Chapter 3

Sensing

Chapter 2 provided ways to move a mobile robot. The coming chapter introduces the robot's "eyes and ears". Without sensors, the robot is completely blind to the world. Motion commands can be sent out, but no information would return to the robots brain. As mentioned in Chapter 1, the robot makes decisions based on its internal information state (I-state). The I-states will be constructed from three sources of information:

1. **Prior Knowledge:** Kinematic models, control system parameters, noise models, wheel radii, initial position, and so on.

2. **Command History:** The memory of commands that have been applied from the start of the task up to the current time.

3. **Sensor Observations:** Data that is in direct response to physical stimuli during task execution.

The third source is the focus of this chapter. Information from all three sources will be combined to make powerful filters in Chapter 5. Section 1.4 introduced the idea of a hypersensor that could perfectly measure everything about the world. It should become clear in this chapter why this is impossible with sensing hardware. It is furthermore not compatible with our goal of making the robot's brain as small as possible (recall the motivation from Section 1.4).

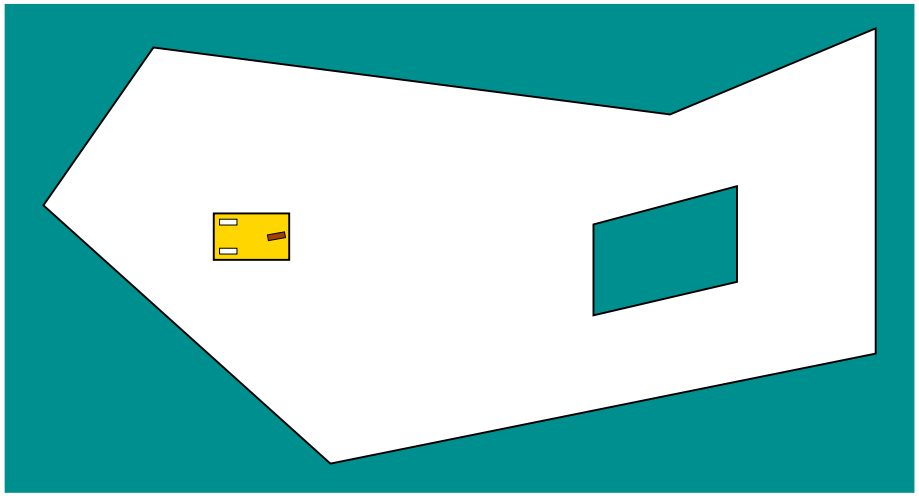


Figure 3.1: A kinematic model will help in predicting where the robot will go, but what sensors are needed to help the robot when obstacles are encountered? What can the robot learn about its environment through sensing? What can it determine about its own configuration?

3.1 Sensing Hardware

To begin considering sensors, think about how to help the robot shown in Figure 3.1. We can make it drive along while using the methods of Chapter 2 to predict where it will travel. What happens if it encounters an unexpected obstacle, which blocks it path? Without any sensors, it might proceed to grind into the obstacle while forcing its wheels to slide. The resulting final position and direction would obviously be much different than predicted by a transition equation (such as (2.19)).

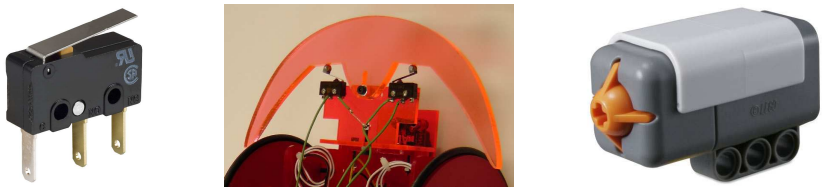
3.1.1 Observing obstacles in the robot's environment

In mobile robotics, a *contact sensor* (or *boundary sensor*) is often placed on the edge of the robot, facing the part that is likely to hit an obstacle. At any time, the sensor provides only one of two possible outputs:

1. **Clear:** The sensor is not pressed by the obstacle.

2. **Blocked:** The obstacle is pressing against the sensor.

A cheap and easy way to implement this sensor is using a limit switch, as shown in Figure 3.2(a). When a force is applied to the arm of the sensor, it retracts and presses a button. The effect is to complete the electrical circuit. Figures



(a) (b) (c)

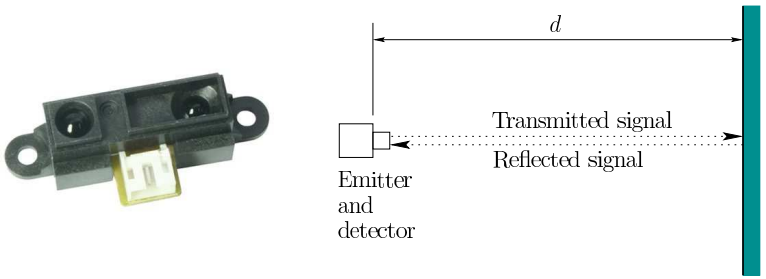
Figure 3.2: Contact sensors: (a) A limit switch closes the electrical circuit when sufficient force is applied to the arm. (b) A bumper attached to the front of a SERB mobile robot built at the University of Illinois. The bumper pivots about a center pin, enabling it to press one of two limit switches, depending on whether the robot hits the obstacle from the left or right corner. (c) The contact sensor from the Lego Mindstorm NXT robotics kit.

3.2(b) and 3.2(c) show examples that use limit switches on real robots. Note that the sensor needs to be placed in any location that might hit an obstacle. For an omnidirectional robot (imagine bumper cars at the carnival), it would be good to place contact sensors all around its boundary. Information regarding the robot's orientation with respect to the boundary can even be inferred based on which particular limit switch was pressed.

Using the contact sensor, the robot will know when it is time to try a different command. For example, it can change to reverse after driving forward into a wall. The observation of the blocked output can cause a transition to the next command in a discrete-time control system, which was introduced in Section 2.3. The transition is caused by an event (collision!), rather than some elapsed Δt.

What if the obstacle (or robot) is delicate? Rather than colliding, it would be better to have a sensor that indicates whether the robot is within a few centimeters of collision. An *infrared (IR) proximity (or range) sensor* (Figure 3.3(a)) adequately provides this information. The idea is to direct an infrared light at the obstacle and judge the distance based on the intensity of the reflected light. Figure 3.3(b) shows the general reflection principle. This makes assumptions about the reflectivity properties of the typical obstacle. By using infrared light and coding the beam, interference with other light sources is avoided. Another reflection-based sensor for detecting obstacle proximity is the *sonar*, which emits a high-pitched sound and measures the time of the return echo. This is called a *time of flight* technique, which is equivalent to the echolocation principle used by bats. Based on the round-trip travel time and the speed of sound, the distance can be estimated. If the sensor output is noisy or inaccurate, it may nevertheless be useful for producing a binary output. If the calculated distance falls below a predetermined threshold, the sensor may report that the obstacle is "close"; otherwise, it is "far".

IR sensors and sonars are useful for simple detection of obstacle proximity, but they are less reliable for measuring the distance to the obstacle. Two reasons are



(a) (b)

Figure 3.3: (a) The SHARP GP2D120 (around $12 US), a typical infrared range sensor used on mobile robots. Its published range is between 4cm and 30cm. (b) Most range sensors work by the reflection principle. A signal is transmitted to the obstacle, and the distance is judged based on the reflected signal.

interference with other signals and assumptions on obstacle reflectivity that are difficult to ensure in practice. Fortunately, there are much better sensing methods for precisely estimating the distance. The reflection principle is used again, but this time with a laser. A laser is a *coherent* light source, which means that virtually all emitted light waves have the same frequency and phase. A typical LED produces light waves of the same frequency, but their phases vary. For a laser, the peaks and valleys of the waves are aligned as they propagate through space. The laser source is also *collimated*, which means that it is focused into a narrow beam that hits a tiny spot on the obstacle.

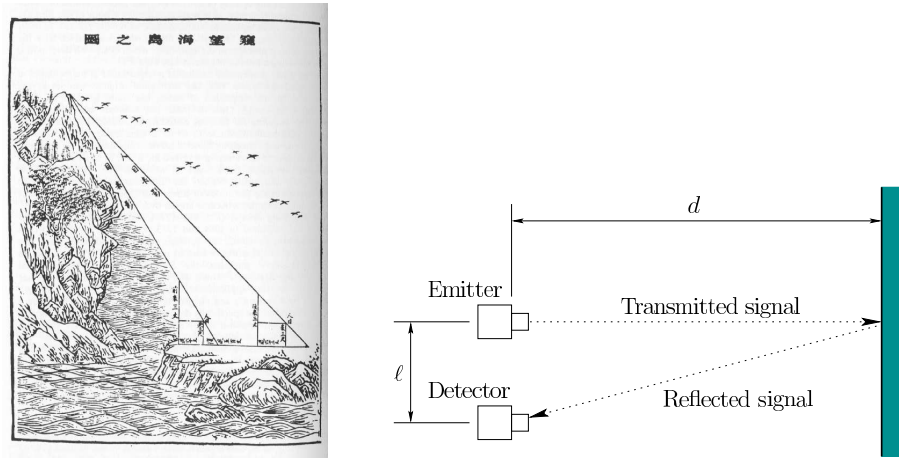
Using a laser, the distance to the wall can be measured by one of three techniques:

1. **Time of flight:** This is the same principle as for the sonar; however, this technique is expensive for lasers because light travels roughly 3cm in one nanosecond; extremely accurate timing electronics are needed.

2. **Phase shift:** The beam is emitted using amplitude modulation (as in AM radio) and then the difference in phase between the emitted and reflected waves can be easily detected when they are superimposed.

3. **Triangulation:** This ancient principle (Figure 3.4(a)) uses an alternative line of sight to find where the laser hits. This yields a triangle with enough known quantities to calculate the distance (Figure 3.4(b)).

The phase-shift approach revolutionized mobile robotics research in the 1990s when the SICK laser scanner (Figure 3.5) became available. Though expensive, it produced rapid, dense, and accurate distance measurements at levels that were unimaginable a decade earlier, using only sonars and IR sensors. The laser is shined into a spinning mirror that reflects the beam in many directions. In 2009,



(a) (b)

Figure 3.4: (a) In ancient China over 1700 years ago, Pei Xiu developed a triangulation method for measuring the distance to unreachable places. (Figure courtesy of Wikipedia.) (b) By comparing a sensed feature from two lines of sight, triangular geometry yields enough constraints to infer the distance. This requires calibration, which in this example means that the following are accurately known: 1) the spacing ℓ between the emitter and detector; 2) the emitter and detector are perpendicular to the transmitted signal; and 3) the incoming angle of the reflected signal. The reflected signal angle is the only variable, which depends on the obstacle distance.

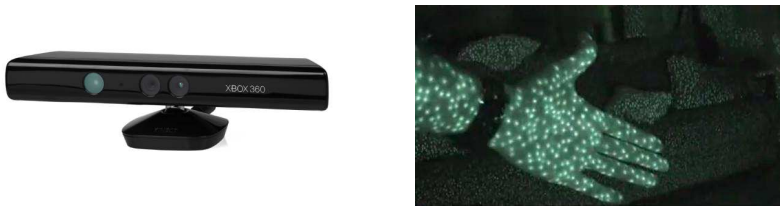
the SICK LMS 100 had a cost of $5000 (USD), and could scan over a 270 degree range at 50 times a second, with a measurement every 0.25 degrees.

The SICK laser and its variants provide distance measurements along a single plane, which is usually horizontal in mobile robotics. Scanners also exist that provide measurements in both the horizontal and vertical directions at the same time. The principle of *structured light* has been used for decades as a way to achieve this. The idea is to shine a light pattern, such as stripes or a grid, into the environment and the use a camera to determine the point distances via triangulation.

The next revolution in mobile robotics came with the appearance of the Kinect sensor (Figure 3.6), which was mass produced for tracking human body movement in video games. Therefore, it is very cheap, costing around $100 (USD). The Kinect projects a structured light grid in the infrared range so that it is less sensitive to ordinary lighting conditions and is invisible to the user. It uses a camera and triangulation to determine the point distances and even associates the RGB color values observed with the camera to each measured point. The Kinect is referred to as an RGBD ("D" for depth) camera, which is a quickly growing category of

Figure 3.5: The SICK LMS 200 laser scanner, available since the 1990s, provided dense, accurate measurements by beaming the laser into a rapidly rotating mirror.

devices and associated research approaches. The cheap availability of such dense measurements has spawned substantial research activity in manipulating *point clouds*. The ROS 3D Point Cloud library from Willow Garage and projects based on it give a sampling of what is possible. With the widespread availability of dense sensor data, we will confront the big brain vs. small brain issue from Section 1.4: Should the robot brain be filled with point clouds that demand heavy processing, or can most of the data be discarded? If the task is to provide a high-fidelity model of the surrounding environment with dense, accurate information, then perhaps yes. However, some robot tasks do not require taking so many measurements in the first place.



(a) (b)

Figure 3.6: (a) The Kinect RGBD sensor provides dense color and depth measurements at low cost. (b) Although to humans it appears to be passive, the Kinect projects infrared dots into the scene to enable triangulation (making it active).

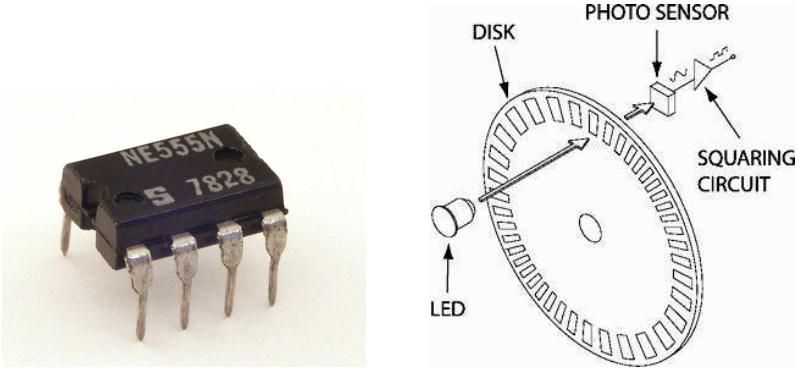
3.1.2 Observing the robot's state

Recall that the models in Chapter 2 are expressed in terms of the robot configuration, which in the planar case is the position x, y and orientation θ. The problem of determining the robot configuration as it moves around is called *localization*, which is presented in Chapter 6 as a filtering problem. Sensor outputs provide inputs to these filters.

What sensors are available to keep track of the configuration as the robot moves? The sensors from Section 3.1.1 could be used to calculate the robot configuration based on which position and orientation is the best "fit" for the observed data. The robot configuration could also be determined using *landmarks* (or *beacons*). For example, a digital camera and computer vision system could be used to recognize landmarks that are placed around the environment. These could be intentionally designed and placed to facilitate robot navigation, or the vision algorithm could find its own environment features to utilize effectively. Another approach is to place RFID tags on the floors and walls to allow simple detection with a small radio. At a minimal extreme, even a simple photoresistor (light-dependent resistor) can provide crude information about the robot configuration. Some parts of the building may be lighter than others. At the other extreme are Global Position Systems (GPS), which provide accurate time and position on the earth by measuring the phase shift in radio signals from satellites, and in some cases, additional base stations. The same principle is used to estimate the location of cell phone users from the phase shift determined at cell phone towers. These systems are useful for outdoor mobile robots, such as an autonomous car. For indoor use, a similar principle can be applied by using ultrawideband (UWB) radio technology. Even using the signal intensities from known WiFi bases has been known to provide rough position estimates in buildings [].

Without directly using information from obstacles or other features in the environment, consider how to determine the robot configuration. A common approach is *dead reckoning*, which is to calculate a future state based on the current states and the transition equation. If the initial configuration is given, then future configurations could be calculated using a transition equation; however, this requires a perfect kinematic model. We can nevertheless consider what sensors would help improve this calculation. If a command is given to the robot over some time interval ΔT, then a *time sensor* should be used to measure the elapsed time. A cheap example is the 555 Timer shown in Figure 3.7(a); there are more 555 timer chips in the world than people! More accurate time measurement is provided by a quartz crystal and a counter (as in a digital wristwatch). If the wheel rotation rate is perfectly calibrated, then its final orientation can be predicted using a timer. If it is not trusted, then an optical wheel encoder (Figure 3.7(b)) can be used to carefully measure the amount of total rotation. This provides useful information, however, it cannot account for errors in the wheel radius or shape, imperfections on the floor, slippage, and so on.

Another source of useful information is an *inertial measurement unit* (IMU) (Figure 3.8), which has been used for decades in aircraft to estimate their orientation. In recent years, IMUs have been developed using MEMS technology and are widely used in smart phones to measure tilt angles. A *gyroscope* keeps track of the vehicle orientation, relative to the starting orientation. It tends to be accurate over a short time interval, but the inaccuracy grows over time. This is often referred to as *drift error* or more generally, *dead reckoning error*.



(a) (b)

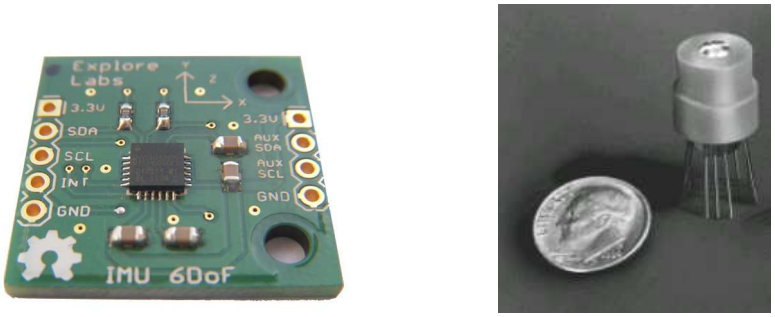
Figure 3.7: (a) The Signetics 555 Timer, one of billions in existence. (b) An optical wheel encoder, to digitally calculate the amount of rotation.

The next concern is how to compensate for such drift. For this, information from the environment is once again needed. For a vehicle that is capable of 3D rotations, the drift error is often corrected using an *accelerometer*, which measures the direction of the gravity vector with respect to the body of the robot. In this way it can correct drift error in all directions except rotations with respect to the "up" direction, which is parallel to the gravity vector. This is often called a *yaw angle*. A *compass* can help in this by sensing the direction toward north. The raw measurements from which to construct a compass are provided by a *magnetometer*, which senses the Earth's magnetic field. The result is a 3D vector that roughly points toward magnetic north.

