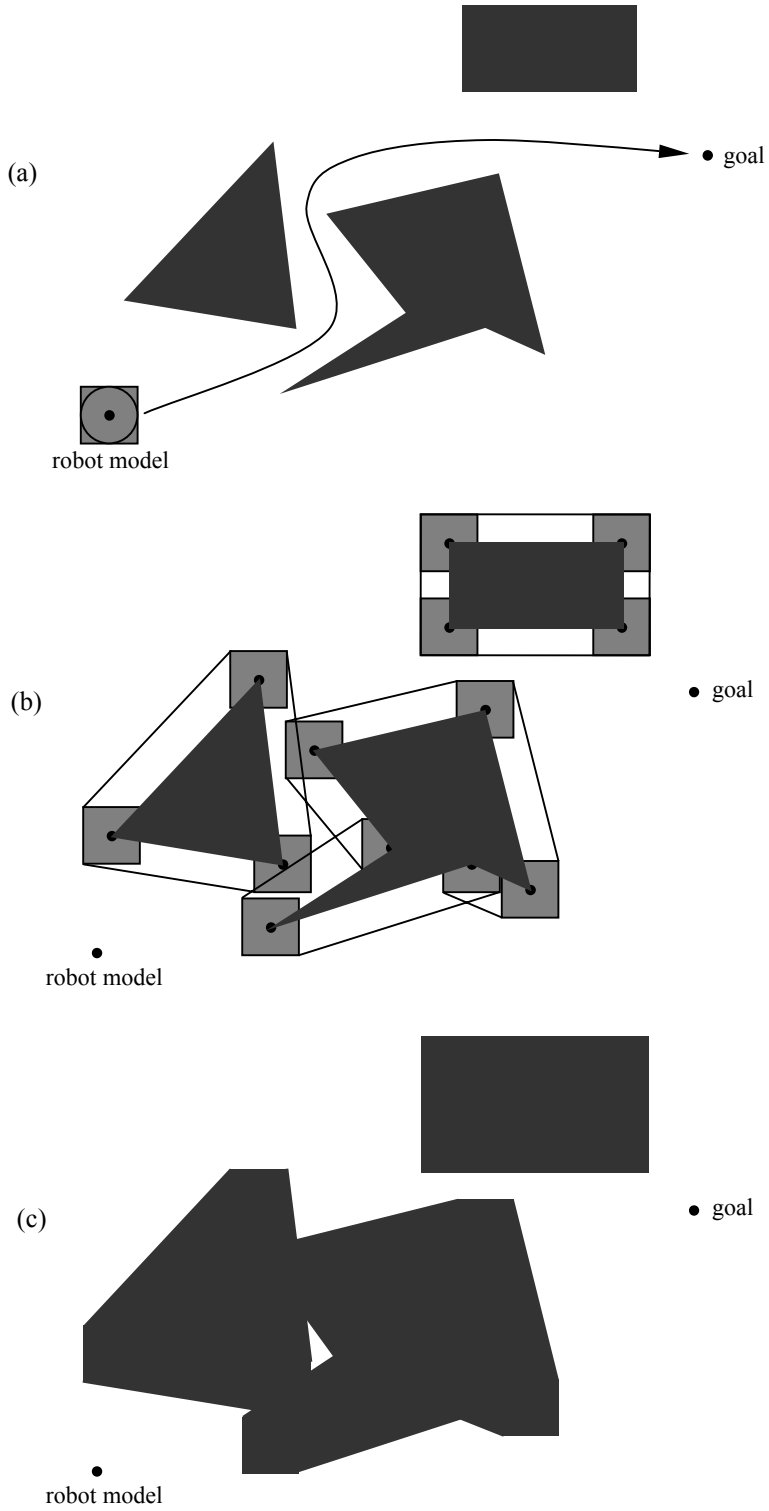


Figure 9.8 Process of expanding polygon obstacles for a robot modelled as a square: (a) a non-free path; (b) expanding the obstacles; (c) expanded obstacles.

