Filtering and Planning in Information Spaces*

Steven M. LaValle
Department of Computer Science
University of Illinois

Abstract

This tutorial presents a fresh perspective on modeling sensors and then using them for filtering and planning. The concepts and tools are motivated by many problems of current interest, such as tracking, monitoring, navigation, pursuit-evasion, exploration, and mapping. First, an overview of sensors that appear in numerous systems is presented. Following this, the notion of a virtual sensor is explained, which provides a mathematical way to model numerous sensors while abstracting away their particular physical implementation. Dozens of useful models are given. In the next part, a new perspective on filtering is given based on information spaces. This includes classics such as the Kalman and Bayesian filters; however, it also opens up a new family of reduced-complexity filters that try to maintain as little information as possible while performing their required task. Finally, the planning problem is presented in terms of filters and information spaces.

Contents

*Technical Report, Dept. of Computer Science, University of Illinois at Urbana-Champaign, October 2009. This paper accompanied an IROS tutorial in St. Louis, USA, on 11 Oct. 2009.

1 Introduction

Think about the devices we build that intermingle sensors, actuators, and computers. Whether they be robot systems, autonomous vehicles, sensor networks, or embedded systems, they are completely blind to the world until we equip them with sensors. All of their accomplishments rest on their ability to sift through sensor data and make appropriate decisions. This tutorial therefore takes a completely *sensor-centric* view for designing these systems.

It is tempting (and common) to introduce the most complete and accurate sensors possible to eliminate uncertainties and learn a detailed, complex model of the surrounding world. In contrast, this tutorial heads in the opposite direction by starting with sensing first and then *understanding* what information is minimally needed to solve specific tasks. If we can accomplish our mission without knowing certain details about the world, then the overall system may be more simple and robust.

This can be partly understood by considering computational constraints. One way or another, we want computers to process and interpret the data obtained from sensors. The computers might range from limited embedded systems to the most powerful computer systems. The source of their data is quite different from classical uses of computers, in which data are constructed by humans, possibly with the help of software. When data are obtained from sensors, there is a direct *sensor mapping* from the physical world onto a set of sensor readings. Even though sensors have been connected to computers for decades, there has been a tendency to immediately digitize the sensor data and treat it like any other data. With the proliferation of cheap sensors these days, it is tempting to easily gather hordes of sensor data and google them for the right answer. This may be difficult to accomplish, however, without carefully understanding the sensor mapping. A large part of this tutorial is therefore devoted to providing numerous definitions and examples of practical sensor mappings.

When studying sensors, one of the first things to notice is that most sensors leave a huge amount of ambiguity with regard to the state of the physical world. Example: How much can we infer about the world when someone triggers an infrared sensor to turn on a bathroom sink? In many fields, there is a common temptation to place enough powerful sensors so that as much as possible about the physical world can be reconstructed. The idea is to give a crisp, complete model that tends to make computers happy. In this tutorial, however, we argue that it is important to start with the particular task and then determine its *information requirements*: What critical pieces of information about the world do we need to maintain, while leaving everything else ambiguous? The idea is to "handle" uncertainty by avoiding big models whenever possible. This is hard to accomplish if we design a general purpose robot with no clear intention in mind; however, most devices appearing in practice have specific, well-defined tasks to perform.

Depending on your background, there might be surprises in this tutorial:

1. **Discrete vs. continuous: Not very important:** Even though computation is discrete and the physical world is usually modeled with continuous spaces, the distinction is not too important in this tutorial. The field of hybrid systems is devoted to the interplay between continuous models, usually expressed with differential equations, and discrete computation models. The point in this tutorial, however, is to study sensor mappings. These may be from continuous to continuous spaces, continuous to discrete, or even discrete to discrete (if the physical world is modeled discretely).

2. **Information spaces, not information theory:** As an elegant and useful mathematical framework for characterizing information transmitted through a noisy channel, information

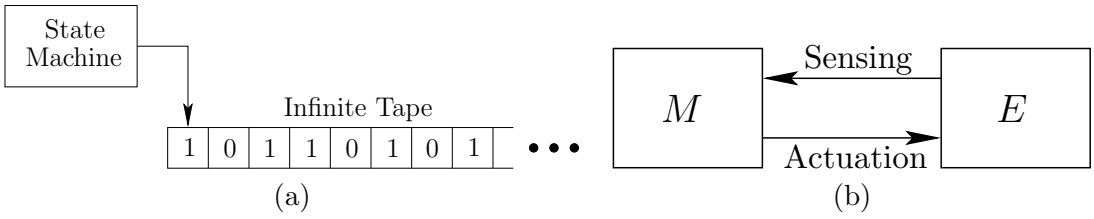


Figure 1: (a) For classical computation, the full state is given by the finite machine state, the head position, and the binary string written on the tape. (b) In this tutorial, there is both an internal computational state and an external physical state.