3.1.3 Communicating with sensors in the environment

Sensing need not occur on board the robot. Using a wireless network, information may be transmitted to the robot from sensors that are scattered around the environment. Some examples are shown in Figure 3.9. For example, if the robot enters a room, an occupancy sensor would report that some motion has occurred. Alternatively, a pressure-sensitive floor mat could indicate that a robot traversed it. If there are people or other robots in the environment, then an additional task of determining *which* entity caused the sensor output arises.

If there are other robots, then they can can transmit their sensor outputs or



(a) (b)

Figure 3.8: (a) Inertial measurement units (IMUs) have been used for decades for tracking aircraft orientation. Nowadays, IMUs are made from microelectromechanical systems (MEMS), resulting in tiny, affordable sensors that are used in smart phones. Pictured is the Explore Labs breakout board for the InvenSense IMU (the black square), which costs around $50 US. (b) The Dinsmore 1490 compass senses eight quantized directions and costs around $14 US.

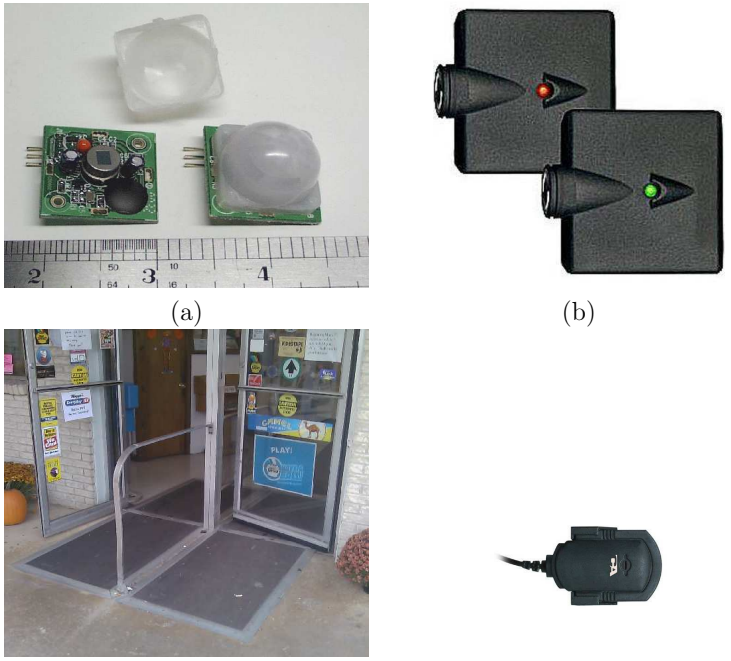
other relevant information to each other. The task might be to collectively explore and map out the environment. Alternatively, they might systematically search a building for a moving fugitive. Imagine playing "hide and seek" with robots. The robots should communicate the status of the search with each other; otherwise, they might search unnecessarily.

With sensors distributed throughout an environment, traditional computer networking issues are extended into the physical world, resulting in a *sensor network*. Each node in such a network could be a static sensor or a mobile robot that carries sensors, resulting in a *mobile sensor network*. Several concerns have been addressed in that context: 1) The network should be robust with respect to failing nodes and the failure to receive messages, 2) power consumption should be minimized, particularly for battery-powered devices, 3) a heterogeneous collection of numerous devices should be able to interface and communicate reliably, 4) an approach to solve a task should use as little communication bandwidth as possible.

3.1.4 General characteristics of sensors

Now that many examples of sensors have been given, it is helpful to systematically identify and characterize their common properties. First of all, it is not even clear how to define a sensor. To be considered a sensor, it seems that the device must be used by a larger system for some form of inference or decision making. The light-dependent resistor (LDR) in Figure 3.10(a) alters the current or voltage when placed in a circuit. It can be considered as a *transducer*, which is a device that converts one form of energy into another; the LDR converts light into an electrical signal. When connected to a larger system, such as a robot, we will happily consider it as a sensor. Figure 3.10(b) shows a complete global position system (GPS) device, which measures position, orientation, and velocity information. As a black box, it produces information similar to the LDR placed into a tiny circuit; however, its operation is much complex because it measures phase shifts between signals emitted by orbiting satellites and performs extensive filtering, the topic of Chapter 5. When connected to a larger system, its precision and error characteristics are much harder to analyze (for example, are trees blocking satellites?). The process occurring inside the sensor is much more complex than for a simple transducer. A sensor could quite easily be more complex than a robot that uses it.

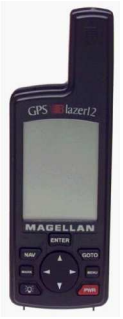
We might take a device that was designed for another purpose and abuse it into being a sensor. For example, the wireless card in Figure 3.10(c) was designed mainly for communications; however, it can also be configured in a larger system to simply serve as a signal meter. It was illustrated in [9] that when used as a



(a) (b) (c) (d)

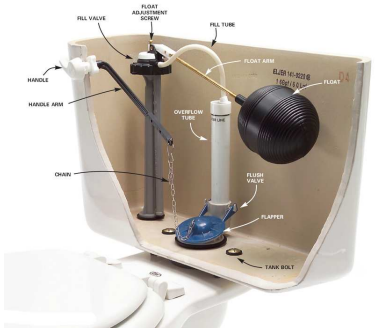
Figure 3.9: Examples of external sensors: (a) A passive IR occupancy detector, often found on ceilings. (b) A Genie garage door detection beam. (c) A supermarket pressure mat, used to determine whether to open the doors. (d) A PC microphone.

(a) Light-dependent resistor



(b) GPS unit

(c) Wireless card



(d) Toilet float mechanism

Figure 3.10: Are all of these "sensors"?.

sensor, it provides powerful localization information. This should cause us to look around and abuse any device we can find into performing as a sensor.

Finally, it seems that the float mechanism in a toilet water tank, shown in Figure 3.10(d), serves as a sensor to determine when to shut off the flow valve. This is perfectly fine as a sensor in a purely mechanical system, but in this book, we consider only sensors that provide input to electronic systems.

Based on these examples, it seems best to avoid a precise definition of a sensor. Roughly, they are devices that respond to external stimuli and provide signals to a larger system. The stimuli that sensor measure fall into several catgories [14]:

1. **Spatial:** Displacement, velocity, acceleration, distance to something, proximity, position, attitude, area, volume, level/tilt, motion detection

2. **Temporal:** Clock, chronometer (elapsed time), frequency.

3. **Electromagnetic:** Voltage, current, power, charge, capacitance, inductance, magnetic field, light intensity, color. These may operate within a circuit or within open space.

4. **Mechanical:** Solid (mass, weight, density, force, strain, torque), fluid (acoustic, pressure, flow, viscosity), thermal (temperature), calories.

5. **Chemical:** Molecular composition, pH, humidity, pollution, ozone, radiation (nuclear).

6. **Biomedical:** Blood flow, heart rate, pressure.

Keep these categories in mind when designing a robotic system. A vast array of physical phenomena can be measured through sensing technologies. Through clever use of unusual stimuli, surprisingly capable mobile robots can be developed with low cost and simple systems.

Most sensors are characterized in terms of a *transfer function*, which relates the possible inputs (phenomena) to the outputs (sensor readings). In Section 3.3, the important notion of a *sensor mapping* is introduced, which can be considered as a generalization and idealization of the transfer function. The transfer function is central in engineering manuals that characterize sensor performance [5, 14].

Several important terms and concepts will be introduced with respect to the transfer function. For simplicity here, suppose that the transfer function is a function $g : \mathbb{R} \rightarrow \mathbb{R}$, and the sensor reading is $g(x)$ for some phenomenon x. Thus, the sensor transforms some real-valued phenomenon into a real-valued reading. The domain of g may describe an *absolute* value or compare *relative* values. For example, a clock measures the absolute time and a chronometer measures the change in time.

The transfer function g may be *linear* in simple cases, as in using a resistor to convert current into voltage; however, more generally it may be *nonlinear*. Since the so-called real numbers are merely a mathematical construction, the domain and range of g are actually discrete in practice. The *resolution* of the sensor is indicated by the set of all possible values for $g(x)$. For example, a digital thermometer may report any value in the set $\{-20, -19, \ldots, 39, 40\}$ degrees Celsius. For a more complex example, a camera may provide an image of 1024×768 pixels, each with 24-bit intensity values.

Whereas resolution is based on the range of g, *sensitivity* is based on the domain. What set of stimuli produce the same sensor reading? For example, for what set of actual temperatures will the digital thermometer read 18 degrees? To fully understand sensitivity in a general way, we study the preimages of sensor mappings in Section 3.3. This is a fundamental source of uncertainty covered in this book.

Additional uncertainty may arise due to lack of *repeatability*. If the sensor is used under the exact conditions multiple times, does it always produce the same reading? Calibration can eliminate systematic (or repeatable) errors to improve sensor accuracy. Recall from Chapter 2 that calibration was critical for using kinematic models of motion. The same is true for sensors. For example, suppose we have purchased a cheap digital thermometer that has good repeatability but is usually inaccurate by several degrees. We can use a high-quality thermometer

(assumed to be perfect) to compare the readings and make a lookup table. For example, when our cheap thermometer reads 17 and the high-quality thermometer reads 14, we will assume for ever more that the actual temperature is 14 whenever the cheap thermometer reads 17. The lookup table can be considered as a mapping that is composed with g to compensate for the errors. As another example, a wristwatch is actually a chronometer that is trying to behave as an absolute time sensor. Via frequent calibration (setting the watch), we are able to preserve this illusion.

In the context of robotics, some additional distinctions arise for classifying sensors. Motivated by biological terms, sensors that are used to measure internal characteristics of the robot are often called *proprioceptive*. Examples are the motor speed, wheel angles, tilt angle, and battery voltage. This is in contrast to *exteroceptive* sensors, which measure the surrounding environment. Most of the sensors from Section 3.1.1 are exteroceptive, whereas most from Section 3.1.2 are exteroceptive. However, the distinction between internal and external is not always clear. For example, a laser scanner (such as the SICK LMS 100) could be used to learn about surrounding obstacles or could be used to determine the robot configuration that is consistent with the already-known obstacles. The first use seems proprioceptive whereas the second seems exteroceptive.

Another important sensor distinction is based on whether it emits energy to disturb the environment. Sensors that do this are called *active*. Shining a laser beam on the obstacle to measure its distance is a prime example. The Kinect is a clever example of an active sensor because the projected infrared dots are invisible to humans. Thus, it is secretly active. A sensor that is not active is called *passive*. Examples are a compass, light sensor, and microphone.

3.2 Modeling Physical Phenomena

Recall from Chapter 2 that kinematics describes the geometry of motion without regard to particular physical implementations. This was a powerful idea because numerous robot systems could be modeled with the same equations. We now want to accomplish something similar, which is to describe the *geometry of sensing* without regard to the particular sensor implementations. Since a sensor converts stimuli in the physical world into data, we want to develop a mathematical model of the physical world that is sufficient for predicting the output of an ideally functioning sensor.

This motivates the mathematical definition of a space that accounts for all stimuli or conditions that affect sensor outputs. As mentioned in Section 1.3, this is called the *physical state space*. We will shorten the name to *P-space* for convenience. The mathematical model of a sensor will then be called a *virtual sensor*, which could have many alternative physical implementations. The key idea is to relate the physical world to the sensor output by carefully specifying:

1. The *physical state space* (or *P-space*), in which each physical state is a cartoon-like description of the possible world external to the sensor.

2. The *observation set*, which represents all possible sensor output values or observations.

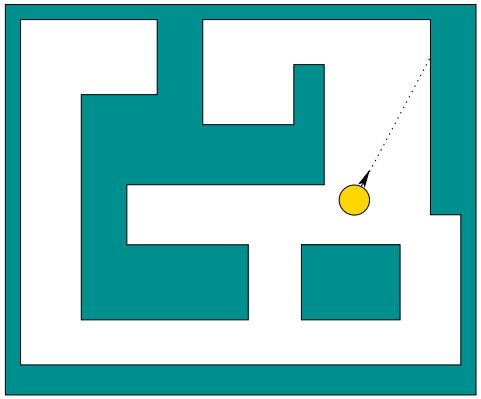


Figure 3.11: A mobile robot is placed in an indoor environment with polygonal walls (more generally called obstacles). It measures the distance to the wall in the direction it is facing.

For each external state in the model, a *sensor mapping* indicates what output or *observation* that the sensor should produce. Each observation arrives in the robot's "brain" and will contribute to its internal, information state (I-state).

It is important to realize in this section that each external, physical state, or P-state, could be extremely large. The P-space, which is the set of all of possible P-states is even larger. Do not worry. This will not present a problem. For most sensors, a tremendous amount of uncertainty arises because the sensor does not observe *everything* about the external world. Understanding how to solve tasks in spite of this uncertainty is fundamental to all of robotics. Additional uncertainty may arise due to sensor noise or calibration errors, but this is considered separately in Section 3.5. There will not necessarily be complete, precise representations of the P-state in the robot's brain (computer). Instead, the P-space is part of the mathematical modeling technique. We are perfectly happy to describe an enormous P-space in which the P-states do not have to be explicitly manipulated in the computer. Instead, the robot will manipulate its own internal I-states, which are hopefully much simpler to manage. The sensors provide a crucial link between P-states and I-states.

To start thinking about P-spaces, consider the scenario shown in Figure 3.11, in which an indoor mobile robot measures the distance to the wall in the direction that it happens to be facing. This could, for example, be achieved by mounting a laser-based distance sensor on the front of the robot. If the sensor is functioning perfectly and reads 3 meters, then what do we learn about the external world? This

depends on what is already known before the sensor observation. Two important questions are:

1. Do we already know the robot's configuration (position and orientation)?

2. Do we have a precise geometric map of all of the walls?

If the robot has both of these already, then it would learn nothing more from the sensor observation.[1]

If the robot's configuration is known but it does not have a map of the walls, then the sensor reading provides information about how the walls are arranged. Admittedly, very little information is provided. It is known, at least, that the robot is not trapped in a closet with maximum diameter less than 3 meters. If the robot instead had a spinning laser, as in a SICK LM100, then it would obtain much more knowledge about the walls in a single sweep of the sensor.

Now suppose that the robot has a complete map of the walls, but not the configuration. In this case, it learns something about the its position and orientation. This is part of the localization problem (Chapter 6).

If the robot has neither the map nor its configuration, then something is learned about *both*. The process of learning both at the same time with multiple motions and observations is referred to as *simultaneous localization and mapping (SLAM)*, the subject of Section 7.5. This relies on both the sensing models of this chapter and the kinematic models of Chapter 2. As the robot moves and gains more observations, algorithms stitch the results together to produce a map using the kinematics model to determine how observations from different configurations are related.

Across all of these cases, the purpose of defining the P-space is to characterize the set of *possible* external worlds that are consistent with each sensor observation and whatever background information is given. If information is already known by the robot and remains static during execution, then it is not included as part of the physical state.

Since the physical state contains both configuration and map information, a common structure frequently appears for P-space. Let $\mathcal{Z}$ be any set of sets. Each $Z \in \mathcal{Z}$ can be imagined as a "map" of the world and each $z \in Z$ would be the configuration or "place" in the map. If the configuration and map are unknown, then the P-space would be the set of all (z, Z) such that $z \in Z$ and $Z \in \mathcal{Z}$. It is important to keep in mind this general structure throughout the book. Typically, z refers to the quantities measured by proprioceptive sensors, and Z corresponds to exteroceptive sensors.

3.2.1 A mobile robot among obstacles

This section develops P-spaces for the case of a mobile robot that moves in the plane among obstacles, which block its motion and obstruct some sensors. Return to Figure 3.11 and consider the simple case of having no obstacles. Let $\mathbf{P}$ denote the P-space, which is the set of all possible P-states. Without obstacles, a reasonable choice is to make

$$\mathbf{P} = \mathbf{X}, \tag{3.1}$$

in which $\mathbf{X}$ is the state space from Section 2.3. In most cases each state $\mathbf{x}$ in $\mathbf{X}$ specified a possible configuration of the robot, except in Section 2.3.3, in which it included velocities. Usually, $\mathbf{X}$ represents the set of all configurations, which are the positions and orientations of the robot. Thus, $\mathbf{x} = (x, y, \theta)$.

A position (x, y) could be any point in the plane, $\mathbb{R}^2$. An orientation θ could be any point in the interval $[0, 2\pi)$. The interval is written with "[" to indicate that 0 is included, and it is written with ")" to indicate that 2π is not included. This allows us to include *any* possible orientation, but there is a problem. The orientation $2\pi - 0.00001$ is very close to 0, but in the interval $[0, 2\pi)$ they appear at opposite ends. The robot would be facing nearly the same direction whether $\theta = 2\pi - 0.00001$ or $\theta = 0$. Furthermore, if we increase θ a small amount from $2\pi = 0.00001$, then the orientation will arrive at $\theta = 0$. It seems that the ends of the interval are connected. (Think of a counter wrapping around to 0 after it reaches its maximum value.) To reflect this strange behavior, we say that θ could take any value in $[0, 2\pi]$, with the understanding that 0 and 2π are *the same* value. This can be formally defined with an equivalence relation. It is helpful to give this interval special notation, to remember that its endpoints are identical. Thus, let $S^1 = [0, 2\pi]$ with the understanding that 0 and 2π are identical. This is like a continuous version of modular arithmetic.

If there are no limits on the position coordinates, x and y, then the P-space is

$$\mathbf{P} = \mathbf{X} = \mathbb{R}^2 \times S^1, \tag{3.2}$$

in which $\times$ represents the Cartesian product from set theory. Each P-state is

[1] If knowledge of these were perfect, however, it might be able to use the observation to further calibrate the sensor (assuming the sensor was not already perfectly calibrated).

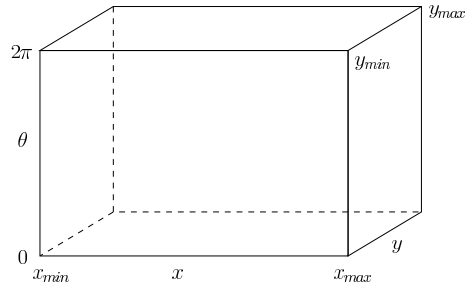


Figure 3.12: With upper and lower limits on x and y, The P-space for a planar mobile robot looks like a solid rectangular box. Note that the top and bottom of the box are identical due to the wraparound of θ at 0 and 2π.