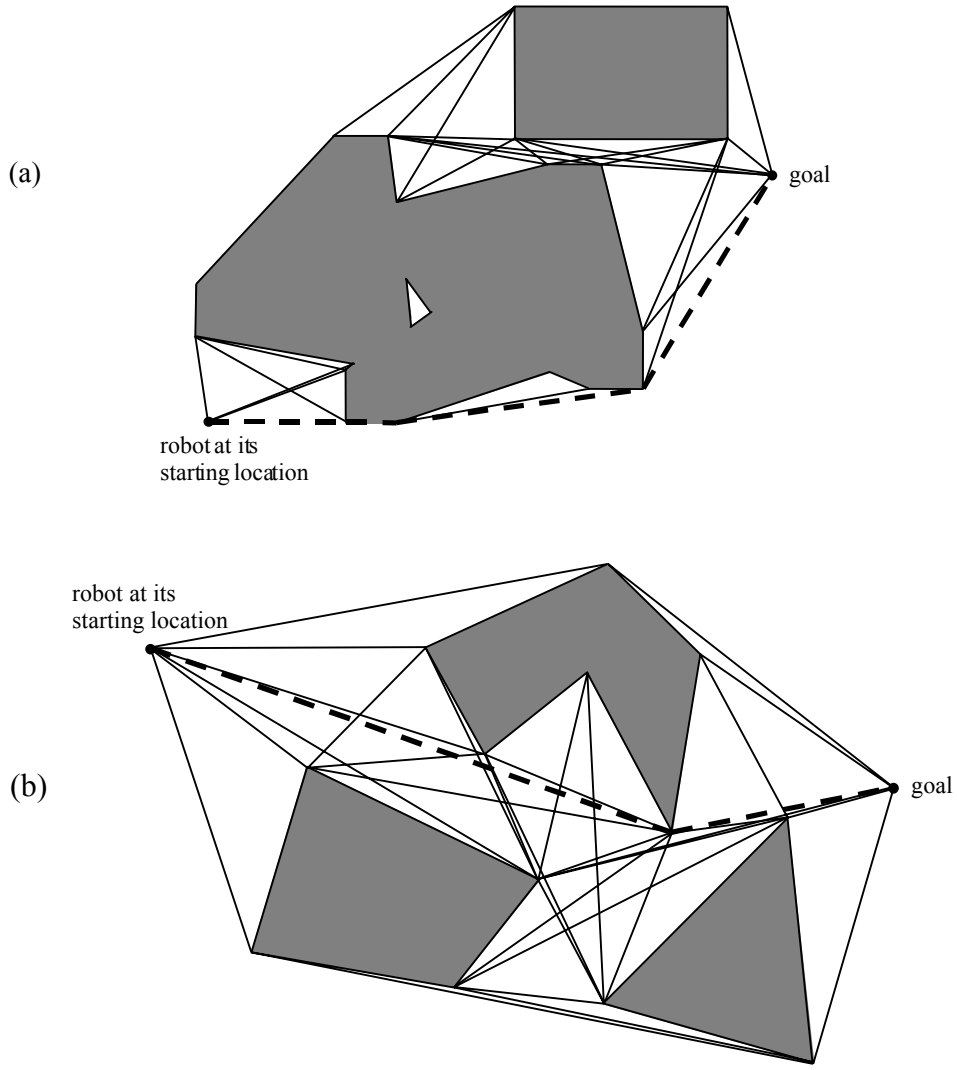


Figure 9.9 Visibility graphs and their shortest paths: (a) for the expanded obstacles from Figure 9.8; and (b) for three obstacles represented as polygons. The shortest paths are shown as dashed lines.

9.5 Path Planning using Simplified Visibility Graphs

The visibility graph-based method can find the shortest piecewise linear path to the goal location but it contains a large number of segments which can make computing the solution a slow process. Luckily, a simplified visibility graph can be created that can be solved much faster and often contains the same shortest path. As before this method assumes that the location and shape of all obstacles are known in advance, and that the obstacles are static polygons. It also involves common tangent line segments and convex hulls. See Figure 9.10 for an example of common tangent line segments for two polygons. The convex hull of a polygon can be obtained by first imagining you have wrapped it with a rubber band, and then filling in the space inside the elastic band. The convex hull eliminates all concave sections of the polygon. An example is shown in Figure 9.11.

The procedure for generating the simplified visibility graph (SVG) is as follows:

1) Add to the SVG the line segments to the obstacle vertices that are also common tangents.
2) For each obstacle:
    (a) If both the starting location and the goal location lie outside the convex hull of the obstacle then add the line segments joining the start and goal locations to the visible

convex hull vertices to the SVG. Add each convex hull edge that connects a visible vertex to a common tangent vertex to the SVG.