theory is extremely powerful. The concepts are fundamental to many fields; however, *information spaces* were formulated in the context of game theory and control theory for systems that are unable to determine their state. Thus, this tutorial talks more about how to accomplish tasks in spite of huge amounts of ambiguity in state, rather than measuring information content, using entropy-based constructs. There may indeed be interesting connections between the two subjects, but they are not well understood and are therefore not covered here.

3. **Perfectly accurate and reliable sensors yield huge amounts of uncertainty:** Uncertainty in sensing systems is usually handled by formulating statistical models of disturbance. For example, a global positioning system (GPS) may output location coordinates, but a Guassian noise model might be used to account for the true position. It is important, however, to study the often neglected source of uncertainty due simply to the sensor mapping. Consider the sensor pad at the entrance to a parking garage or drive through restaurant. It provides one bit of information, usually quite reliably and accurately. It performs its task well, in spite of enormous uncertainty about the world: What kind of car drove over it? Where precisely did the car drive? How fast was it going? We are comfortable allowing this uncertainty to remain uncertain. We want to study these situations broadly. This is complementary to the topic of noisy sensors, and both issues can and should be addressed simultaneously. This tutorial, however, focuses mainly on the underrepresented topic of uncertainty that arises from the sensor mapping.

Based on the discussion above, it is clear that sensing and computation are closely intertwined. For robotic devices, *actuation* additionally comes into play. This means that commands are issued by the computer, causing the device to move in the physical world. Therefore, many problems of interest mix all three: sensing, actuation, and computation. Alternative names for sensing are *perception* or even *learning*, but each carries distinct connotations. A broader name for actuation is *control*, which may or not refer to forcing changes in the physical world. Based on this three-way mixture and its increasing relevance, we are forced more than ever to develop new mathematical abstractions and models that reduce complexity and meet performance goals.

Figure 1 shows a conceptual distinction between classical computation and the three-way mixture considered in this tutorial. In Figure 1(a), the Turing machine model is shown, in which a state machine interacts with a boundless binary tape. This and other computation models represent useful, powerful abstractions for ignoring the physical world. Figure 1(b) emphasize the interaction between the physical world and a computer. Imagine discarding the Turing tape and interacting directly with a wild, unknown, chaotic world through sensing and actuation.

A natural questions arises: What is the "state" of this system? In the case of the Turing machine the full state is given by: the finite machine state, head position, and the binary string written

on the tape. For Figure 1(b), this becomes replaced by two kinds of states: internal and external. The internal state corresponds to the state inside of the computation box. Some or all of the internal state will be called an *information state* (or *I-state*), to be defined later. The external state corresponds to the state of the physical world. The internal state is closer to the use of state in computer science, whereas the external state is closer to its use in control theory. The internal vs. external distinction is more important than discrete vs. continuous; either kind of state may be continuous or discrete.

These internal states will be defined to live in an *information space* (or *I-space*), which is where filtering and planning problems naturally live when sensing is involved. In this tutorial, we will define and interpret these spaces in many settings. A continuing mission is to make these spaces as small as possible while being able to efficiently compute over them and to understand their connection to the external states.

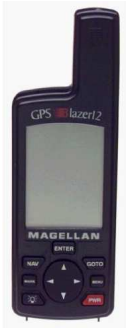
Here are some key themes to take from this tutorial:

- Start from the task and try to *understand* what information is actually *required* to be extracted from the physical world.
- Since sensors leave substantial uncertainty about the physical world, they are best understood as inducing partitions of the external state space into indistinguishable classes of physical states.
- We can design *combinatorial filters* that are structurally similar to Bayesian or Kalman filters, but involve no probabilistic models. These are often dramatically simpler in complexity. They are also perfectly compatible with probabilistic reasoning: Stochastic models can be introduced over them.
- There is no problem defining enormous physical state spaces, provided that we do not directly compute over them. However, state estimation or recovery of a particular state in a giant state space should be avoided if possible.
- Virtual sensor models provide a powerful intermediate abstraction that can be implemented by many alternative physical sensing systems.