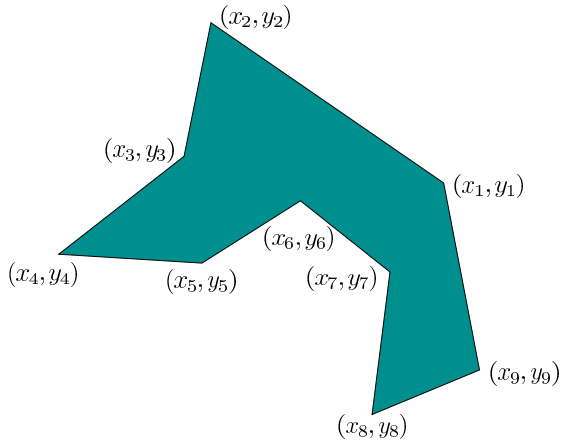


Figure 3.13: A polygonal obstacle is defined by listing its vertices in circular order.

written as $\mathbf{p} = (x, y, \theta)$. Suppose that x and y are limited so that

$$\begin{aligned} x_{min} \leq x \leq x_{max} \\ y_{min} \leq y \leq y_{max}. \end{aligned} \tag{3.3}$$

In this case, $\mathbf{P}$ can be visualized as a solid rectangular box, as shown in Figure 3.12. Every P-state is a point in the box. As the P-state changes, the point travels through the box. However, remember the strange behavior of θ and S^1. The top and bottom of the box are "the same". If a point travels to the top of the box, then it immediately reappears at the bottom of the box with the same (x, y) position. This is much like flying the ship off of the edge of the screen in the classic Asteroids arcade game and having it reappear on the other side. If there are no limits on x and y, then $\mathbf{P}$ can be visualized as an infinite slab of thickness 2π. The point may travel anywhere in the horizontal direction, but the top and bottom sides of the slab are the same. The structure of this space is well studied in mathematics and many engineering disciplines. It is referred to as $SE(2)$, the *special Euclidean group* of dimension two. For more details in the context of robotics, see [11].

The P-space $\mathbf{P} = \mathbf{X} = \mathbb{R}^2 \times S^1$ contains enough information to model some sensors from Section 3.1.2, which measure the robot configuration directly. For example, a compass measures θ, which depends only on the configuration (x, y, θ). This is already useful, but we can define many other sensor models if we allow the observations to depend on more information than just the robot configuration. Recall the sensors of Section 3.1.1, which measured distances to obstacles. This motivates the inclusion of obstacle models in the P-state.

Suppose that the robot is given a description of an obstacle O in its environment. To keep the description brief and precise, it is encoded as a polygon by

listing its n vertices in (circular) order:

$$((x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_n, y_n)). \tag{3.4}$$

Figure 3.13 shows an example.

We want to remove from $\mathbf{P}$ all states in which the robot platform overlaps with the obstacle O because these are impossible. We interpret O as a subset of $\mathbb{R}^2$. To keep the discussion simple for now, suppose that the entire robot platform is merely a point, located precisely at (x, y). In this case, any potential P-state $\mathbf{p} = (x, y, \theta)$ for which $(x, y) \in O$ is prohibited. If we remove all positions in O, then the remainder is the set of all allowable positions. This is called the *free space* F. Using set notation:

$$F = \mathbb{R}^2 \setminus O, \tag{3.5}$$

in which $A \setminus B$ denotes the resulting set from removing all elements of B that happen to lie in A. A subtle but important issue remains in defining F: What happens at the boundary between O and F? There are two situations:

1. The robot cannot touch obstacles. In this case, F should not include the boundary.

2. The robot may touch obstacles. In this case, F includes its entire boundary.

These cases can be described formally using open and closed sets, which are a generalization of open intervals, such as $(0, 1)$, and closed intervals $[0, 1]$; see [8, 11] for more details.

Using F, the P-space is

$$\mathbf{P} = F \times S^1 \tag{3.6}$$

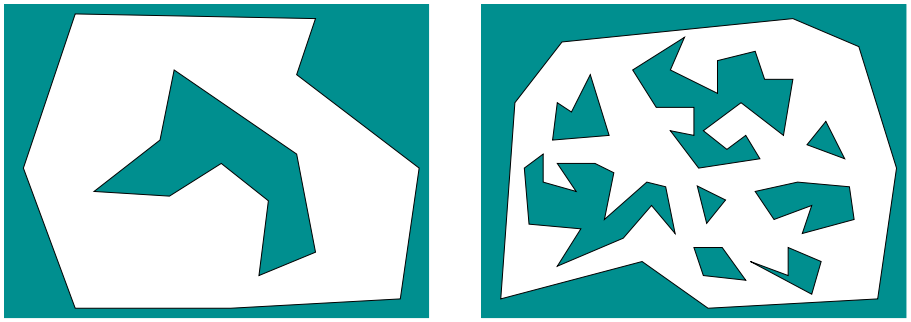
in which each $\mathbf{p}$ in $\mathbf{P}$ is still represented as

$$\mathbf{p} = (x, y, \theta). \tag{3.7}$$

There is no limit on orientation; therefore, S^1 appears above to represent the possible values for θ.

It is, of course, unrealistic to expect the environment to extend infinitely without obstacles. To trap the robot in a bounded region, we simply enclose it in another polygon. The obstacle is the outside part of the polygon and the free space is the interior (Figure 3.14(a)). To handle many obstacles, each can be represented as a polygon, leading to a very complicated environment model (Figure 3.14(b)). Other representations could be used as well, such a disc-shaped obstacles.

What if the robot platform is not a point? This leads to a well-studied problem in the area of robot *motion planning*. If the robot is a disc of radius r with its special point (x, y) at its center, then (x, y) must be kept at least distance r from the nearest obstacle (Figure 3.15). If the robot and obstacle are convex polygons, then the positions (x, y) that cause collision can be efficiently computed as a Minkowski sum [4]. However, if the robot is allowed to rotate, then the ability



(a) (b)

Figure 3.14: (a) An outer polygon forms an obstacle that makes the free space bounded. (b) To make complicated free spaces, an unlimited number of polygons may be added to the model.

to place the robot at position (x, y) without collision depends on the particular orientation θ. Let the set of all configurations (x, y, θ) that cause collision be denoted by $\mathbf{X}_{obs}$, which is a subset of the configuration space $\mathbf{X}$. The available collision free configurations are represented by the complement $\mathbf{X}_{free} = \mathbf{X} \setminus \mathbf{X}_{obs}$. Even though a polygon is obtained for each θ, the boundary of $\mathbf{X}_{free}$ is a nonlinear function of θ. Although computational methods exist, in most cases it is difficult to implement them correctly and to extremely expensive to compute an explicit representation of the set of all configurations that avoid collisions. See [1, 3, 10, 11] for more details, including extensions to 3D robots and multibody robots.

In this book, it is sufficient to remember that $\mathbf{X}_{free}$ represents the set of configurations that avoid collision with obstacles. If the robot is a point, then $\mathbf{X}_{free} = F \times S^1$, which is the expression in the right of (3.6). In the case of a disc robot, F is stripped around its boundary by thickness r to keep the robot center away from the obstacles. For a general robot shape, we use $\mathbf{X}_{free}$ to obtain the P-space:

$$\mathbf{P} = \mathbf{X}_{free}. \tag{3.8}$$

What happens when the obstacles are not completely known? In this case, we need to expand $\mathbf{P}$ to account for whatever possibilities exist. The set of all possible obstacles could be enormous, but we need only to define it, rather than explicitly represent it in a computer or enumerate all possibilities. Just as the set $\mathbb{R}$ is uncountably infinite and contains extremely complicated structures, the set $\mathbf{P}$ will be described and manipulated while avoiding its inherent complexity wherever possible.

To start with, suppose a point robot is given one of three scenarios for the free space (Figure 3.16). Perhaps it is given the floor plans to three different buildings and then is placed into one of them without being told which. In this case, the

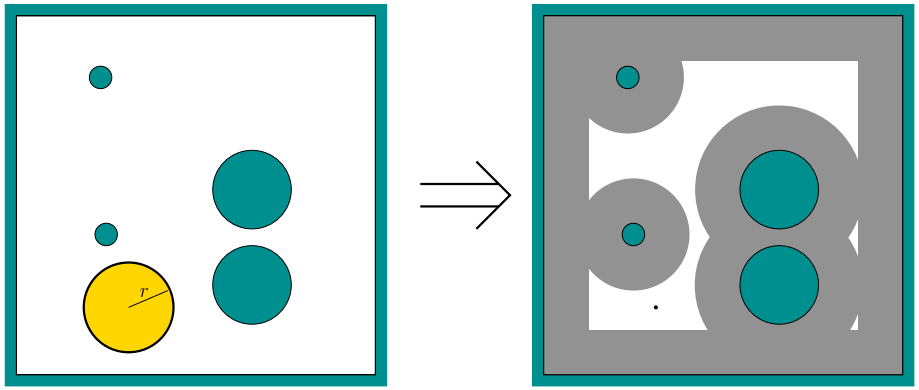


Figure 3.15: If the robot is not a point, then the free space depends in a complicated way on the robot configuration. In the case of a disc-shaped robot of radius r, it must keep its center at least distance r from the obstacles.

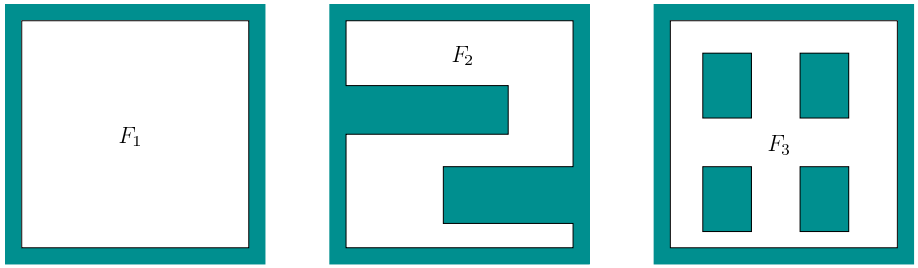


Figure 3.16: The free space could be one of three possibilities, resulting in $\mathcal{F} = \{F_1, F_2, F_3\}$.

P-space should account for the three cases: F_1, F_2, or F_3. One way to write the P-space is

$$\mathbf{P} = (F_1 \times S^1) \sqcup (F_2 \times S^1) \sqcup (F_3 \times S^1), \tag{3.9}$$

in which $\sqcup$ means disjoint union. In other words, it is possible for the same (x, y, θ) to appear in two or more of the three terms above. Therefore, we must force each one to be distinct (this is like variable name collision in programming). We could tag them as $(x, y, \theta)_1$, $(x, y, \theta)_2$, and $(x, y, \theta)_3$ to keep them distinct.

A simpler, equivalent expression of $\mathbf{P}$ is obtained by first defining

$$\mathcal{F} = \{F_1, F_2, F_3\}, \tag{3.10}$$

which is a set of sets. We then write

$$\mathbf{P} \subset \mathbb{R}^2 \times S^1 \times \mathcal{F}. \tag{3.11}$$

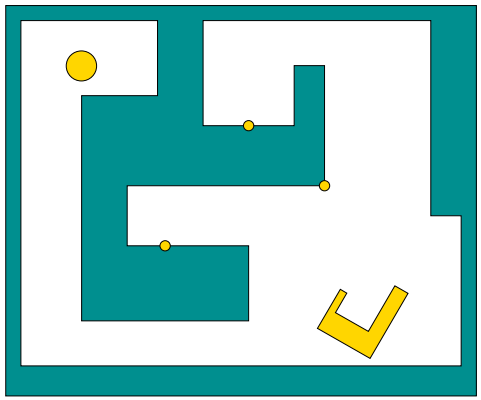


Figure 3.17: The environment may contain various kinds of *bodies*, such as robots, landmarks, objects, pebbles, targets, obstacles, evaders, treasures, and towers.

In this case, each $\mathbf{p} \in \mathbf{P}$ is written as

$$\mathbf{p} = (x, y, \theta, F), \tag{3.12}$$

with the requirement that $(x, y) \in F$ and $F \in \mathcal{F}$. This is exactly the structure pointed out at the beginning of this section with z, Z, and $\mathcal{Z}$.

Increasing the number of possible free spaces does not cause any conceptual difficulties. For example, perhaps $\mathcal{F}$ represents the set of *all* possible F that be constructed from polygons. What becomes important at this stage is specifying the constraints on $\mathcal{F}$. For example, F might exclude free spaces that extend infinitely. It might also exclude nonconvex polygons.

One final complication is that the robot might not be a point. In this case, a different $\mathbf{X}_{free}$ results for each possible F. The P-space for a general configuration space appears as

$$\mathbf{P} \subset \mathbf{X} \times \mathscr{X}_{free}, \tag{3.13}$$

in which $\mathbf{X}$ is the configuration space and $\mathscr{X}_{free}$ is the set of all possible $\mathbf{X}_{free}$. Every $(\mathbf{x}, \mathbf{X}_{free}) \in \mathbf{P}$, must satisfy $\mathbf{x} \in \mathbf{X}_{free}$ and $\mathbf{X}_{free} \in \mathscr{X}_{free}$.

In some of the examples above, there are multiple, distinct obstacles. We could imagine having an *augmented* map in which a label is associated with each obstacle. For example, if there are n discs, then they may be numbered from 1 to n. Knowing *which* obstacle it touches, for example, provides useful information to a lost robot. This model becomes a special case of the models considered next.

3.2.2 A bunch of bodies

Many tasks in robotics require more modeling beyond obstacles and free space. The robot may interact with many other entities, including people, landmarks, and other robots. Each such entity will be called a *body*, which could have one of

a number of possible interpretations in practice. A body B *occupies* a subset of $\mathbb{R}^2$ (or $\mathbb{R}^3$ for 3D models). Some bodies can be transformed using their own configuration parameters. For example, a body could be a point that is transformed by (x, y) parameters or a rectangle that is transformed by (x, y, θ) parameters. We can write $B(x, y, \theta) \subset \mathbb{R}^2$ to indicate the set of points occupied by B when at configuration (q_x, q_y, q_θ). In general, transformable bodies could be as complex as any robot; however, most of the time we will keep them simple to avoid the complexities of $\mathbf{X}_{free}$. Unless otherwise specified, assume transformable bodies are points or small discs to avoid problems with rotations. Otherwise, static bodies, such as obstacles, could be quite complicated.

In robotics and related subjects, many different names are given to bodies to indicate their interpretation for solving a particular task. Here are some common examples:

- **Robot:** A body that carries sensors, performs computations, and executes motion commands.
- **Landmark:** Usually a small body that has a known location and is easily detectable and distinguishable from others.
- **Feature:** A part of the environment that is easy to identify using a sensor. An alternative name is *observable*.
- **Object:** A body that can be detected and manipulated by a robot. It can be *carried* by a robot or *dropped* at a location.
- **Pebble:** A small object that is used as a marker to detect when a place has been revisited.
- **Target:** A person, a robot, or any other moving body that we would like to monitor using a sensor.
- **Obstacle:** A fixed or moving body that obstructs the motions of others.
- **Evader:** An unpredictable moving body that attempts to elude detection.
- **Treasure:** Usually a static body that has an unknown location but is easy to recognize by a sensor directly over it.
- **Beacon:** A body that transmits a signal, such as a cell-phone tower or a lighthouse. An alternative name is *tower*.

For discussing how bodies affect the design of sensors and filters it will be more important to understand the mathematical *characteristics* of a body, rather than its common name. Here are three crucial characteristics to establish for each body:

1. What are its *motion capabilities*?
2. Can it be *distinguished* from other bodies?

3. How does it *interact* with other bodies?

First consider motion capabilities. At one extreme, a body could be *static*, which means that it never moves. This is how we have modeled obstacles up until now. Even in this case, the its configuration could be unknown and therefore included in the P-state. If the body moves, then it may have *predictable* or *unpredictable motion*. Furthermore, the body may be able to move by itself, as in a person, or it may move only when manipulated by other bodies, such as a robot pushing a box.

Next we handle distinguishability. Consider a collection of bodies $B_1, \ldots, B_n$ that are distinguishable simply by the fact that each is uniquely defined. Define any equivalence relation $\sim$:

$$B_i \sim B_j \text{ if and only if they cannot be distinguished from each other.}$$

Another way to achieve this is by defining a set of *labels* and assigning a not-necessarily-unique label to each body. For example, the bodies may be people, and we may label them as male and female. More complicated distinguishability models are possible, but are not considered here.[2]

Finally, think about how bodies might interact or interfere with each other. Three interaction types are generally possible between a pair B_1, B_2, of bodies:

- **Sensor obstruction:** Suppose a sensor would like to observe information about body B_1. Does body B_2 interfere with the observation? For example, a truck could block the view of a camera, but a sheet of glass might not.

- **Motion obstruction:** Does body B_2 obstruct the possible motions of body B_1? If so, then B_2 becomes an obstacle that must be avoided.

- **Manipulation:** In this case, body B_1 could cause body B_2 to move. For example, if B_2 is an obstacle, then B_1 might push it out of the way.

In the rest of this chapter and throughout the book, many different kinds of bodies will appear and it is crucial to pay attention to their characteristics: 1) motion capabilities, 2) distinguishability, and 3) interaction. These are more important than their names.

3.3 Perfect Virtual Sensors

We now define mathematical models of instantaneous sensors: They use the current P-state to immediately produce an observation. Let $\mathbf{P}$ be any P-space. Let

[2]For example, indistinguishability does not even have to be an equivalence relation: Perhaps B_i and B_j are pairwise indistinguishable, B_j and B_k are pairwise indistinguishable, but B_i and B_k could be distinguishable.

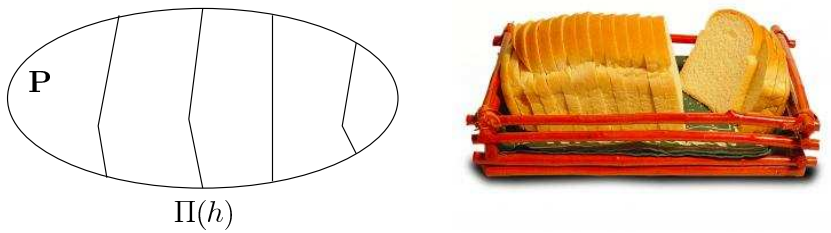


Figure 3.18: (a) A sensor should be viewed as a gadget that partitions the P-space $\mathbf{P}$. (b) A sensor definition is much like running $\mathbf{P}$ through a bread slicer. Each "slice" is a preimage.