(b) If either the starting location or the goal location lies inside the convex hull of the obstacle then add the line segments joining the start and goal locations to the visible obstacle vertices, excluding vertices inside the convex hull, to the SVG. Add each obstacle edge that connects a visible vertex to a common tangent vertex to the SVG.

3) End of procedure.

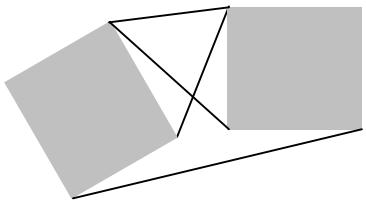


Figure 9.10 The common tangent line segments for two polygonal obstacles.

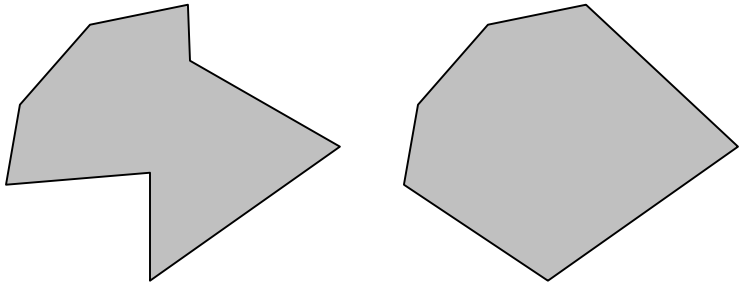


Figure 9.11 Left: a non-convex polygon. Right: its convex hull.

The SVG generating procedure and the shortest piecewise linear path to the goal will be demonstrated in class for the environments shown in Figures 9.12, 9.13, 9.14 and 9.15.

Figure 9.12 First example environment (with two convex obstacles).

Figure 9.12 Second example environment (with one convex and one non-convex obstacle).

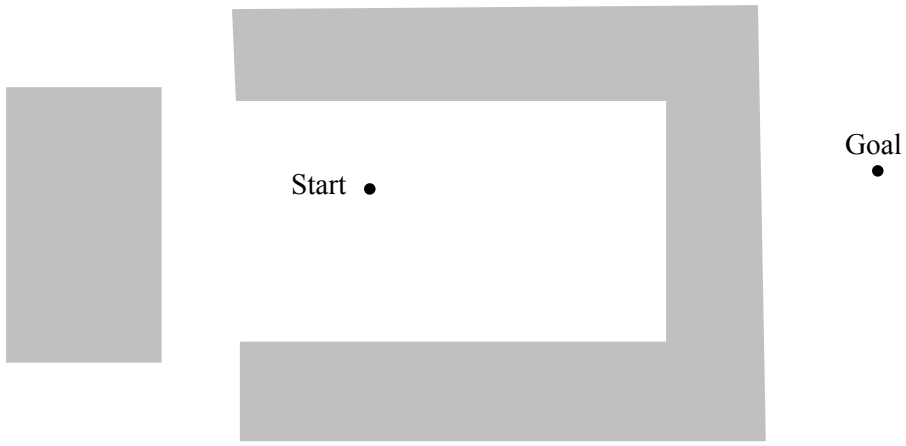


Figure 9.13 Third example environment (with one convex and one non-convex obstacle).

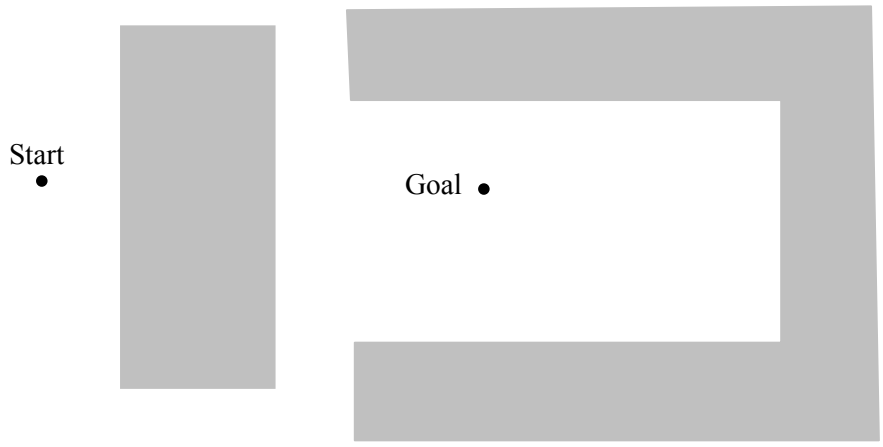


Figure 9.14 Fourth example environment (with one convex and one non-convex obstacle).