The remainder of this tutorial is divided into four main parts:

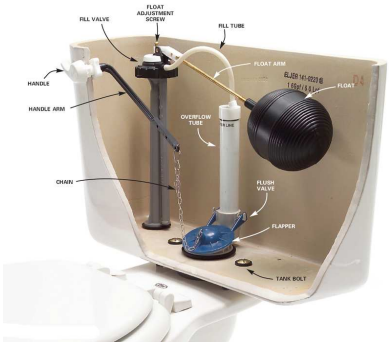
1. **Physical sensors:** Before going into mathematical models, a broad overview of real sensors will be given along with discussions about what we would like to sense.
2. **Virtual sensors:** This part introduces mathematical models of sensors that are abstracted away from the particular physical implementation. Using a definition of the physical state space, a sensor is defined as a mapping from physical states to data that can be measured.
3. **Filtering:** Information accumulates from multiple sensor readings over time or space and needs to be efficiently combined. *Spatial filters* generalize ancient triangulation methods and combine information over space. For *temporal filters*, we find and attempt to "live" in the smallest I-space possible, given the task. The concepts provide a generalization of Kalman and Bayesian filters. The new family includes reduced-complexity filters, called *combinatorial filters*, that avoid physical state estimation.
4. **Planning:** The next step in many applications is to determine whether the world can be manipulated to achieve tasks. In this case, a *plan* specifies actuation primitives (or actions) that are conditioned on the I-states maintained in a filter.

(a) Light-dependent resistor



(b) GPS unit

(c) Wireless card



(d) Toilet float mechanism

Figure 2: Some examples of sensors.

The filtering and planning parts can be distinguished by being *passive* and *active*, respectively. A filtering problem might require making inferences, such as counting the number of people in a building or determining the intent of a set of autonomous vehicles. A planning problem usually disturbs the environment, for example by causing a robot to move a box across the floor.

2 Physical Sensors

2.1 What Is a Sensor?

What is a sensor? Even though we are quick to find examples, it is a difficult question to answer precisely. Consider some devices shown in Figure 2. To consider each 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 2(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 2(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. 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 2(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 [12] that when used as a 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 2(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 we in this tutorial we consider only sensors that provide input to electrical or computer systems.

Based on these examples, it seems best to avoid having a precise definition of a sensor. We will

(a) Shopping mall (b) Control room (c) Assisted living (d) Coral reef

(e) Roomba (f) CMU Boss (g) UAV (h) Protein

Figure 3: Several motivational settings in which we would like to use sensors to monitor or control the environment.

talk about numerous sensors, with the understanding they are just devices that respond to external stimuli and provide signals to a larger system. The next step is to consider the kinds of scenarios in which we will be placing sensors.

2.2 Where Might We Want to Use Sensors?

It is difficult to exhaustively list settings where sensors might be placed. To nevertheless provide some perspective on the kinds places where the concepts from this tutorial may apply, consider the motivating examples shown in Figure 3. Figure 3(a) shows a shopping mall with numerous people move around. Common tasks could be monitoring activities for security or studying consumer habits. Related to this, Figure 3(b) shows a security control room in which video is monitored from numerous sources within the same building. How much can be reconstructed about the movements of people, as they become visible to various cameras? We might want to count people, estimate their flow, or classify them. Now consider a home setting, in which security is a common problem; see Figure 3(c). An increasingly important engineering problem is to monitor activities of people who require assisted living. By keeping track of their movements, changes in their behavior can be detected. Furthermore, if they become trapped, an alarm can be sounded for emergency action. In this setting, people prefer not to be monitored by cameras for privacy reasons. What kind of minimally invasive sensors can be used to accomplish basic monitoring tasks? Figure 3(d) shows similar task, but instead involves monitoring wildlife. Imagine gathering data on air, land, or sea animals for scientific and conservation purposes.

The examples so far have involved passive monitoring, without directly interfering with the environment. Figures 3(e)-(g) show three examples of robotic vehicles that interact with their environment. Sensing is combined with actuation to move vehicles. In the 3(e), a low-cost robot vacuums floors inside of homes.

Figure 3(f) shows the vehicle that won the DARPA Urban Challenge, which involved driving autonomously through a town while taking into account traffic rules and other vehicles. Automated driving is gaining increasing interest for both transportation and military use. We can imagine robots or autonomous vehicles in the sea, on land, in the air, and in space; Figure 3(g) shows an autonomous aerial vehicle (UAV). Other robotic examples include arms that weld in a factory (as in PUMA or ABB robots), mobile robots that arrange inventory in a warehouse (as in Kiva Systems), and humanoids.