$\mathbf{Y}$ denote the *observation set*, which is the set of all possible sensor outputs. A *virtual sensor* is defined by a function

$$h : \mathbf{P} \to \mathbf{Y}, \tag{3.14}$$

called the *sensor mapping*, which is very much like the transfer function described in Section 3.1.4. The interpretation is that when the P-state is $\mathbf{p}$, the sensor instantaneously observes $\mathbf{y} = h(\mathbf{p})$, which is a member of $\mathbf{Y}$. Equation 3.14 is the most important definition in this chapter.

What could an observation $\mathbf{y}$ tell the robot about the surrounding world? To understand this, we should think about all P-states could have produced $\mathbf{y}$. For a given sensor mapping h this is defined as

$$h^{-1}(\mathbf{y}) = \{\mathbf{p} \in \mathbf{P} \mid \mathbf{y} = h(\mathbf{p})\}, \tag{3.15}$$

and is called the *preimage* of $\mathbf{y}$. In words, $h^{-1}(\mathbf{y})$ is the set of all P-states that produce the same observation $\mathbf{y}$. If h were invertible, then h^{-1} would represent the inverse; however, because virtually all sensor models are many-to-one mappings, $h^{-1}(\mathbf{y})$ is a subset of $\mathbf{P}$, which yields all $\mathbf{p}$ that map to $\mathbf{y}$.

Consider the collection of subsets of $\mathbf{P}$ obtained by forming $h^{-1}(\mathbf{y})$ for every $\mathbf{y} \in \mathbf{Y}$. These sets are disjoint because a state $\mathbf{p}$ cannot produce multiple observations. Since h is a function on all of $\mathbf{P}$, the collection of subsets forms a partition of $\mathbf{P}$. For a given sensor mapping h, the corresponding partition of $\mathbf{P}$ is denoted as $\Pi(h)$.

The connection between h and $\Pi(h)$ is fundamental to sensing; see Figure 3.18. As soon as $\mathbf{P}$, $\mathbf{Y}$, and a sensor mapping h are defined, you should immediately think about how $\mathbf{P}$ is partitioned. The sets in $\Pi(h)$ can be viewed as equivalence classes. For any P-states $\mathbf{p}_1$ and $\mathbf{p}_2$, equivalence implies that $h(\mathbf{p}_1) = h(\mathbf{p}_2)$. These states are indistinguishable when using the sensor. In an intuitive way, $\Pi(h)$ gives the sensor's sensitivity to P-states, or the "resolution" at which the P-state can

be observed. The equivalence classes are the most basic source of uncertainty associated with a sensor.

In the remainder of this chapter, many virtual sensor models will be defined in terms of (3.14). Think about the partition of the P-space induced by each one. Each virtual sensor model can be physically implemented in several alternative ways, using the real sensors of Section 3.1. If (3.14) seems too idealistic, considering that sensors may be unpredictable and have delayed measurements, do not worry. Sensor disturbances and other complications are handled in Sections 3.5 and 3.6. The value of (3.14) is similar to the value of the perfect kinematic models of Chapter 2. The provide the nominal behavior upon which the robotic system is designed. Following that, additional complicatations are taken into account.

3.3.1 The Opposite Extremes

To illustrate the concept of virtual sensors, we start with some of the simplest extremes. The following is the most useless virtual sensor:

Sensor 1 (Dummy). *Take any P-space* $\mathbf{P}$ *and let* $\mathbf{Y} = \{0\}$. *The sensor mapping is*

$$h(\mathbf{p}) = 0. \tag{3.16}$$

The same observation $\mathbf{y} = 0$ *is produced for all* $\mathbf{p}$ *in* $\mathbf{P}$.

The dummy sensor never changes its output, thus providing no information about the external world. The particular observation $\mathbf{y} = 0$ is not important. An equivalent model would be obtained using any singleton set for the definition of $\mathbf{Y}$. We could also define an equivalent "broken" sensor. For example, $\mathbf{Y} = [0, 1]$, but the observation is always "stuck" on 0.

Each virtual sensor model will be referred to with a **Sensor** heading, as shown for the dummy sensor above. Keep in mind that each is a *mathematical model* of a physical sensor or family of sensors. Whenever it is clear, we will use the name *sensor* instead of *virtual sensor*, to save space.

What exists at the other extreme from the dummy sensor? Recall the absurd hypersensor from Section 1.4, which is able to recover "everything" about the physical world from a single observation. Using the concepts in this chapter, it is simple to define:

Sensor 2 (Hypersensor). *Let* $\mathbf{Y} = \mathbf{P}$. *The sensor mapping is*

$$\mathbf{Y} = h(\mathbf{p}) = \mathbf{p}. \tag{3.17}$$

In other words, $\mathbf{y} = \mathbf{p}$ *for all* $\mathbf{p}$ *in* $\mathbf{P}$. ■

Now consider the dummy sensor and hypersensor as bread slicers that partition the P-space. For the dummy sensor, the preimage is $h^{-1}(\mathbf{y}) = \mathbf{P}$, regardless of the observation $\mathbf{y}$. Thus, the partition $\Pi(h)$ contains only one "slice", which is

$\mathbf{P}$ itself. The sensor appears to do nothing to divide up $\mathbf{P}$, which should not be surprising. On the other hand, the hypersensor slices the P-space with the finest possible granularity. For each $\mathbf{P}$, the observation is $\mathbf{y} = \mathbf{P}$, and the preimage is $h^{-1}(\mathbf{y}) = \{\mathbf{p}\}$. Therefore, the partition $\Pi(h)$ is the finest possible. Every $\mathbf{p}$ in $\mathbf{P}$ has a corresponding singleton $\{\mathbf{p}\}$ in $\Pi(h)$. Although both of these virtual sensors appear absurd, they are useful by providing the extreme limits, much in the same way that we find $-\infty$ and ∞ useful when performing calculations with real numbers. All other sensors will fall somewhere in between. In some cases, if a sensor experiences too much disturbance, it is helpful to know that it is equivalent to a dummy sensor. In other cases, perhaps a combination of sensors becomes so powerful that it is equivalent to the hypersensor.

These extreme sensor models can be combined in a simple and useful way. If the P-space is formed by a Cartesian product, then the virtual sensor might appear as a dummy with respect to one component and as a hypersensor to another. This produces a *projection sensor*:

Sensor 3 (Projection). *Let* $\mathbf{P} = \mathbf{P}_1 \times \mathbf{P}_2$ *with each P-state represented as* $\mathbf{p} = (\mathbf{p}_1, \mathbf{p}_2)$. *The sensor mapping is*

$$\mathbf{y} = h(\mathbf{p}_1, \mathbf{p}_2) = \mathbf{p}_1. \tag{3.18}$$

Many other variations are possible, depending on how many components are in the Cartesian product and which of those components are observed. ■

The sensor mapping (3.18) reveals everything about first component $\mathbf{p}_1$, but nothing about $\mathbf{p}_2$. The preimage of $\mathbf{y} = \mathbf{p}_1$ is the set of all $(\mathbf{p}_1, \mathbf{p}_2)$ in which $\mathbf{p}_2$ could be *any* in $\mathbf{P}_2$. For example, if $\mathbf{P} = \mathbb{R} \times \mathbb{R} = \mathbb{R}^2$, then each P-state can be imagined as robot position (x, y). The projection sensor observes only the x coordinate: $x = h(x, y)$. The preimage is a vertical line, positioned at x. In terms of the partition $\Pi(h)$, the sensor slices $\mathbf{P}$ into the set of all vertical lines. In terms of the x coordinate, Sensor 3 appears to be a hypersensor. In terms of the y coordinate, it is a dummy sensor.

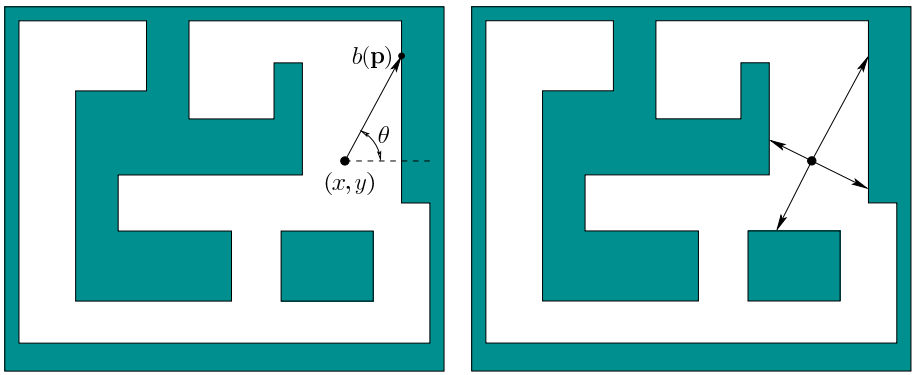
Another simple example of a projection sensor is a *compass*:

Sensor 4 (Compass). *Let* $\mathbf{P} = \mathbb{R}^2 \times S^1$ *with each P-state represented as* $\mathbf{p} = (x, y, \theta)$. *The sensor mapping is*

$$\mathbf{y} = h(x, y, \theta) = \theta. \tag{3.19}$$

■

The compass observes only the direction ($\theta \in S^1$) that the robot is facing. Each preimage is the set of all (x, y, θ) for which x and y could be any real value. In other words, the compass partitions $\mathbf{P}$ into planar slices. Each slice corresponds to the unobserved position (x, y).



(a) Directional depth (b) k-directional depth

(c) Omnidirectional depth (d) Boundary distance

Figure 3.19: Several alternative scenarios for depth sensors.

3.3.2 Depth Sensors

We now introduce virtual sensor models that are appropriate for most of the physical sensors in Section 3.1.1. Their primary use was to measure distances to obstacles. Using the P-space models from Section 3.2.1, *depth sensors* base the observation on distance from the sensor to the obstacle boundary. The particular P-space used here is $\mathbf{P} \subset \mathbb{R}^2 \times S^1 \times \mathcal{F}$, which first appeared in (3.11). Each P-state is represented as $\mathbf{p} = (x, y, \theta, F)$, in which (x, y) is the robot position, θ is its orientation, and F is the free space. Assume that the robot is a point.

Figure 3.19 shows several depth-measuring scenarios. Figure 3.19(a) depicts a robot taking a single measurement in the direction it is facing. The physical implementation could perhaps use a laser with phase shifting mounted on the front of the robot, but this is not important here. We care only about the information

provided and what that information depends on. The virtual sensor model for this scenario is:

Sensor 5 (Directional Depth Sensor)**.** *Let $b(\mathbf{p})$ denote the point on the boundary of F that is struck by a ray emanating from (x, y) and extended in the direction of θ (see Figure 3.19(a)). The sensor mapping is*

$$h_d(x, y, \theta, F) = \|(x, y) - b(\mathbf{p})\|, \tag{3.20}$$

in which $\|\cdot\|$ denotes the length of the vector. ■

Note that all four components of $\mathbf{p}$ were needed: x, y, θ, and F. Without F it would be impossible to predict the location of $b(\mathbf{p})$. However, be careful! This does not imply that the robot knows F. It is only stating that to predict what the sensor will observe, F is necessary, along with x, y, and θ.

Now suppose the sensor is facing a different direction with respect to the robot. Let ϕ be the angle between the sensor direction and the robot direction. In (3.20), $\phi = 0$. To account for a sensor that faces another direction, $b(\mathbf{p})$ is extended to $b(\mathbf{p}, \phi)$, to enable dependency on ϕ. This yields a ϕ-offset directional depth sensor:

Sensor 6 (ϕ-Offset Directional Depth Sensor)**.** *In terms of the offset angle ϕ, the depth measurement is given by*

$$h_\phi(x, y, \theta, F) = \|(x, y) - b(\mathbf{p}, \phi)\|. \tag{3.21}$$

■

A k-directional depth sensor is constructed by applying (3.21) for k different values of ϕ. For the example in Figure 3.19(b), the four values of ϕ are 0, $\pi/2$, π, and $3\pi/2$.

Sensor 7 (k-Directional Depth Sensor)**.** *Let $\phi_1, \ldots, \phi_k$ be a set of k offset angles for each directional depth sensor. The sensor mapping yields a vector of observations*

$$\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_k), \tag{3.22}$$

in which each $\mathbf{y}_i$ is given by h_{ϕ_1} in (3.21). ■

This model would be appropriate for an array of sonars or even the SICK laser scanner from Figure 3.5. In that case, $k = 1080$. Rather than reporting a 1080-dimensional vector $\mathbf{y}$ of observations, it is often convenient to model such sensors as producing a *continuum* of depth observations. See Figure 3.20. An observed distance is obtained for every ϕ from 0 to 2π. This can be described as a function $\mathbf{p} : S^1 \rightarrow [0, \infty]$. For each $\phi \in S^1$, a distance between 0 and ∞ is observed. If there are obstacles in all directions, then infinite distance will never be observed. The resulting virtual sensor model is:

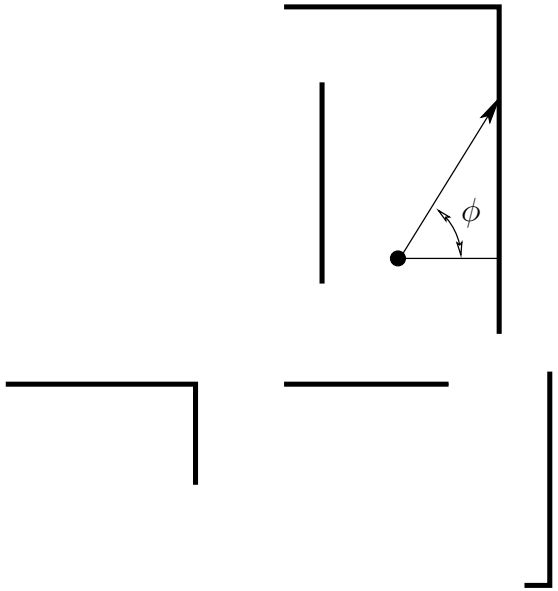


Figure 3.20: For Sensor 8, a function $y : S^1 \rightarrow [0, \infty)$ is obtained in which each $y(\phi)$ is the depth in the direction $\theta + \phi$. The figure shows how the depth data appears for the environment in Figure 3.19(a) and $\theta = 0$.

Sensor 8 (Omnidirectional Depth Sensor)**.** *This virtual sensor obtains a distance measurement in every direction $\phi \in S^1$ at the same time. Hence, the observation is a function of ϕ, defined as*

$$\mathbf{y}(\phi) = h_\phi(x, y, \theta, F). \tag{3.23}$$

The observation set $\mathbf{Y}$ is actually the set of all functions of the form $\mathbf{p} : S^1 \rightarrow [0, \infty]$ (that could be obtained given $\mathbf{P}$). ■

This covers the cases of directional depth measurement with increasing resolution with respect to directions. Several restricted models are also needed to handle many realistic settings. For example, as shown in Figure 3.21, the range of distances and angles might be limited. In general, the sensor might yield a special symbol to denote that the depth is out of range. For example, the sensor range may be limited to 3 meters. For an obstacle further than that, it reports "out of range".

What if we want to know the distance to the nearest wall, regardless of direction? See Figure 3.19(d). This could be obtained by taking the minimum of (3.23) over all ϕ. In practice this would be overkill, especially because some physical sensors can obtain this distance directly, without measuring directional distance. For

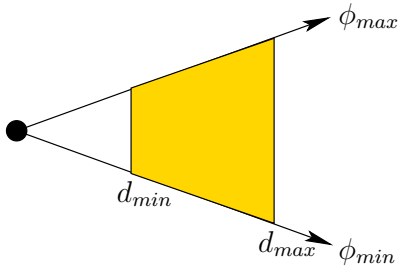


Figure 3.21: A depth sensor that has limited range and angle.

example, a physical sensor could work by transmitting an omnidirectional signal and measuring the phase shift or intensity of the reflection. Even though it might not measure this direction, it can be conveniently defined in terms of directional depth:

Sensor 9 (Boundary Distance Sensor)**.** *For measuring the distance to the nearest wall, the sensor mapping is:*

$$h_{bd}(\mathbf{p}) = \min_{\phi \in [0, 2\pi)} h_\phi(\mathbf{p}, \phi). \tag{3.24}$$

■

Although the function h_ϕ depends on the particular robot direction θ, the resulting minimum value over all ϕ does not. In other words, we can add an offset angle to ϕ and the same result would be obtained in (3.24). This is appropriate because the distance to the wall should not depend on the robot direction (assuming the robot is a point).

Recall from Section 3.1.1 that some sensors produce such poor distance information that they can only be reliably applied to determine whether the obstacle is "near" or "far". This motivates the next virtual sensor model:

Sensor 10 (Proximity Sensor)**.** *For a fixed distance ϵ, the proximity sensor indicates whether the distance is within ϵ. The sensor mapping is*

$$h_\epsilon(\mathbf{p}) = \begin{cases} 1 & \text{if } h_{bd}(\mathbf{p}) \leq \epsilon \\ 0 & \text{otherwise.} \end{cases}, \tag{3.25}$$

which uses the sensor mapping from (3.24). ■

The proximity sensor simply applies a threshold to the boundary distance sensor observation. In the limiting case of $\epsilon = 0$, the proximity sensor determines whether the robot is touching the boundary. This could be implemented with a simple limit switch. The model is:

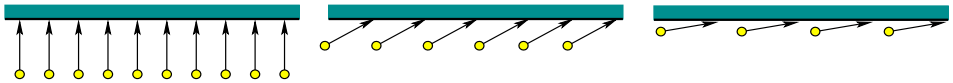


Figure 3.22: The preimage for a single-directional depth sensor is a two-dimensional subset of $F \times S^1$, assuming the free space F is given. Shown here are several robot configurations within the same preimage.

Sensor 11 (Boundary Sensor)**.** *The sensor mapping for a boundary (or contact) sensor is:*

$$h(x, y, \theta, F) = \begin{cases} 1 & \text{if } (x, y) \in \partial F \\ 0 & \text{otherwise.} \end{cases} , \tag{3.26}$$

in which ∂F means the boundary of F. ■

The sensor mapping (3.26) could have been defined by just setting $\epsilon = 0$ in Sensor 10.