9.6 Distance Sensor-Based Path Planning

The visibility graph and SVG methods assume that the shapes and locations of the obstacles are known in advance (e.g. the entire factory layout has been modelled using CAD). Since this is not often true, we will now cover a sensor-based method that can be used to plan paths in unknown environments. It has the further advantage that the obstacles do not have to be static. The disadvantage of this method is the paths it produces are less optimal (i.e. not as short) as those produced by the visibility graph and SVG methods.

Many mobile robots are equipped with ultrasonic, laser and/or infrared sensors that can measure the distance from the robot to the objects surrounding it. For example, the Roomba® robot uses infrared sensing to follow walls. The method presented in this section assumes the robot is equipped with a distance sensor with a 360^o view, and will be termed the Distance Sensor-Based (DSB) method. Rather than attempting to plan the entire path, the robot only plans the direction it should move next based on the information that the sensor gathers about its immediate (local) environment. The robot continuously updates the short-term plan for the path as it moves and obtains new sensor information.

The DSB method [8, 9, 10] uses two modes of motion: 1) goal seeking mode, and 2) obstacle boundary following mode. The robot begins by moving towards the goal, based on the distance sensor information and the pre-defined goal location. This motion can lead to a local minimum condition where to move in any direction would require the robot to move away from the goal. When this is detected the robot switches to the boundary following mode. It switches back to the goal seeking mode when the distance from the robot to the goal is less than the distance from the local minimum point to the goal. A detailed flowchart of the DSB method is presented in Figure 9.15. Note that the function $d(a, b)$ returns the distance between points a and b.

When the robot has detected an obstacle to the goal, the robot uses a structure known as the local tangent graph in order to choose the locally optimal direction to follow. In Figure 9.15 and the example shown in Figures 9.16-9.18, the robot's current location is denoted as r and its goal location is denoted as g. The local tangent graph is constructed based on the edges that the robot can see within its sensor range of radius R. If the sensing range is large enough to include the entire surface of all of the obstacles then the lines through the graph vertices, O_i, will be tangential lines and the DSB method can produce similar results to the SVG method. This is not true for smaller sensing ranges, and the optimality of the DSB method decreases with decreasing sensor range.

After finding the graph vertices, the computer calculates the distances between the robot and each vertex, and the distance between each vertex and the goal. Based on these distances, the shortest path can be chosen. In Figure 9.16, since $d(r, O_1)+d(O_1, g)$ is the shortest of the four distances, the robot would advance towards $r_{next}=O_1$. The robot would travel along this path until the sensor information gets updated, at which time the robot would update its course accordingly.

In Figure 9.17, it can be seen that all of the vertex points available to the robot would force the robot to move away from the goal. This indicates that the robot has reached a local minimum, so it switches to boundary-following mode.

Figure 9.18 shows the point where the robot would return to goal seeking mode since $d(r,g)<d(r_{prev}, g)$. In this example, since nothing else is in front of it, the robot will move directly to the goal as shown by the dotted line.