All of the depth sensors can be viewed as bread slicers that partition the P-space into sets that produce the same depth observations. Think about the preimages associated with the various depth senors and the resulting partition $\Pi(h)$. First consider the special case of a given polygonal free space F, leading to $\mathbf{P} = F \times S^1$. For Sensor 5 (directional depth), each preimage $h^{-1}(\mathbf{y})$ is generally a two-dimensional subset of $\mathbf{p}$ that corresponds to all possible configurations from which the same directional distance could be obtained. Thus, $\Pi(h)$ is a collection of disjoint, two-dimensional subsets of $E \times S^1$. Equivalent P-states along a single wall are depicted in Figure 3.22.

Using Sensor 11, $\Pi(h)$, contains only two classes: All P-states in which the robot is in the interior of F, and all P-states in which it is on the boundary of F. Sensor 8 provides omnidirectional depth. This is quite powerful, which consequently leads to very small preimages. In most cases, these correspond to the finite set of symmetry classes of the environment, which are presented in Chapter 6 in the context of robot localization.

If the free space is not known, but is instead restricted to a set of possible F in $\mathcal{F}$, then the preimages enlarge accordingly. Each $h^{-1}(\mathbf{y})$ contains a set of possible x, y, θ, and F that could have produced the observation. In the case of a boundary sensor, $h^{-1}(1)$ represents "all environments and configurations in which the robot is touching an obstacle". For the omnidirectional sensor, $h^{-1}(\mathbf{y})$ indicates all ways that F could exist beyond the field of view of the sensor. These preimages are often simple to describe with words, but we usually do not want to expand them indivudually in the robot's brain.

3.3.3 Comparing Virtual Sensors

After seeing so many virtual sensor models, you might already have asked, what would it mean for one sensor to be more powerful than another? It turns out that there is a simple, clear way to determine this in terms of preimages.

In this section, assume that the P-space $\mathbf{P}$ is predetermined and fixed. Consider any two sensor mappings, $h_1 : \mathbf{P} \to \mathbf{Y}_1$ and $h_2 : \mathbf{P} \to \mathbf{Y}_2$. Each one slices the P-space in different ways according to its preimages. The partitions of the P-space are $\Pi(h_1)$ and $\Pi(h_2)$.

In general, a partition Π_1 is called a *refinement* of a partition Π_2 if every set in Π_1 is a subset of some set in Π_2. In terms of bread, it means that every slice obtained from h_1 is a piece of a slice formed by h_2. If this happens, then we declare that h_1 is *better than* (or *dominates*) h_2, which we write as $h_1 \succeq h_2$.

For some P-state $\mathbf{p}$, imagine receiving $\mathbf{y}_1 = h_1(\mathbf{p})$ and $\mathbf{y}_2 = h_2(\mathbf{p})$. If $h_1 \succeq h_2$, then their preimages are related as:

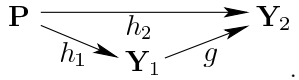
$$h_1^{-1}(\mathbf{y}_1) \subseteq h_2^{-1}(\mathbf{y}_2) \subseteq \mathbf{P} \tag{3.27}$$

This clearly means that h_1 provides at least as much information about $\mathbf{p}$ as h_2. Furthermore, using $\mathbf{y}_1$, we could infer what observation $\mathbf{y}_2$ would be produced by h_2. Why? Since $\Pi(h_1)$ is a refinement of $\Pi(h_2)$, every $\mathbf{p} \in h_1^{-1}(\mathbf{y}_1)$ must produce the same observation $\mathbf{y}_2 = h_2(\mathbf{p})$.

This means that h_1 is powerful enough to *simulate* h_2 with some appropriate post-processing. Let this post-processing be represented as a function $g : \mathbf{Y}_1 \to \mathbf{Y}_2$. We want to apply g to the observation from h_1 in a way that simulates h_2:

$$h_2(\mathbf{p}) = g(h_1(\mathbf{p})). \tag{3.28}$$

Here is a diagram of the functions:



As a simple example, suppose h_1 is the boundary distance sensor mapping (Sensor 9) and h_2 is the boundary sensor mapping (Sensor 11). Applying h_1 yields the distance $\mathbf{y}$ to the boundary. The function g is then applied to transform $\mathbf{y}$. We define $g(\mathbf{y}) = 1$ if $\mathbf{y} = 0$; otherwise, $g(\mathbf{y}) = 0$.

The existence of g implies that the observations of h_1 can be used to "simulate" h_2, without needing additional information about the P-state. One important point, however, is that it might be computationally impractical or infeasible for the robot to compute g. The computability of g and algorithmic complexity of computing g for various pairs of sensor models leads to interesting questions that have not been solved to date.

Using the dominance relation $\succeq$, we can easily compare any of the sensors in Section 3.3.2. Note that $\succeq$ is only a *partial* ordering: Most sensor pairs are incomparable. Figure 3.23 shows how some sensors are related, assuming typical parameter choices for each model. The most powerful sensor of Section 3.3.2 is the omnidirectional depth sensor because it yields the finest partition of $\mathbf{P}$. We can use it to simulate all other depth sensors. Note that these relationships hold regardless of the particular collection $\mathcal{F}$ of possible environments. It does not

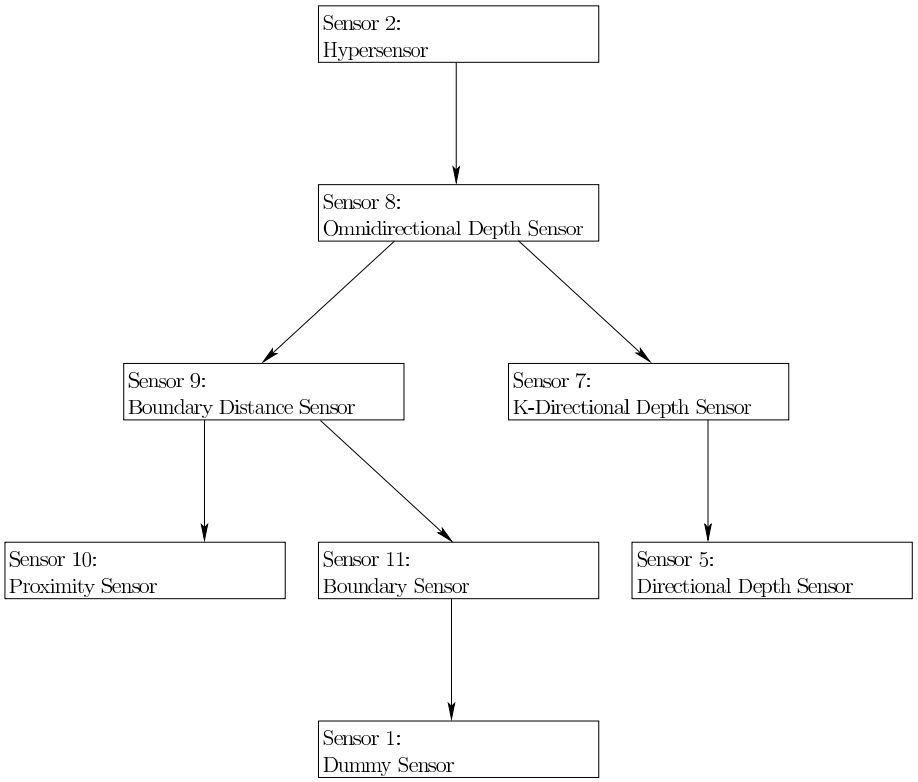


Figure 3.23: Several models from Section 3.3 are related using the idea of dominance, based on refinements of the partitions they induce over $\mathbf{P}$. Models higher in the tree induce finer partitions. A lower sensor model can be "simulated" by any model along the path from the root of the tree to itself.

matter whether the environment is given or is open to some infinite collection of possibilities.

What happens as we include many more sensors, and continue to extend the diagram in Figure 3.23? It turns out that *all* possible sensors of the form $h : \mathbf{P} \rightarrow \mathbf{Y}$ over a fixed P-space $\mathbf{P}$ can be related in a clear way, and the tree extends into a lattice.

To set up this structure, we first need to define what it means for two sensors to be *equivalent*. In this case, they can simulate each other. We define an equivalence relation:

$$h_1 \sim h_2 \text{ if and only if } \Pi(h_1) = \Pi(h_2). \tag{3.29}$$

In other words, two sensors are equivalent if they partition $\mathbf{P}$ the same way. This definition assumes the P-space $\mathbf{P}$ is fixed, but the relation $\sim$ can be applied to sensor mappings with different observation spaces. As an example of equivalent

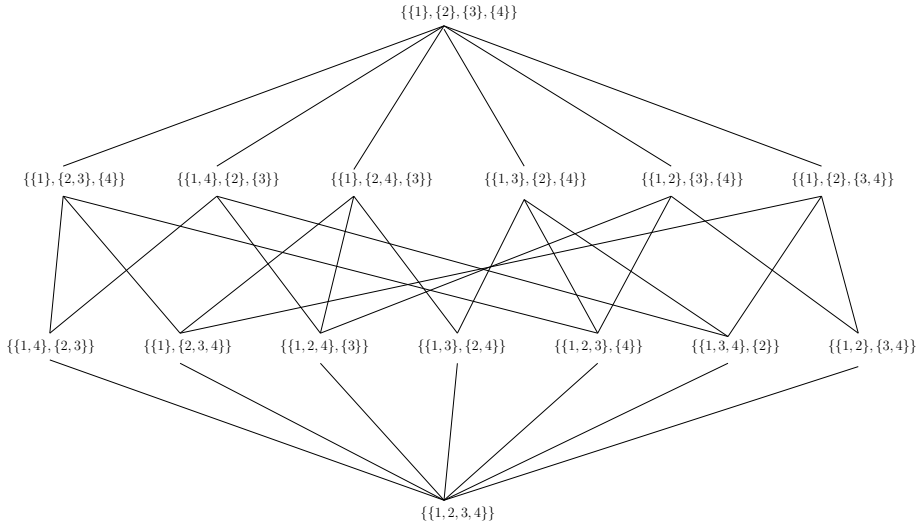


Figure 3.24: The partition lattice for a four-element set. The best and worst sensors are at the top and bottom, respectively.

sensors, the following generates a huge family of sensors that are all equivalent to the hypersensor (Sensor 2):

Sensor 12 (One-To-One)**.** *The sensor mapping h is any one-to-one function from $\mathbf{P}$ to $\mathbf{Y}$.* ■

For every one-to-one sensor h, the partition $\Pi(h)$ contains only singleton sets. Hence, all one-to-one sensors are equivalent to the identity sensor.

If we no longer pay attention to the particular h and $\mathbf{Y}$, but only consider the resulting partition of the P-space, then we imagine that a sensor *is* a partition of X. Continuing in this way, the set of all possible sensors is the set of all partitions of $\mathbf{P}$.

The relationship between sensors in terms of dominance then leads to the well-known idea of a *partition lattice*, depicted in Figure 3.24 for the set $\mathbf{P} = \{1, 2, 3, 4\}$. Recall from abstract algebra that a *lattice* is a set together with a partial order relation $\succeq$ for which every pair of elements has a unique *least upper bound (lub)* and a *greatest lower bound (glb)*. The lub of Π_1 and Π_2 is the lowest element in the lattice that dominates both of them. For example, in Figure 3.24, the lub of $\{\{1\}, \{2, 3, 4\}\}$ and $\{\{1, 3\}, \{2, 4\}\}$ is $\{\{1\}, \{2, 4\}, \{3\}\}$. The glb is the highest element in the lattice that both Π_1 and Π_2 dominate. The glb of $\{\{1\}, \{2, 3, 4\}\}$ and $\{\{1, 3\}, \{2, 4\}\}$ is $\{1, 2, 3, 4\}$. Starting with any set, the set of all partitions forms a lattice. The relation $\succeq$ is defined using refinements of partitions: $\Pi_1 \succeq \Pi_2$ if and only if Π_1 is a refinement of Π_2. This is precisely our

way of defining sensor dominance. For any $\mathbf{P}$, we therefore have a resulting *sensor lattice* that characterizes all possible sensor mappings. Figure 3.23 revealed only a small portion of this lattice.

The glb and lub have interesting interpretations in the sensor lattice. Suppose that for two partitions, $\Pi(h_1)$ and $\Pi(h_2)$, neither is a refinement of the other. Let $\Pi(h_3)$ and $\Pi(h_4)$ be the glb and lub, respectively, of h_1 and h_2. The glb $\Pi(h_3)$ is the partition obtained by "overlaying" the partitions $\Pi(h_1)$ and $\Pi(h_2)$. Take any P-state $\mathbf{p}$. Let $\mathbf{y}_1$, $\mathbf{y}_2$, $\mathbf{y}_3$, and $\mathbf{y}_4$, be the observations obtained by applying h_1, h_2, h_3, and h_4, respectively. An element of $\Pi(h_3)$ is obtained by intersecting preimages,

$$h_1^{-1}(y_1) \cap h_2^{-1}(y_2). \tag{3.30}$$

There is a straightforward way to construct some representative sensor mapping h_3 using only h_1 and h_2. Let $\mathbf{Y}_3 = \mathbf{Y}_1 \times \mathbf{Y}_2$ and $h_3 : \mathbf{P} \to \mathbf{Y}_3$ be defined as

$$\mathbf{y}_3 = (\mathbf{y}_1, \mathbf{y}_2) = (h_1(\mathbf{p}), h_2(\mathbf{p})). \tag{3.31}$$

This means that both h_1 and h_2 are combined to produce a single sensor. The partition $\Pi(h_3)$ is just the common refinement.

The lub, $\Pi(h_4)$, is the opposite of $\Pi(h_3)$ in some sense. The partition $\Pi(h_4)$ is as coarse as it needs to be so that every element contains the complete preimages of h_1 and h_2. It is like finding large enough bread slices to contain the slices from both h_1 and h_2. Again starting from any $x \in X$, $\Pi(h_4)$ is the finest partition for which $h^{-1}(y_1) \cup h^{-1}(y_2) \subseteq h^{-1}(y_3)$.

Nice examples are provided by the family of linear sensors, which are widely used in control theory ([2]):

Sensor 13 (Linear Sensor)**.** *Let* $\mathbf{P} = \mathbf{Y} = \mathbb{R}^3$. *The sensor mapping is*

$$\mathbf{y} = C\mathbf{p}, \tag{3.32}$$

for some 3 *by* 3 *real-valued matrix* C. *An extension could easily be made to the case of* $\mathbf{Y} = \mathbb{R}^m$ *and* $\mathbf{P} = \mathbb{R}^n$. *In this case,* C *is an* $m \times n$ *matrix.*

If C has rank 3, then then the sensor mapping is one-to-one, which makes it a special case of Sensor 12. If C has rank 0, then the sensor is equivalent to Sensor 1 (dummy). If C has rank 2, then the preimages $h^{-1}(\mathbf{y})$ are lines in $\mathbb{R}^3$. The glb of any two sensors that have rank 2 results in a sensor with rank 3. Every pair of lines intersects to form a point, which represents the preimage of the glb sensor mapping. The lub of two sensors for which C has rank two is a sensor mapping for which C has rank one. Every preimage of the lub sensor mapping is a plane in $\mathbb{R}^3$ that is spanned by the intersecting lines from the preimages of the original two sensors.

3.4 More Sensor Families

This section introduces a variety of other virtual sensor models. The sensor models introduced so far in this chapter are sufficient to understand the later sections and

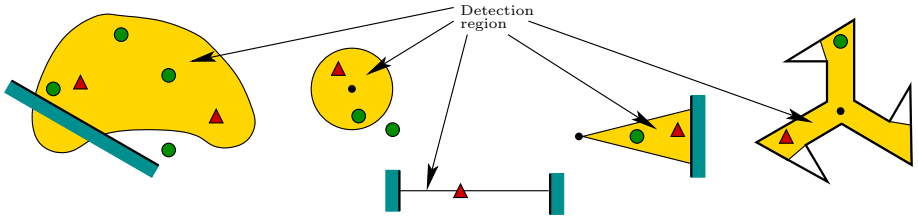


Figure 3.25: A detection region V is a subset of the free space in which moving bodies (shown as triangles and discs) are detected. Five examples are shown. The detection regions may appear in various shapes and may or may not be attached to a movable body.

chapters; however, a more complete collection of models that can be used to design mobile robot systems is provided here.

3.4.1 Detection Sensors

As the name suggests, this virtual sensor family models physical sensors that *detect* whether one of more bodies are within their sensing range. The sensors are not concerned with recovering precise locations. These could be implemented in a variety of ways, including the use of cameras, occupancy detectors, and pressure mats.

Three fundamental aspects are important to keep in mind with detection:

1. Can the sensor move? For example, it could be mounted on a robot or it could be fixed to a wall.

2. Are the bodies so large relative to the range of the sensor that the body models cannot be simplified to points?

3. Can the sensor provide additional information that helps to classify a body within its detection region?

If the answer is "no" to all three questions, then the simplest case is obtained:

> A static detection sensor that indicates whether at least one point body is within its range.

For this case, a predetermined subset V of the free space F is called the *detection region*. Suppose that F contains one ore more point bodies that can move around. Note that V can be any shape, as shown in Figure 3.25. If a body enters V, then it is detected. Here is the simple sensor model:

Sensor 14 (Static Binary Detector)**.** *A single point body moves in* F *and its position is denoted by* b. *If* F *is known, then the P-space is* $\mathbf{P} = F$ *and the sensor*

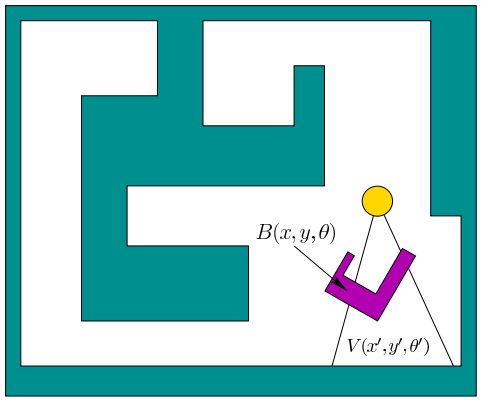


Figure 3.26: For larger bodies, we must declare either *some* or *all* of the body must be in view to be detected.

mapping is

$$h(b) = \begin{cases} 1 & \text{if } b \in V \\ 0 & \text{otherwise.} \end{cases} \tag{3.33}$$

If F is not known, then $h(b)$ is replaced by $h(b, F)$, and $\mathbf{P} = (b, \mathcal{F})$. ■

The sensor simply indicates whether the body is in the detection region V. This could be implemented, for example, with a cheap occupancy sensor that is mounted on the wall.