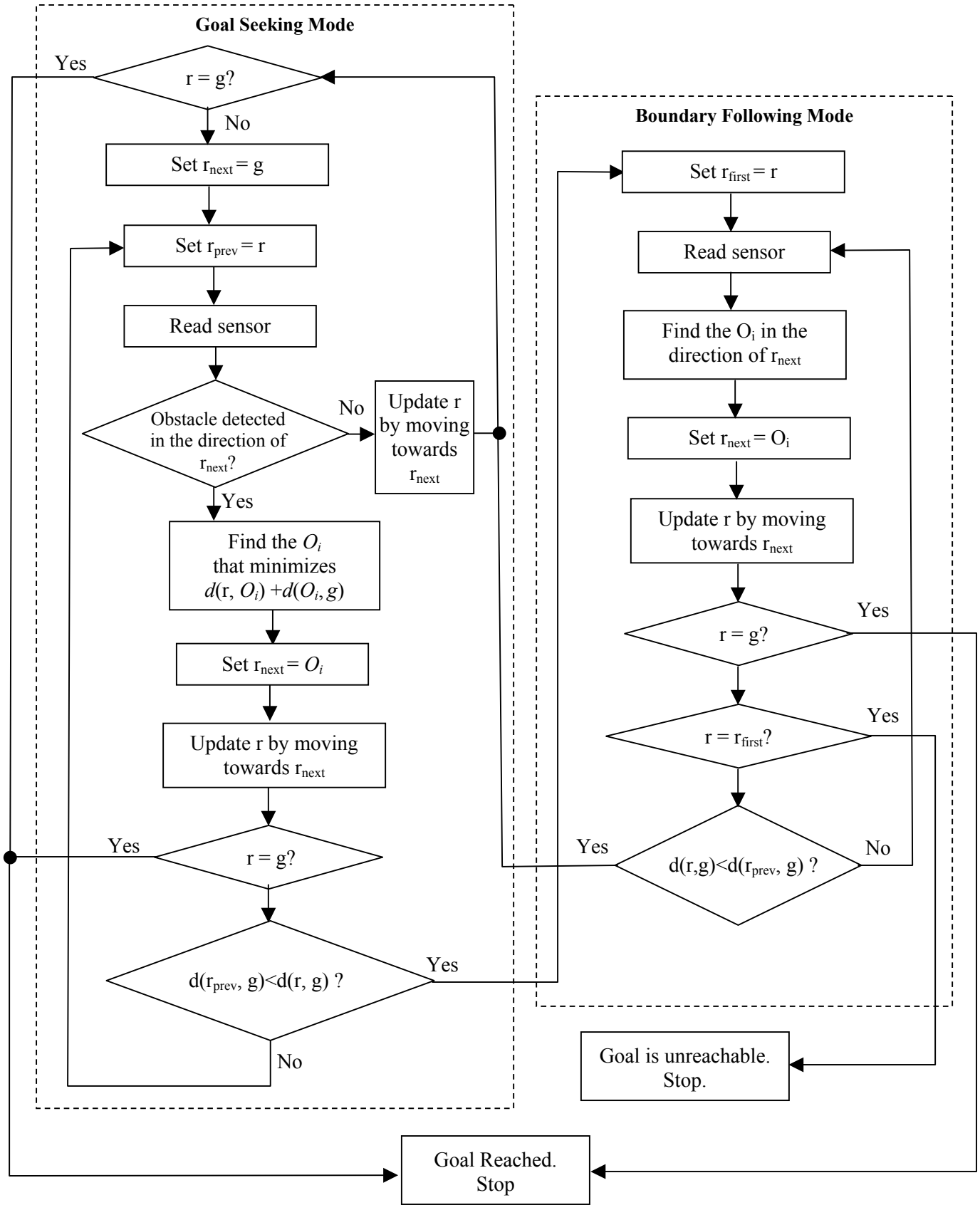


Figure 9.15 Flowchart for the DSB path planning method.

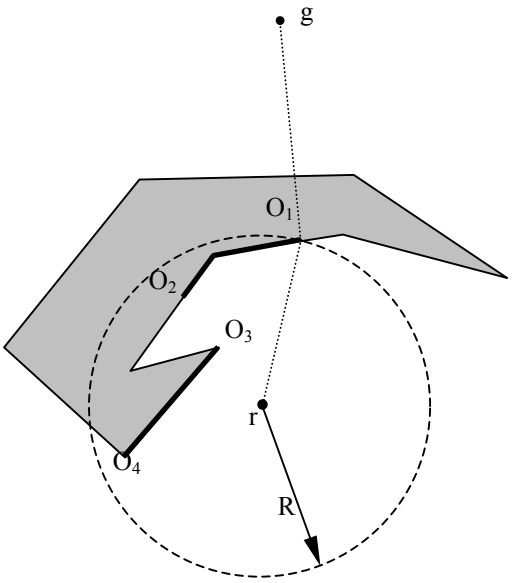


Figure 9.16 Robot operating under goal seeking mode.