We will now extend (3.33) in three independent ways based on the questions above. Each way will be handled separately, but all three extensions can be combined. First, suppose that the sensor can move, as in a camera that is mounted on a mobile robot. In this case, the detection region depends on the robot configuration:

Sensor 15 (Moving Binary Detector)**.** *Let (x, y, θ) denote the configuration of a point body that is carrying the sensor. Let $V(x, y, \theta) \subset F$ represent the configuration-dependent detection region. If F is known, then the P-space is $\mathbf{P} = F \times F \times S^1$ and the sensor mapping is*

$$h(b, x, y, \theta) = \begin{cases} 1 & \text{if } b \in V(x, y, \theta) \\ 0 & \text{otherwise.} \end{cases} \tag{3.34}$$

As usual, this can be easily adapted to the case in which F is not known. ■

Now return to the case in which the sensor is static. We instead allow the body to have its own shape. Imagine, for example, that the body is a robot that is independent of the sensor. In this case, it has its own configuration:

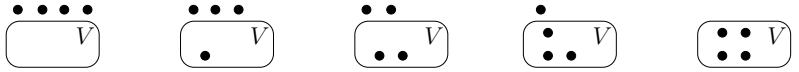


Figure 3.27: A fixed detection sensor among 4 moving points in $\mathbb{R}^2$ yields these 5 equivalences classes for the partition $\Pi(h)$ of X. In this model, the observation y is the number of points in V.

Sensor 16 (Detecting Larger Bodies)**.** *Let $B(x, y, \theta)$ represent the subset of F that is occupied by the body when it is at position (x, y) and orientation θ. The sensor mapping is:*

$$h(x, y, \theta) = \begin{cases} 1 & \text{if } B(x, y, \theta) \cap V \neq \emptyset \\ 0 & \text{otherwise.} \end{cases} \tag{3.35}$$

■

The sensor detects the body if any part of it enters V. This is similar to the definition of configuration-space obstacle region, $\mathbf{X}_{obs}$, in motion planning [3, 10, 11]. See Figure 3.26. An alternative definition would require the body to be contained in the detection region: $B(x, y, \theta) \subseteq V$. If the sensor can additionally move, then V in (3.35) is replaced with $V(x', y', \theta')$ to account for its configuration (x', y', θ') as well.

If there are multiple bodies, then the virtual sensor model must specify how this is handled. A simple case is:

Sensor 17 (At-Least-One-Body Detector)**.** *Let $B = \{b_1, \ldots, b_n\}$ denote a set of n point bodies that move in F. This model detects whether there is at least one body in the detection region $V(x, y, \theta)$. The sensor mapping is*

$$h(x, y, \theta, b_1, \ldots, b_n) = \begin{cases} 1 & \text{if for any } i,\ b_i \in V(x, y, \theta) \\ 0 & \text{otherwise.} \end{cases} \tag{3.36}$$

■

Alternatively, the sensor could keep track of the number of bodies in its detection region:

Sensor 18 (Body Counter)**.** *The sensor mapping is*

$$h(x, y, \theta, b_1, \ldots, b_n) = |B \cap V(x, y, \theta)|, \tag{3.37}$$

in which $|\cdot|$ denotes the number of elements in a set. ■

As an illustrative special case, suppose that n point bodies move in $F = \mathbb{R}^2$. A static detection sensor is installed that counts how many points are within a fixed

detection region V. The P-space is $\mathbf{P} = \mathbb{R}^{2n}$ and the observation space is $\mathbf{Y} = \{0, 1, \ldots, n\}$. Now consider the preimages. The partition $\Pi(h)$ is formed by $n+1$ equivalence classes. Figure 3.27 shows how these subsets of X could be depicted for the case of $n = 4$. If the sensor were additionally able to distinguish between the points and determine which are in V, then there would be 2^n equivalence classes. Such a sensor would be strictly more powerful and the equivalence classes would be correspondingly smaller.

More generally, we can consider bodies that are partially distinguishable to the sensor. In a physical implementation, a camera could be used with computer vision techniques to classify and label bodies in the image. Let L be a set of class labels, attribute values, or feature values that can be assigned to bodies. Let ℓ be an assignment mapping:

$$\ell : \{1, \ldots, n\} \rightarrow L. \tag{3.38}$$

In other words, ℓ gives a label to each body. This enables detection sensors to be extended to observe the label of the detected body. A simple model that achieves this is:

Sensor 19 (Labeled-Body Detector)**.** *Suppose that we want to detect whether a body is in the detection region and has a particular label $\lambda \in L$. In this case, the sensor mapping is:*

$$h_\lambda(b_1, \ldots, b_n) = \begin{cases} 1 & \text{if for some } i,\ b_i \in V \text{ and } \ell(i) = \lambda \\ 0 & \text{otherwise.} \end{cases} \tag{3.39}$$

■

Numerous other variations are useful to construct:

1. A detection sensor could count bodies that share the same label.
2. Each body could be modeled as having its own configuration parameters, to allow translation and rotation.
3. The number of bodies may not be specified in advance
4. If the boundary of V has multiple connected components (separate blobs), then the sensor might indicate which component was crossed.
5. Multiple detection sensors could be in use, each of which classifies bodies differently.

3.4.2 Relational Sensors

We now take detection sensors as a starting point and allow them to provide a critical piece of information:

How is one body situated relative to another?

This leads to the family of *relational sensors* [7]. A detection sensor only tells us which bodies are in view, whereas a relational sensor additionally indicates how they are arranged. A relational sensor is not, however, powerful enough to give precise relative locations. These virtual sensors could be implemented with cameras or other imagining technologies. Even though some precise locations could be extracted, it might not be reliable enough and the robot might not even require it to solve its tasks. Therefore, relational sensors provide a powerful abstraction for describing a level of useful information that can be reliable extracted and used to solve tasks.

A *relation* R on a set X is function that assigns a *true* or *false* value for every pair (x_1, x_2), in which $x_1, x_2 \in X$. Formally, $R : X \times X \rightarrow \{true, false\}$. Let R be any relation on the set of all bodies. For a pair of bodies, B_1 and B_2, examples of possible definitions for $R(B_1, B_2)$ are:

- B_1 is in front of B_2
- B_1 is to the left of B_2
- B_1 is on top of B_2
- B_1 is closer than B_2
- B_1 is bigger than B_2.

This information actually depends on the P-state. We therefore write the relation as $R_\mathbf{p}$ and define it using an index set $\{1, \ldots, n\}$ to refer to particular bodies. Using this notation for the "in front of" example, $R_\mathbf{p}(i, j)$ means that body B_i is in front of B_j when viewed by the sensor from the P-state $\mathbf{p}$.

This leads to the simplest case:

Sensor 20 (Primitive Relational Sensor)**.** *This sensor indicates whether the relation $R_\mathbf{p}$ is satisfied for two bodies B_i and B_j that are in the detection region:*

$$h(\mathbf{p}) = \begin{cases} 1 & \text{if } R_\mathbf{p}(i, j), \text{ and } B_i \text{ and } B_j \text{ are detected} \\ 0 & \text{otherwise.} \end{cases} \tag{3.40}$$

■

As the name suggests, the primitive relational sensor can be used repeatedly build *compound* relational sensors. The idea is to make a sensor that produces a vector of binary observations, one from each primitive. The resulting observation can be considered as a graph $G_\mathbf{p}$ for which the vertices are the set of bodies and a directed edge exists if and only if $R_\mathbf{p}(i, j)$. As the state changes, the edges in $G_\mathbf{p}$ may change.

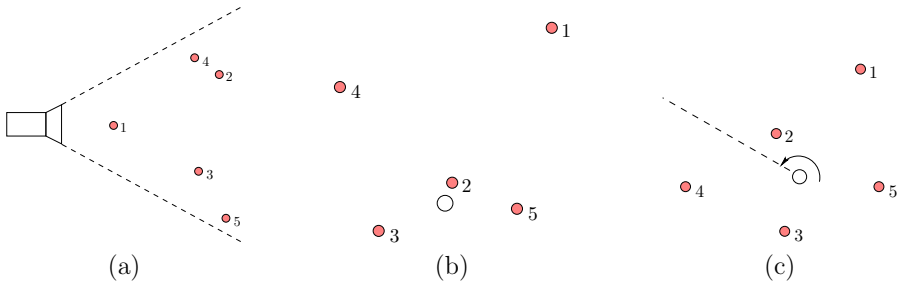


Figure 3.28: Three kinds of compound relational sensors: (a) The linear sensor observes that the landmarks are ordered from left to right as $(4, 2, 1, 3, 5)$. (b) This sensor sorts the landmarks closest to farthest, resulting in the observation $(2, 3, 5, 4, 1)$. (c) The cyclic sensor sweeps counterclockwise and yields the cyclic ordering $(1, 2, 4, 3, 5)$

Some useful compound relational sensor models will now be defined. Suppose there is a finite set of static point bodies in the plane that are distinctly labeled. Consider a relation $L_{\mathbf{p}}$, for which $L_{\mathbf{p}}(i, j)$ means that body $\mathcal{B}_i$ appears to be to the left of $\mathcal{B}_j$ when viewed from the sensor configuration $\mathbf{p} = (x, y, \theta)$. If these are in the field of view of a camera, we should be able to determine the value of the relation for any pair of points. See Figure 3.28(a). Each binary observation that determines whether $L_{\mathbf{p}}(i, j)$ holds is just an application of the primitive, Sensor 20. For all bodies in the detection region, these can be combined to yield a linear ordering of the bodies. In the example, the compound sensor observation would be $\mathbf{y} = (4, 2, 1, 3, 5)$. If the bodies were capable of moving, then any permutation might be possible, and $\mathbf{Y}$ would be the set of all 5! permutations. Thus, we obtain a sensor that produces a linear ordering of the bodies:

Sensor 21 (Linear Permutation Sensor)**.** *A relation $L_{\mathbf{p}}$ is defined which implies $\mathcal{B}_i$ is to the left of $\mathcal{B}_j$ in the detection region, using a predetermined convention for defining "left". The sensor mapping applies to the sensor configuration and the configurations of the bodies, and produces a permutation (linear ordering) of the bodies as the observation.* ■

It is tempting to make primitive relations that have more than two outputs, especially if the bodies appear in some degenerate positions. For example, the sensor might not be able to determine whether b_1 is to the left or right of b_2 because they are perfectly aligned in the sensor view. Such cases can be handled by defining multiple relations. For example, one primitive could be $L_{\mathbf{p}}$, and a new one $A_{\mathbf{p}}$ could indicate whether they are aligned.

Figure 3.28(b) shows how to obtain an alternative permutation based on sorting the bodies from nearest to farthest:

Sensor 22 (Distance Permutation Sensor)**.** *A relation $C_{\mathbf{p}}(i, j)$ is true is and only if $\mathcal{B}_i$ is closer than $\mathcal{B}_j$ to a point that represents the sensor location. This forms a primitive relational sensor. By applying this primitive to every body in the detection region, an observation of the distance ordering is obtained: The bodies are sorted from closest to furthest.*

In a physical implementation, imagine that each body has a radio transmitter. A sensor that measures the signal strengths could in principle sort them according to strength, and hence distance. This would work only under idealized conditions. In practice, it might be preferable to allow the sensor to report that two landmarks are of approximately equal distance away, when it is unable to reliably decide which is further.

For some problems, two-argument relations are insufficient. For example, we might want a primitive observation that tells whether body $\mathcal{B}_k$ is to the left or right of a ray that starts at point $\mathcal{B}_i$ and pierces point $\mathcal{B}_j$. This relation involves triples of points, and can be expressed as $R_{\mathbf{p}}(i, j, k)$. For example, extend Sensor 21 (linear permutation) to a sensor that performs a 360° sweep. In this case, the notion of "left of" is not well defined because of the cyclic ordering. However, for a set of three points, $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$, we can determine whether the cyclic permutation is $(\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$ or $(\mathcal{B}_1, \mathcal{B}_3, \mathcal{B}_2)$. Note that others are equivalent, such as $(\mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_1) = (\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$. The resulting sensor model is:

Sensor 23 (Cyclic Permutation Sensor)**.** *For a set of point bodies in an omnidirectional detection region, the sensor mapping produces a cyclic ordering of the bodies with respect to a clockwise* 360° *sweep.* ■

An example is shown in Figure 3.28(c).

If the bodies are only partially distinguishable, then many interesting relational sensors can be defined by assigning labels to bodies. For example, the sensor might detect that a man is to the left of a woman.

3.4.3 Gap Sensors

This next family of sensor models is closely related to the previous three families. The idea is to report information obtained along the boundary of the detection region $V(\mathbf{p})$, which is denoted as $\partial V(\mathbf{p})$. For most 2D cases, $\partial V(\mathbf{p})$ is a closed curve. To motivate this model, recall Sensor 8 (omnidirectional depth), Figure 3.19(d), and Figure 3.20. The data from the omnidirectional depth sensor are depicted again in Figure 3.29(a), but this time discontinuities or *gaps* in the depth measurements are shown. When sweeping counterclockwise, imagine a sensor that reports:

> A wall, then a gap g_1, then a wall, then a gap g_2, then a wall, ...

The alternation between an obstacle or body and a gap in the distance measurements is the information provided by a gap sensor. In general, a gap sensor

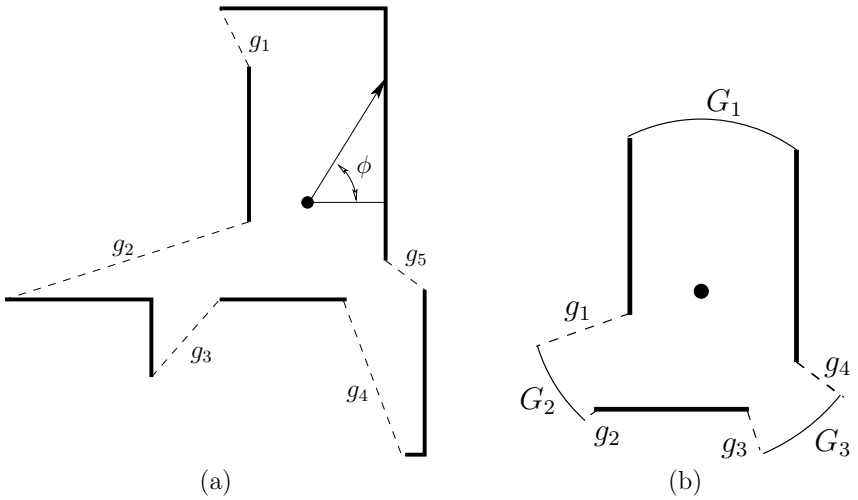


Figure 3.29: Gap sensor models: (a) Five discontinuities in depth are observed. (b) Two kinds of gaps are obtained for a sensor that has limited range.

observation is a sequence, for example $(B_2, g_1, B_3, g_2, B_1)$, which alternates between bodies and gaps. Examples will be given in which this sequence is linear or cyclic. For the mobile robot models in Section 3.3.2, the obstacle can be treated as a static body, so that the observation alternates between gaps and the obstacle boundary.

Sensor 24 (Simple Gap Sensor)**.** *Suppose that a robot carries a sensor with an omnidirectional detection region and is placed into a free space F that is bounded by a simple polygon and contains no interior obstacles. Treating the obstacle O as a special body, the gap sensor for Figure 3.29(a) observes*

$$y = (O, g_1, O, g_2, O, g_3, O, g_4, O, g_5), \tag{3.41}$$

which is interpreted as a cyclic sequence. Since it is impossible to have two consecutive gaps, the O components contain no information, and (3.41) can be simplified to $\mathbf{y} = (g_1, g_2, g_3, g_4, g_5)$. Once again, this observation is cyclic; for example, $\mathbf{y} = (g_3, g_4, g_5, g_1, g_2)$ is equivalent. ■

In reality, most sensors have limited range, which motivates the next model.

Sensor 25 (Depth-Limited Gap Sensor[3])**.** *Suppose that for an omnidirectional sensor, nothing can be sensed beyond some fixed distance. The resulting data from a depth sensor would appear as in Figure 3.29(b). There are two kinds of gaps: one*

[3]This model is based on the one introduced in [12].

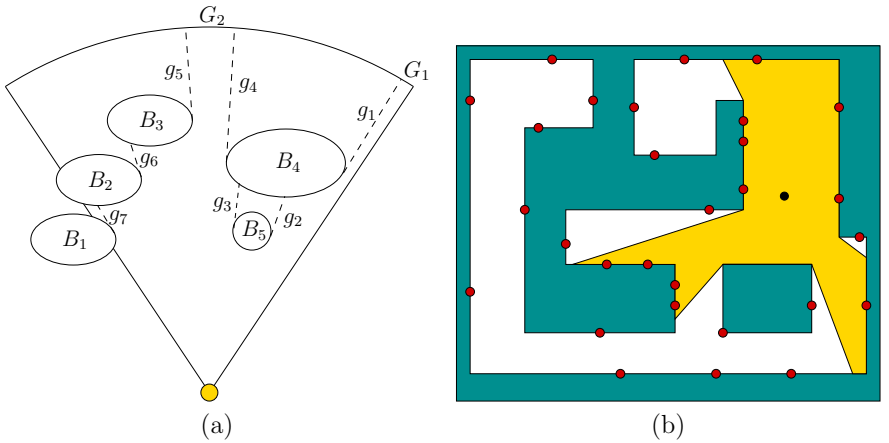


Figure 3.30: (a) A gap sensor among multiple bodies. (b) A sensor that counts landmarks between gaps.

from a discontinuity in depth and the other from a range of angles where the depth cannot be measured because the boundary is too far away. Let the discontinuity gaps be labeled g_i, as before, and the new gaps be labeled G_i. The observation for the example in Figure 3.29(b) is

$$\mathbf{y} = (O, G_1, O, g_1, G_2, g_2, O, g_3, G_3, g_4), \tag{3.42}$$

which again is a cyclic sequence. In contrast to Sensor 24 (simple gap), the appearances of O cannot be deleted without losing information. ■

If there are multiple bodies, then the observation includes the body label:

Sensor 26 (Multibody Gap Sensor)**.** *Suppose there are multiple bodies, as shown in Figure 3.30(a). The sensor sweeps from right to left, and is not omnidirectional. In this case, the observation is a linear sequence,*

$$\mathbf{y} = (G_1, g_1, B_4, g_2, B_5, g_3, B_4, g_4, G_2, g_5, B_3, g_6, B_2, g_7, B_1). \tag{3.43}$$

■

For Sensor 26, it was assumed that the bodies are completely distinguishable. As in Sensor 19, it is once again possible assign labels to be bodies. In this case, Sensor 26 could be extended so that the observation yields a sequence of gaps and labels, as opposed to gaps and bodies.

Following along these lines, the next model simply counts the number of bodies between gaps. It is based on a model called the *combinatorial visibility vector* in [6].

Sensor 27 (Landmark Counter)**.** *Let F be bounded with no interior holes. Let the bodies be a finite set of points that are static and distributed at distinct locations along the boundary of F. All bodies are assigned a common label, such as "feature", meaning that they are completely indistinguishable. When in the interior of F, the sensor observation is a cyclic sequence of integers, corresponding to the number of bodies between each pair of gaps. The observation for the example in Figure 3.30(b) is* $\mathbf{y} = (3, 3, 4, 0, 1)$. ■

This sensor can be further adapted and extended in several ways:

1. A linear sequence could be obtained by placing the sensor on the boundary, or by observing the starting point of the omnidirectional sweep.
2. Any level of partial or full distinguishability of bodies could be allowed.
3. The bodies could be placed in the interior of the detection region.
4. The bodies could be capable of motion.

3.4.4 Field Sensors

These virtual sensors are motivated by directly measuring waves that propagate through the free space. However, they are more general than that. A *field* is a function

$$f : \mathbb{R}^n \to \mathbb{R}^m. \tag{3.44}$$

The domain of f can be imagined as a position in the physical world. For most of our models, $n = 2$, but $n = 3$ is also useful. Each $f(x, y)$ for a position (x, y) produces an m-dimensional vector. Usually, $m \leq n$. Figure 3.31 shows some simple examples.

The case of $m = 1$ is itself is useful, in which the field yields a scalar value at every position:

$$f : \mathbb{R}^2 \to [0, \infty). \tag{3.45}$$

The scalar values could represent the magnitudes of the vectors at each position. This could correspond, for example to a radio signal or the amount of visible light. If an energy source propagates through a 3D space, the amount that its observed at a sensor is proportional to the distance squared:

$$\mathbf{y} = k/d^2, \tag{3.46}$$

for some constant k and distance d from the source. A more unusual example is to define $f(x, y) = 1$ if $(x, y) \in O$ and $f(x, y) = 0$ if $(x, y) \in F$. This causes a clear division of $\mathbb{R}^2$ into an obstacle O and free space F. Rather than having sharp obstacle boundaries, a real value could be assigned at each location to yield *altitude* of an outdoor terrain, as shown in Figure 3.32:

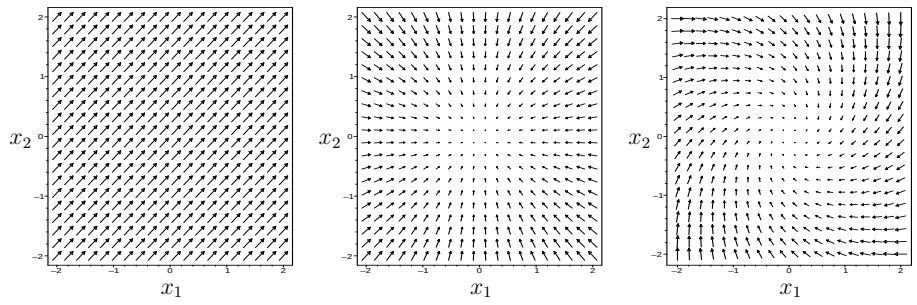


Figure 3.31: The physical world may contain vector fields on which to base sensor measurements: (a) $f(x, y) = (1, 1)$, (b) $f(x, y) = (-x, -y)$, (c) $f(x, y) = (y - x, -x - y)$.

Sensor 28 (Altimeter)**.** *Let $f(x, y)$ represent altitude for each position (x, y). If f is given, then $\mathbf{P} = \mathbb{R}^2 \times S^1$ and the sensor mapping is*

$$\mathbf{y} = h(x, y, \theta) = f(x, y). \tag{3.47}$$

If f is not given, then let Φ represent the set of all possible f. It is assumed that all functions Φ satisfies some realistic properties, such as bring continuous or having an upper limit on slope. The P-space becomes

$$\mathbf{P} = \mathbb{R}^2 \times S^1 \times \Phi \tag{3.48}$$

and the sensor mapping becomes

$$\mathbf{y} = h(x, y, \theta, f) = f(x, y). \tag{3.49}$$

■

In the physical world, the altimeter could be implemented using air pressure. Under water, a variant can be made to measure depth based on water pressure. The virtual sensor model could also be extended to consider obstacles, which block motion.

Now suppose that the world is two-dimensional and a given 2D field $f : \mathbb{R}^2 \to \mathbb{R}^2$ is propagating through it. To keep it simple, there are no obstacles. A simple sensor observes the values:

Sensor 29 (Direct Field Sensor)**.** *A vector field*

$$f(x, y) = (f_1(x, y), f_2(x, y)) \tag{3.50}$$

Figure 3.32: A NASA-constructed image of the Cape Peninsula, South Africa. At each position, both the elevation and its appearance from Landsat are matched.

is given to the robot. The sensor mapping is

$$\mathbf{y} = h(x, y, \theta) = (f_1(x, y), f_2(x, y)), \tag{3.51}$$

which yields a two-dimensional observation vector. ■

Rather than observing the entire vector, it might only observe the magnitude:

Sensor 30 (Direct Intensity Sensor)**.** *For a given 2D vector field $f(x, y)$, the sensor mapping is*

$$h(x, y, \theta) = \|f(x, y)\| = \sqrt{f_1^2(x, y) + f_2^2(x, y)}, \tag{3.52}$$

which yields a nonnegative real intensity *value.* ■

For radio signals, this could be implemented using a non-directional signal meter.

Recall that a distance threshold was used in Sensor 10 to make a proximity sensor. The field intensity can be used in the same way:

Sensor 31 (Intensity Alarm)**.** *For a fixed value ϵ, an "alarm" can be made that indicates when the intensity is above ϵ:*

$$h(x, y, \theta) = \begin{cases} 1 & \text{if } \|f(x, y)\| \geq \epsilon \\ 0 & \text{otherwise.} \end{cases} \tag{3.53}$$

■

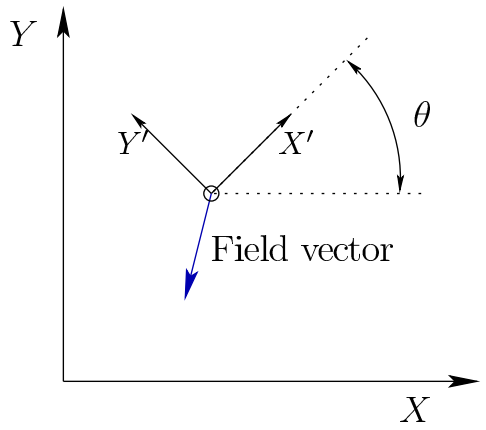


Figure 3.33: If the sensor is rotated by θ, then the observed field vector $f(x, y)$ is rotated by $-\theta$ with respect to the coordinate system in which the field is defined. To recover the vector in this frame, it must be rotated by θ.

In most settings, it is unreasonable to expect to recover the precise magnitude. For example, maybe the output power of a radio transmitter is not known. We might nevertheless want to develop a sensor that returns higher values as the intensity increases. This motivates:

Sensor 32 (Transformed Intensity Sensor)**.** *Let $g : [0, \infty) \to [0, \infty)$ be any strictly monotonically increasing smooth function. The sensor mapping is*

$$h(x, y, \theta) = g(\|f(x, y)\|). \tag{3.54}$$

If the observations $h(x)$ are linearly proportional to the field intensity, then g is a linear function. In general, g may be nonlinear. ■

To make the model more interesting, g might not be given. In this case, the set of possible g functions becomes a component of the P-space and g becomes part of the P-state (in other words, $\mathbf{p} = (x, y, \theta, g)$). Such a sensor can still provide useful information. For example, if $y = h(x)$ is increasing over time, then we know that the radio transmitter is closer, even though g is unknown.

The fields have been defined in terms of a fixed coordinate system that was used to specify the world. As a sensor rotates, however, it will observe the field *relative* to its own coordinate system; see Figure 3.33.

Sensor 33 (Relative Field Vector Observation)**.** *For a given 2D vector field $f(x, y)$, the relative observation is given by the sensor mapping,*

$$\mathbf{y} = h_{rv}(x, y, \theta) = R(-\theta) f(x, y), \tag{3.55}$$

in which $R(\cdot)$ is a 2D rotation matrix. ■

Consider how this sensor could be used. If f is given and θ is unknown, then θ can be determined using (3.55) and solving the two linear equations. Likewise, if θ is known and f is unknown, then the field vector value $f(x, y, \theta)$ in the fixed coordinate frame can be calculated as

$$f(p) = R(\theta)h_{rv}(x). \tag{3.56}$$

These concepts extend to a 3D world, in which $R(\theta)$ is replaced in (3.56) by a 3 by 3 rotation matrix R. In (3.55), $R(-\theta)$ is replaced by the inverse rotation, which is R^T, the transpose of R.

For an example of these concepts in the physical world, suppose a magnetometer is used to measure the earth's magnetic field, which is a 3D vector at any point on the earth. The field is known and the sensor observation is used to determine the orientation of the sensor. Even when holding a simple compass, the orientation of the compass is determined by how many degrees away from magnetic north that compass is pointing. These concepts can be used to construct Sensor 4, which is an idealized compass that behaves as a projection sensor. To describe its operation with some mathematical details, then Sensor 33 is used to define the field and account for the sensor's rotation.

3.5 Sensors That Misbehave

Recall from Chapter 2 that for describing how robots move, it was useful to define mathematical models of the kinematics that perfectly characterize what should happen before unexpected disturbances are accounted for. The same is true here. Up until now, virtual sensors have been characterized by the information they are supposed to provide for their intended use. As in the case of kinematics, the sensor observations obtained in practice may be disturbed from their nominal values. This introduces another source of uncertainty; however, it is important to remember that most uncertainty due to sensor arises from the course partitions of the P-space. This is a side effect of the sensor mapping. As in Section 2.4, there are two main representations of the uncertainty due to disturbances: nondeterministic and probabilistic.

3.5.1 Nondeterministic Disturbance

Suppose that a sensor provides an observation instantaneously, but there is uncertainty about which observation will occur at P-state $\mathbf{p}$. By allowing nondeterministic uncertainty, the sensor mapping specifies a *set of possible observations* that could be obtained from some P-state.

Let an observation space $\mathbf{Y}$ be an ordinary observation space and let $\mathbf{P}$ be any P-space. As in Section 2.4, we define a disturbance set that interferes with the

outcome. Let $\mathbf{V}$ denote a set of possible disturbances. A *nondeterministic sensor mapping* is defined as

$$\mathbf{y} = h(\mathbf{p}, \mathbf{v}), \tag{3.57}$$

in which $\mathbf{p}$ is the P-state and $\mathbf{v} \in \mathbf{V}$ is the disturbance. To make the definition more general, the set of allowable disturbances could depend on $\mathbf{p}$, resulting in $\mathbf{V}(\mathbf{p})$.

As a simple example if (3.57), suppose that a sensor is supposed to directly measure a one-dimensional position $\mathbf{p} = x$. In the perfect case, $\mathbf{y} = h(\mathbf{p}) = x$. However, we want to account for the possibility of measurement error up to size ϵ.

Sensor 34 (One-Dimensional Position Sensor)**.** *Let $\mathbf{P} = \mathbf{Y} = \mathbb{R}$ and let ϵ be the maximum amount of measurement error. The disturbance set is $\mathbf{V} = [-\epsilon, \epsilon]$ and the sensor mapping is:*

$$h(x, v) = x + v, \tag{3.58}$$

in which $x \in \mathbf{P}$ and $v \in \mathbf{V}$. ■

The sensor mapping does not predict exactly which observation will occur. For a sensor model defined using (3.57), the set of possible observations from P-state $\mathbf{p}$ is:

$$\mathbf{Y}_h(\mathbf{p}) = \{\mathbf{y} \in \mathbf{Y} \mid \exists \mathbf{v} \in \mathbf{V} \text{ for which } \mathbf{y} = h(x, y)\} \tag{3.59}$$

This is similar to the forward projections of Section 2.4.1. Any $\mathbf{y} \in \mathbf{Y}_h(\mathbf{p})$ could be produced as the observation. For example, using (3.58), $\mathbf{Y}_h(x) = [x - \epsilon, x + \epsilon]$. If $x = 2$, then actual observation $\mathbf{y}$ may be any value in $[2 - \epsilon, 2 + \epsilon]$.

The preimage idea from Section 3.3 extends naturally to handle nondeterministic disturbances. The preimage of an observation $\mathbf{y}$ is

$$h^{-1}(\mathbf{y}) = \{\mathbf{p} \in \mathbf{P} \mid \exists \mathbf{v} \in \mathbf{V} \text{ for which } \mathbf{y} = h(\mathbf{p}, \mathbf{v})\}. \tag{3.60}$$

For Sensor 34, the preimage of an observation $\mathbf{y}$ is $[\mathbf{y} - \epsilon, \mathbf{y} + \epsilon]$.

The preimages given by (3.60) do not form a partition of $\mathbf{P}$ as in the case of the original sensor mapping $\mathbf{y} = h(\mathbf{p})$. The problem is that a P-state $\mathbf{p}$ may appear in many different preimages. This is due to the fact that each P-state can produce many possible observations. They instead form a *cover* of $\mathbf{P}$. This means that the union of all preimages $h^{-1}(\mathbf{y})$ is equal to $\mathbf{P}$; however, the preimages are not necessarily disjoint. Intuitively, the preimages are "thickened" to account for disturbances. For Sensor 34, select $\epsilon = 0$. In this case, the ordinary preimages from Section 3.3 are obtained. If $\epsilon > 0$, then a partition is no longer obtained because the preimages overlap. As ϵ grows, the amount of preimage overlap grows. If we went overboard and allowed $\mathbf{V} = \mathbb{R}$, then every preimage would be $h^{-1}(\mathbf{y}) = \mathbb{R}$. This is a cover of $\mathbf{P} = \mathbb{R}$ in which every preimage is simply $\mathbb{R}$. Instead of a partition, it is more like an infinite stack of pancakes.

Sensor 34 can be easily extended to k dimensions, usually with $k = 2$ or $k = 3$. The set $\mathbf{V}$ may form a disc, ball, square, cube, or other shapes.

Now consider measuring the distance to an obstacle. A disturbance $\mathbf{v}$ can be inserted into the depth sensors of Section 3.3.2 and many others, such as field sensors of Section 3.4.4. In many cases, the disturbance can be added to the observation produced by the perfect sensor mapping. For other kinds of sensors, such as detection, the disturbance parameter may affect the output differently. The set $\mathbf{V}$ could represent a finite number of discrete modes that affect the outcome. Alternatively, $\mathbf{V}$ could represent various lighting levels that affect that ability of a sensor to detect bodies.

Consider modifying a static binary sensor given by (3.33). The sensor might produce a *false positive* by yielding $h(x, y, \theta, \mathbf{v}) = 1$ even though (x, y) is not in the detection region V. In this case, the preimage would be $h^{-1}(1) = \mathbf{P}$. If the sensor could also produce a *false negative* by yielding $h(x, y, \theta, \mathbf{v}) = 0$ when (x, y) lies in V, then $h^{-1}(0) = \mathbf{P}$. These two preimages together cover $\mathbf{P}$ twice, and we clearly see that the sensor is absolutely worthless under this model: We can infer nothing about the state from the observation if false negatives and false positives are permitted. In practice, a sensor that commits such errors might nevertheless be useful, but probabilistic modeling is then needed (how likely is it to make a mistake?); this is the subject of Section 3.5.2.

3.5.2 Probabilistic Disturbance

Perhaps we have been observing a sensor over many trials and are able to better characterize the disturbances. Rather than simply talking about the *set* of possible observations, we could statistically learn a probability density over them. Start with (3.59), which provides the set $\mathbf{Y}_h(\mathbf{p})$ of possible observations given $\mathbf{p}$. The models in this section place a probability density over $\mathbf{Y}_h(\mathbf{p})$. A convenient way to express this is

$$p(\mathbf{y}|\mathbf{p}), \tag{3.61}$$

which is a probability density function over $\mathbf{Y}_h$, but conditioned on the particular state, $\mathbf{p} \in \mathbf{P}$. Unfortunately, this representation hides the underlying sensor mapping. Furthermore, all of the important preimage and cover information is obscured. It is therefore critical when using probabilistic models to recall and utilize the underlying structure of the sensor mapping h.

The process of developing (3.61) usually proceeds as:

1. Start with a perfect sensor model of the form $\mathbf{y} = h(\mathbf{p})$, as introduced in Sections 3.3 and 3.4.

2. Model possible disturbances, to obtain $\mathbf{V}$.

3. Define observations in terms of P-states and disturbances, to obtain $\mathbf{y} = h(\mathbf{p}, \mathbf{v})$, as introduced in Section 3.5.1.

4. Determine a suitable probability density function $p(\mathbf{v})$ over the disturbances.

5. Transform $p(\mathbf{v})$ using $\mathbf{y} = h(\mathbf{p}, \mathbf{v})$ to obtain the density $p(\mathbf{y}|\mathbf{p})$.

Furthermore, all of the important preimage and cover information is obscured. It is therefore critical when using probabilistic models to recall and utilize the underlying structure of the sensor mapping h.

Some probabilistic sensor models will now be defined. We first make a probabilistic variant of Sensor 34 (one-dimensional position).

Sensor 35 (Probabilistic 1D Position Sensor)**.** *Let* $\mathbf{P} = \mathbf{Y} = \mathbb{R}$ *and let the disturbance* $\mathbf{v}$ *be chosen according to a Gaussian density with zero mean and variance* σ^2*. The probability density function is*

$$p(\mathbf{y}|\mathbf{p}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\mathbf{p}-\mathbf{y})^2}{2\sigma^2}}. \tag{3.62}$$

■

Note that $p(\mathbf{y}|\mathbf{p})$ is maximized when $\mathbf{p} = \mathbf{y}$, which corresponds to the case of no disturbance. A multidimensional extension can be made by applying a multivariate Gaussian density to characterize the higher-dimensional disturbance.