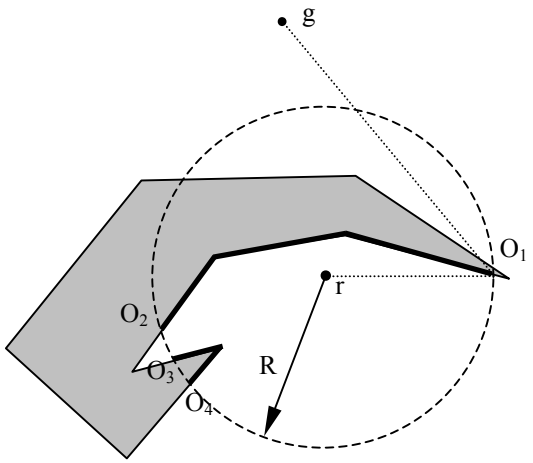


Figure 9.17 Robot switches to boundary following mode due to local minimum.

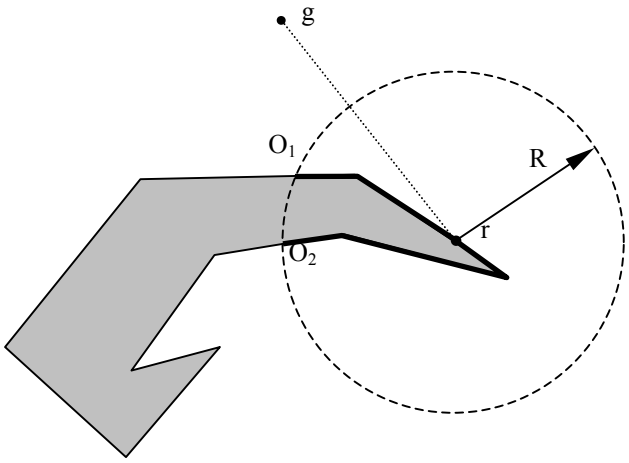
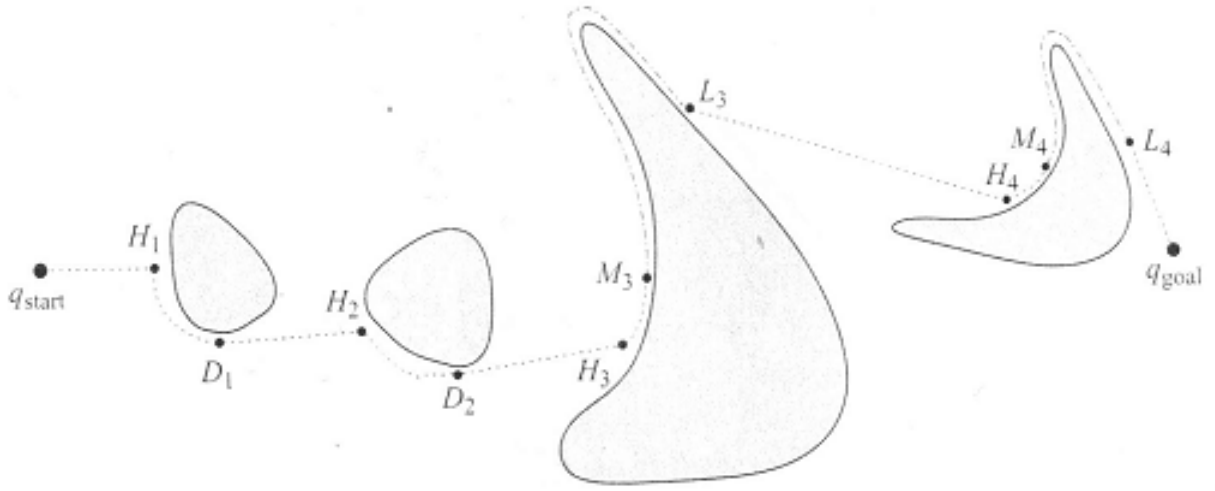