Probabilistic models can be made for disturbances applied to most of the virtual sensor models of this chapter. For a simple example, the faulty binary sensor from Section 3.5.1 can be extended by attaching probabilities to false positives and false negatives. Recall that in the nondeterministic model, the sensor was useless because its output was completely unreliable. It nevertheless becomes useful in the probabilistic setting due to a history of statistical observations that partly characterize its behavior. For a false positive, we need to define $p(\mathbf{y} = 1 \mid b \notin V)$. The false negative case is $p(\mathbf{y} = 0 \mid b \in V)$. The other two combinations (true positives and negatives) can be derived from these. If the probabilities of false negatives and positives are small, then the sensor remains quite valuable. If both are $1/2$, then the sensor is useless. Interestingly, if both are close to one then, the sensor is once again quite useful because it is almost always expected to give the opposite result! This just a strange calibration issue.

For a more interesting example, consider making a probabilistic model of a depth sensor, such as Sensor 5. Many factors could contribute to unexpected measurement error. In practice, people often design probability densities empirically to account for all possibilities. For example, in [13], errors in depth measurements are modeled by four different probability density functions:

1. A Gaussian density that accounts for small noise in the measured distance.

2. An exponential density that decreases with distance, to account for unexpected obstacles in the environment.

3. A discrete probability is assigned to the case in which no depth measurement is obtained. It accounts for failure to detect the reflected signal from the obstacle.

4. A uniform density function that accounts for all other possibilities that are not explained by the previous three densities.

Let these four densities be written in the form $p_i(\mathbf{y}|\mathbf{p})$. The probability density function that takes into account all four sources is formed as

$$p(\mathbf{y}|\mathbf{p}) = \alpha_1 p_1(\mathbf{y}|\mathbf{p}) + \alpha_2 p_2(\mathbf{y}|\mathbf{p}) + \alpha_3 p_3(\mathbf{y}|\mathbf{p}) + \alpha_3 p_3(\mathbf{y}|\mathbf{p}), \tag{3.63}$$

in which α_1 to α_4 are empirically chosen positive weights. To ensure that a valid density function is obtained, the weights must sum to one. Although it is hard to rigorously justify the resulting density, it has been shown to work well in practice.

3.6 Sensing over Time

To conclude this chapter, we finally consider what happens as the P-state changes over time and the sensors respond accordingly. This helps connect the sensor models to the kinematic models of Chapter 2, where robots change their configurations over time. This section is also a step toward filters, which integrate information gathered over time from sensors to form I-states. That will be covered thoroughly in Chapter 5.

3.6.1 State-Time Space

Let T refer to an interval of time, in which the most convenient case is $T = [0, \infty)$, which stems from the belief that the world will not end. Sometimes, we might declare a final time t_{final}, in which case $T = [0, t_{final}]$. Starting from any P-space $\mathbf{P}$, we obtain the *P-state-time space*:

$$\mathbf{Z} = \mathbf{P} \times T. \tag{3.64}$$

Each element $\mathbf{z}$ in $\mathbf{Z}$ can be represented as a pair $\mathbf{z} = (\mathbf{p}, t)$, in which $\mathbf{p}$ is the P-state at time t.

Envision the P-state changing from moving obstacles, robots, or other bodies as an "animation". Since time always marches forward, we can consider the animation as a path through $\mathbf{Z}$ that is parametrized by time. This leads to a *P-state trajectory*,

$$\tilde{\mathbf{p}} : T \rightarrow \mathbf{P}. \tag{3.65}$$

The value $\tilde{\mathbf{p}}(t) \in \mathbf{P}$ represents the P-state at time t. The value $\tilde{\mathbf{p}}(0)$ is called the *initial P-state*. See Figure 3.34, which shows two interpretations of $\tilde{\mathbf{p}}$:

1. A curve that is drawn in $\mathbf{P}$. As usual whenever curves are drawn, the parameter used to trace the curve cannot be seen. In this case it is time t.

2. A one-dimensional set is drawn that crosses $\mathbf{Z}$ from left to right as time increases. This shows the precise timing for each P-state. Note that an "S" shape is not allowed here because time must go forward.

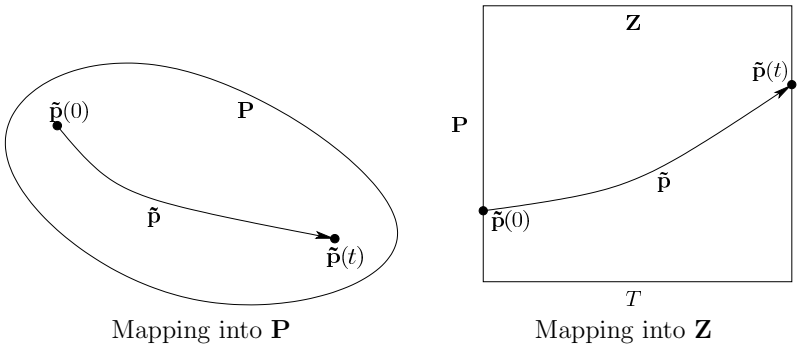


Figure 3.34: A trajectory can be viewed as a time-parametrized path through $\mathbf{P}$ or a one-dimensional set that traverses $\mathbf{Z}$ from left to right.

3.6.2 Time Sensors

To make sensors that observe timing information, first note that $\mathbf{P}$ has been extended to $\mathbf{Z}$. This implies that sensor observations should depend on $(\mathbf{P}, t) \in \mathbf{Z}$, which includes both the P-state and its corresponding time. The original sensor mapping (3.14) is replaced by

$$h : \mathbf{Z} \rightarrow \mathbf{Y}, \tag{3.66}$$

for which we write $\mathbf{y} = h(\mathbf{z})$, or equivalently, $\mathbf{y} = h(\mathbf{p}, t)$.

All of the concepts from Section 3.3 extend naturally from $\mathbf{P}$ to $\mathbf{Z}$. A preimage under the sensing model in (3.66) is

$$h^{-1}(\mathbf{y}) = \{(\mathbf{p}, t) \in \mathbf{Z} \mid \mathbf{y} = h(\mathbf{p}, t)\}, \tag{3.67}$$

Now consider partitions $\Pi(h)$ over $\mathbf{Z}$. A weak sensor may partition $\mathbf{Z}$ into large chunks of state-time space. Following Section 3.3.3, a sensor h_1 dominates another h_2 if and only if its partition $\Pi(h_1)$ of $\mathbf{Z}$ is a refinement of $\Pi(h_2)$. In the same way as for $\mathbf{P}$, a lattice of all partitions of $\mathbf{Z}$ is obtained.

Misbehaving sensors can also be made by extending the concepts from Section 3.5. For nondeterministic models, the sensor mapping is $\mathbf{y} = h(\mathbf{z}, \mathbf{v})$, in which $\mathbf{v}$ is the disturbance parameter. The probabilistic version is specified by a density $p(\mathbf{y}|\mathbf{z})$, which is written equivalently as $p(\mathbf{y}|\mathbf{p}, t)$.

Here is a simple time sensor:

Sensor 36 (Perfect Clock)**.** *The sensor mapping is*

$$\mathbf{y} = h(\mathbf{z}) = h(\mathbf{p}, t) = t, \tag{3.68}$$

which reports the current time. ■

This sensor can be combined with any of the previous sensors in this chapter by replacing $\mathbf{P}$ with $\mathbf{Z}$. This allows a perfect time stamp to be associated with each observation. Here is an example that extends Sensor 14 (static binary detector):

Sensor 37 (Detector with Time Stamp)**.** *A single point body b moves in F and the detection region is $V \subset F$. The sensor mapping is*

$$h(b) = \begin{cases} (1, t) & \text{if } b \in V \text{ at time } t \\ (0, t) & \text{otherwise.} \end{cases} \tag{3.69}$$

For each observation, its associated time is observed. ■

It is usually the case that time is observed without perfect precision. An oscillator provides a series of pulses that are equally spaced over time (recall the 555 timer from Section 3.1.2). In this case, quantized time is obtained:

Sensor 38 (Stage Sensor)**.** *Let Δt be a fixed time interval, which usually corresponds to the oscillation period of a clock circuit. The sensor mapping is*

$$\mathbf{y} = h(\mathbf{p}, t) = \lfloor t/\Delta t \rfloor, \tag{3.70}$$

in which $\lfloor \cdot \rfloor$ means that floor (rounding down to the nearest integer). ■

The observation is an integer that is commonly called the *stage*. The stage in which it was observed could be associated with any of the sensor models introduced before this section. We can even write $\mathbf{y}_k$ to indicate the observation taken at stage k, which corresponds to time $k\Delta t$.

Sensors may yield significant ambiguity with regard to what time the measurement was taken. For example, perhaps the sensor reports that an observation $\mathbf{y}_k$ occurred at stage k, and it is known only that the observation came later than those from earlier stages. This can be modeled by using the same trick as in Sensor 32 by introducing an intermediate function that transforms the observation. This leads to the next sensor model, which provides a stage observation whenever it wants to. This act is called an *event*.

Sensor 39 (Event-Driven Stage Sensor)**.** *Let $g : T \rightarrow [0, \infty)$ be an unknown function that increases monotonically with respect to time. Let G be the set of all such functions. Starting with any P-space $\mathbf{P}$, the sensor mapping is:*

$$\mathbf{y} = h(\mathbf{p}, t, g) = \lfloor g(t) \rfloor. \tag{3.71}$$

The domain of the sensor is $\mathbf{P} \times T \times G$ to account for the unknown g.

For another example of ambiguity regarding time, imagine being on the equator of a spinning planet. The sun shines directly on the planet, illuminating exactly half of it at any time. See Figure 3.35(a). A light detector is placed on the

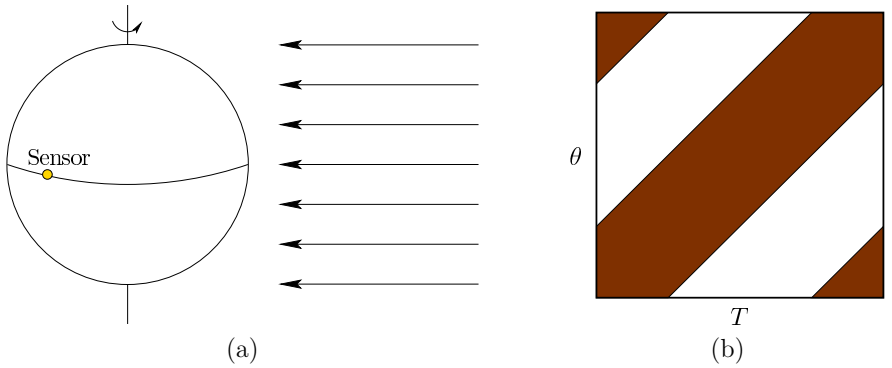


Figure 3.35: (a) Imagine placing a light sensor at an unknown location along the equator of a rotating planet. (b) The sensor preimages partition $\mathbf{Z}$ into light and dark regions that combine both location and time information.

equator so that it observes 1 if there is light, and 0 otherwise. Let T be the period of rotation of the planet, resulting in a full day. Let $\mathbf{P}$ represent the longitude of the sensor along the equator. This is a single angle $\theta \in [0, 2\pi)$. Suppose that the position of the sensor and the time of day are unknown. The sensor mapping yields $h(\theta, t) = 1$ or $h(\theta, t) = 0$, depending whether its senses light. The resulting preimages correspond to "light" and "dark" and are shown in Figure 3.35(b). At any θ, light is observed exactly half of the time. At any time, exactly half of the planet is lit. If 1 is observed, then some information is clearly gained about both the position and the time; however, it is impossible to determine either with precision.

3.6.3 Velocities and Accelerations

Recall from Section 2.3.3 that time derivatives of the configuration could be included in the P-state. These could correspond to the rate of change of the robot, obstacles, or other bodies. Accelerations may also be included. Using a time sensor, the rate of change of a P-state variable could be estimated. For example, if the position is 5 at time $t = 0$ and 6 at time $t = 1/10$, then an estimate of the speed is 10 units per second. In general, velocities and accelerations can be estimated from observations that occur closely over time. A well-known problem with this approach is that noise in the observations becomes greatly amplified in the derivative estimate.

In many cases, velocities and accelerations can be measured directly from the physical world, instead of deriving estimates. Imagine sitting on a merry-go-round that is rotating at constant angular velocity. We would feel a constant centripetal force that attempts to pull us off of the edge. Measuring the weight of this force and introducing calibration provides a direct observation of the angular velocity.

Virtual sensors that measure velocities and accelerations directly are straightforward to define once these quantities are included in the P-space. For example, let the P-state be

$$\mathbf{p} = (x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}, \ddot{x}, \ddot{y}, \ddot{\theta}), \tag{3.72}$$

in which (x, y, θ) represents the configuration of a robot. The linear velocity of the special point is $(\dot{x}, \dot{y})$ and $\dot{\theta}$ is the angular velocity. The linear acceleration is $(\ddot{x}, \ddot{y})$ and the angular acceleration is $\ddot{\theta}$.

Any of these quantities could be measured by applying the projection sensor (Sensor 3). For example, a 2D gyroscope could be modeled as $\mathbf{y} = h(\mathbf{p}) = \dot{\theta}$. A linear accelerometer could be modeled as $\mathbf{y} = h(\mathbf{p}) = (\ddot{x}, \ddot{y})$.

3.6.4 Sensors with Memory

As a natural transition to the temporal filters of Chapter 5, we consider one final extension to the sensor models. It might be the case that the sensor observation depends on a *history* of previous P-states. The most common examples in practice are odometers, such as the wheel encoder in Figure 3.7(b). They accumulate changes over time and report the aggregate amount, such as total distance traveled. The relationship to Chapter 5 is that the sensors here could be realized by employing a filter that uses information from instantaneous sensors (of the form $h : \mathbf{p} \rightarrow \mathbf{y}$). In other words, a history-based sensor usually contains a built-in filter. Without getting into those details yet, imagine the sensor as a "black box" that receives P-states and then outputs an observation at some later time.

Let $\tilde{\mathbf{p}} : [0, t] \rightarrow \mathbf{P}$ represent a *P-state history*, which yields the P-states from time 0 up to time t. This is simply a trajectory over the time interval $[0, t]$. We want to define virtual sensors that take $\tilde{\mathbf{p}}$ as input and produce an observation. Therefore, let $\tilde{\mathbf{P}}$ be the set of *all* possible histories, which is called the *history space*. The set $\tilde{\mathbf{P}}$ includes histories of various stopping times. In other words, possible histories up to time 1 and up to time 5 are included in $\tilde{\mathbf{P}}$.

The sensor mapping is

$$h : \tilde{\mathbf{P}} \rightarrow \mathbf{Y}. \tag{3.73}$$

In this case, a given state trajectory $\tilde{\mathbf{p}} : [0, t] \rightarrow \mathbf{P}$ produces an observation $\mathbf{y} = h(\tilde{\mathbf{p}})$ at time t.

Once again, the notions of preimages, dominance, partitions, and the sensor lattice naturally extend from $\mathbf{P}$ to $\tilde{\mathbf{P}}$. The preimages are

$$h^{-1}(\mathbf{y}) = \{\tilde{\mathbf{p}} \in \tilde{\mathbf{P}} \mid \mathbf{y} = h(\tilde{\mathbf{p}})\}. \tag{3.74}$$

This yields the set of possible histories in $\tilde{\mathbf{P}}$ that produce the same $\mathbf{y}$. The preimages induce a partition of $\tilde{\mathbf{P}}$, and all history-based sensors can be arranged into a sensor lattice over $\tilde{\mathbf{P}}$.

Some useful examples of history-based sensors follow. The first one indicates how far the robot has traveled:

Sensor 40 (Linear Odometer)**.** *Let $(\dot{x}, \dot{y})$ represent the linear robot velocity. A history-based sensor could integrate the magnitude of velocity obtain the total distance traveled:*

$$\mathbf{y} = h(\tilde{\mathbf{p}}) = \theta_0 + \int_0^t \sqrt{\dot{x}^2 + \dot{y}^2} ds. \tag{3.75}$$

■

Note that time derivatives of $\mathbf{p}$ are not explicitly included in the P-state because they can be derived from the history by taking the time derivative of $\tilde{\mathbf{p}}$. An alternative is to integrate $\dot{x}$ and $\dot{y}$ directly, which calculates the current displacement with respect to the starting position.

This sensor uses only orientation information:

Sensor 41 (Angular Odometer)**.** *The sensor mapping is*

$$\mathbf{y} = h(\tilde{\mathbf{p}}) = \theta_0 + \int_0^t \dot{\theta}(s) ds, \tag{3.76}$$

in which $\mathbf{y}$ measures the net orientation change from some starting orientation. ■

An alternative above would be to integrate $|\dot{\theta}(s)|$, which would yield to total amount of rotation that occurred.

In practice, sensors cannot actually produce instantaneous observations. Instead, there is *latency*, which is the time between the occuring stimulus and the resulting observation. Using a history-based sensor, the latency can be explicitly modeled:

Sensor 42 (Delayed Measurement)**.** *Suppose a sensor measures the P-state perfectly, but it takes one unit of time to output the result. This can be modeled as*

$$\mathbf{y} = \begin{cases} \tilde{\mathbf{p}}(t-1) & \text{if } t \geq 1 \\ \# & \text{otherwise,} \end{cases} \tag{3.77}$$

in which # means that the P-state cannot yet be determined. ■

A delayed version of any sensor of the form $h : \mathbf{P} \rightarrow \mathbf{Y}$ or $h : \mathbf{Z} \rightarrow \mathbf{Y}$ can be made in this way.

Without referring directly to velocities, a history-based sensor can be constructed that estimates the distance traveled by comparing positions reported at various times.

Sensor 43 (Discrete-Time Odometer)**.** *Let Δt be a fixed time interval. Let (x_t, y_t) denote the robot position at time t, which can be determined from $\tilde{\mathbf{p}}(t)$. The sensor mapping is*

$$h(\tilde{\mathbf{p}}) = \sum_{i=1}^{\lfloor t/\Delta t \rfloor} \sqrt{(x_{i\Delta t} - x_{(i-1)\Delta t})^2 + (y_{i\Delta t} - y_{(i-1)\Delta t})^2}. \tag{3.78}$$

■

For a state trajectory $\tilde{\mathbf{p}} : [0, t] \rightarrow \mathbf{P}$, the total distance traveled is estimated. The quality of the estimate depends on how small Δt is selected. This sensor incorporates a simple temporal filter, which will be covered in Chapter 5.