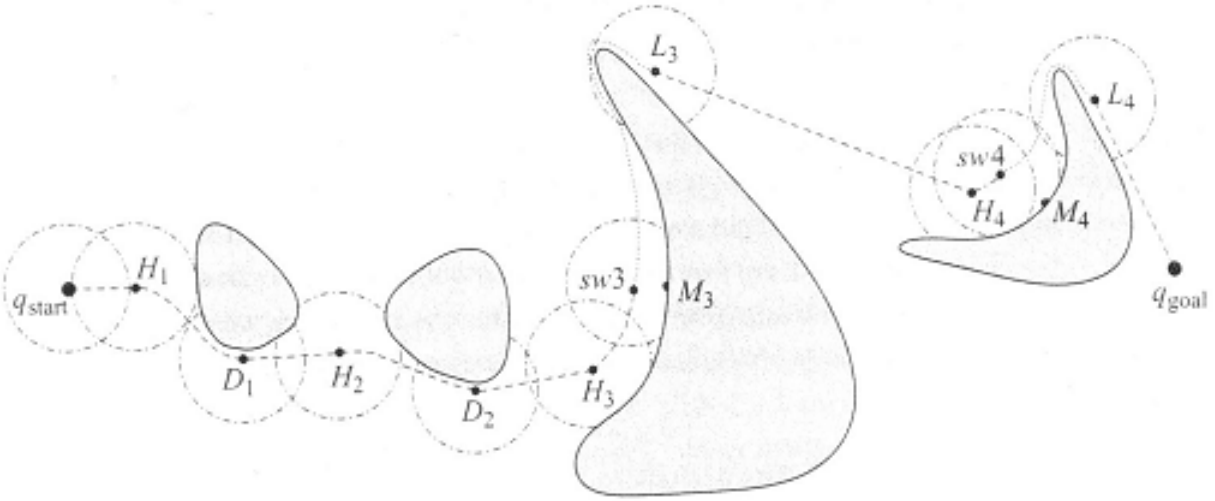
Figure 9.18 Robot returns to goal seeking mode.

Depending on the sensor's type, its detection range can be anywhere from a few millimeters to several metres. The impact of the sensing range has a significant effect on the shortness of the robot's path. Three examples are shown in Figure 9.19 [10]. In this figure the robot's starting location is labeled q_{start} and its goal location is labeled q_{goal}. Its path is shown by the dashed line. In Figure 9.19a the sensor has a very small range and the robot must come very close to the obstacle before it can detect it. It first detects the first obstacle at point H_1, and departs it at point D_1. With the third obstacle it encounters a local mimimum at point M_3, switches to boundary following mode until point L_3, where it leaves the obstacle. A similar pattern occurs with the fourth obstacle before the robot finally reaches the goal location. In Figure 9.19b, the sensing range is large enough to detect the obstacles sooner. This allows the DSB planning method to create a straighter and shorter path than in Figure 9.19a. In Figure 9.19c the sensing range is large enough to detect all of the obstacles, except when the sensor's view is blocked by a closer obstacle (for example when the robot is at its starting location it cannot detect the fourth obstacle since it is hidden behind the third one). This large sensing range provides the major advantage of avoiding the local minima. The robot moves under goal seeking mode in a very efficient fashion until it reaches the goal. Additional examples will be provided in-class.

(a)



(b)



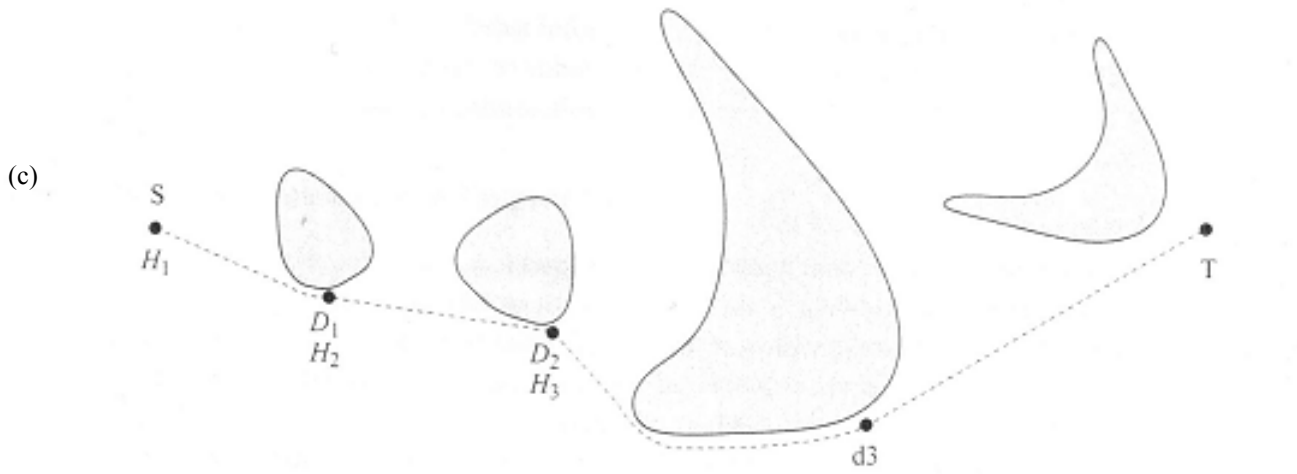


Figure 9.19 Path generated by DSB method with: (a) small sensing range, (b) medium sensing range, and (c) large sensor range [10].

9.7 Implementation of Mobile Robot Motion Control

9.7.1 For a holonomic robot

For the visibility graph and SVG methods, the path may be determined before the robot needs to move (*i.e.* it can be computed offline). To move the robot a trajectory is needed. This requires computing the Cartesian space velocities every Δt seconds (typically $\Delta t \approx 0.05$ seconds) is also required. These can be obtained for the segments of the path using the methods described in Chapter 7. If the robot should follow the path exactly then it will have to stop momentarily at each via point (recall section 7.4). If continuous motion is desired then the deviations due to finite robot acceleration and deceleration must be considered during path planning. Finally, the wheel velocity commands are obtained from the Cartesian velocities using equation 9.5, and transmitted every Δt seconds to the wheel motor controllers in order to move the robot to its goal.

For the DSB method, assuming the sensing range is at least as large as in Figure 9.19b, the path is relatively smooth so continuous Cartesian space motion that approximates the path should be possible. The path cannot be followed exactly since the desired Cartesian velocities are computed online and cannot be achieved due to time delays and finite motor acceleration. The time delays are mainly due to the sensor and the computing hardware. As before, the wheel velocity commands are obtained from the Cartesian velocities using equation 9.5, and transmitted every Δt seconds to the wheel motor controllers in order to move the robot to its goal.

9.7.2 For a Typical Non-holonomic Robot

The motion control of this type of robot is always more difficult than with a holonomic robot since the velocity in the X_m direction is always zero (recall Figure 9.7). This means the robot must be carefully steered to stay on or close to the path. If it is desired that the robot stay on the path with the visibility graph and SVG methods then the robot must stop at each via point, execute a pure rotation (*i.e.* move with $\dot{\theta}_1 = \dot{\theta}_2$) until the necessary orientation angle is reached, and then move along a straight-line until it reaches the end of the path segment. If strictly following the planned path is not a requirement then continuous motion is possible by starting the rotations prior to each via point, just as we do when driving a car.

For the DSB method, the difficulties encountered with a holonomic robot still apply to a non-holonomic robot. The approximate nature of a non-holonomic robot's motion is worsened by the fact that changing direction online takes more time than with a holonomic robot since it requires steering to happen beforehand.

9.7.3 Real World Problems

So far in this chapter we have assumed the wheels do not deform and roll without slipping. Neither of these assumptions is true in the real world. It has also been assumed that the motor controllers follow the commanded velocities perfectly which is similarly impossible in real life. Of these problems, wheel slip is typically the most significant, but can be limited by limiting the commanded wheel accelerations.

Acknowledgements

The important contributions of Lingqi Zeng and Heather Ker to the contents of this chapter are gratefully acknowledged.

References

1. http://www.irobot.com.
2. http://www.ccsrobotics.com.
3. http://www.kornylak.com.
4. K. Watanabe, "Control of an Omnidirectional Mobile Robot," Proceedings of the 1999 International Conference on Knowledge-based Intelligent Electronic Systems, pp. 51 – 60, 1999.
5. R. Rojas and A. Forster, "Holonomic Control of a Robot with an Omni-directional Drive," http://www.mindraces.org/public_references/idsia_publications/ omnidrive_kiart_preprint.pdf.
6. J. Mireles, "Kinematic Models of Mobile Robots," http://arri.uta.edu/acs/jmireles/ Robotics/KinematicsMobileRobots.pdf.
7. J. O'Rourke, Computational Geometry in C, 2nd edition, Cambridge University Press, 1999.
8. I. Kamon, E. Rimon and E Rivlin, "TangentBug: A Range-Sensor-Based Navigation Algorithm," International Journal of Robotics Research, Vol. 17, pp. 934-953, 1998.
9. M. Lanthier, "Goal-Directed Navigation," http:// www.scs.carleton.ca/~lanthier/teaching /COMP4900A/ Notes/6%20-%20Navigation.pdf.
10. H. Choset, K. Lynch, S. Hutchinson, G. Kantorm W. Burgard, L. Kavraki and S. Thrun, Principles of robot motion : theory, algorithms, and implementation, MIT Press, 2005.