Finally, some of the concepts from this tutorial may apply well beyond the scope of the examples here. For example, the problem of measuring protein structure, shown in Figure 3(h), can be viewed as trying to reconstruct as much information possible from limited measurements (which are obtained by sensors, such as mass spectroscopy and NMR).

2.3 What Physical Quantities Are Sensable?

Based on the numerous examples from Section 2.2, it is helpful to group together similar phenomena that can be measured from the surrounding physical world. Consider the following categories of physical quantities:

> **Spatial:** displacement, velocity, acceleration, distance to something, proximity, position, attitude, area, volume, level/tilt, motion detection
>
> **Temporal:** clock, chronometer (elapsed time), frequency.
>
> **Electromagnetic:** voltage, current, power, charge, capacitance, inductance, magnetic field, light intensity, color. These may operate within a circuit or within open space.
>
> **Mechanical:** solid (mass, weight, density, force, strain, torque), fluid (acoustic, pressure, flow, viscosity), thermal (temperature), calories.
>
> **Other:** chemical (composition, pH, humidity, pollution, ozone), radiation (nuclear), biomedical (blood flow, pressure).

Clearly a wide variety of phenomena can be sensed. In Section 2.4, it will be helpful to keep these categories to understand the source of the information they provide.

2.4 What Sensors Are Available?

Dozens of abstract sensors models will soon appear in this tutorial. To emphasize that these are grounded in reality, some widely available sensors are shown in Figure 4. Consider this, in addition to Figure 2, as a market where we can easily obtain real, physical sensors that fulfill the expectations of the mathematical models in Section 3. For each sensor, consider its category from Section 2.3, which indicates the type of phenomenon causing the sensor reading. It is also helpful to imagine sensors as being *simple*, which directly produces an output through transduction (an example is the LDR of Figure 2(a)), or *compound* which may be composed of several simpler sensors and even computational components (an example is the GPS device of Section 2(b)).

The sensors in the top row of Figure 4 cost under $20 US. The *contact sensor* in Figure 4(a) is simply a mechanical switch that forms a circuit when a strong enough force is applied. In combination with a faceplate, this could let a robot know it is hitting a wall. The *sonar* shown in Figure 4(b) emits a high-pitched sound and uses the time that it takes for the sound to rebound from the wall to estimate directional distance. A cheap *compass* (Dinsmore 1490) is shown in Figure 4(c), which indicates 8 possible general directions. A *microphone*, such as the one in Figure 4(d),

(a) Contact sensor (b) Sonar (c) Compass (d) Microphone

(e) Wheel encoder (f) Stopwatch/timer (g) Occupancy detector (h) Safety beam

(i) Camera (j) Wii remote (k) Pressure mat (l) SICK laser scanner

Figure 4: Some examples of widely available sensors, roughly sorted from low-cost to high-cost.

can be used as a sensor in a wide variety of ways, from simple sound detection to sophisticated voice recognition.

Figure 4(e) depicts the inside of a *wheel encoder*, which is used in many applications to count wheel revolutions. By counting the number of light pulses of the LED visible through the disc holes, the total angle can be estimated. Figure 4(f) shows a *stopwatch*, which is just one kind of *clock* or *chronometer* that can be used to estimate time information (either the current time or total elapsed time). The sensors in Figures 4(g) and 4(h) are both based on *infrared light* detection. Figure 4(g) shows an example of a cheap *occupancy detector* (or *motion detector*), and Figure 4(h) shows a *beam sensor*, which is designed to keep a garage door from closing on someone or something.

Figure 4(i) shows a *camera*, which in combination with image processing or computer vision techniques, can perform a wide variety of functions, such as identifying people, tracking motion, analyzing lighting conditions, and so on. Figure 4(j) shows the Wii remote and its sensor bar, which are used in combination by the Nintendo Wii game console to infer hand motions and positions. A cheap camera tracks LEDs on the sensor bar to estimate position and orientation, and accelerometers in the remote estimate velocities. Figure 4(k) shows a pressure mat that sends a signal to open the door when someone steps on it. For all of the sensors shown so far, there are

versions available for under $50 US (some as low as $5). In many settings, though, expensive sensors may be used to provide more complete information. The GPS device of Figure 2(b) is more complex and more expensive than the sensors shown so far in Figure 4. As a final example, however, consider the SICK *laser rangefinder*, which costs around $5000 and provides distance measurements at every half-degree over 180 degrees, with accuracy around one centimeter. Furthermore, a complete scan can be performed in about 1/30 of a second. This sensor has been extremely popular over the last decade in mobile robotics for building indoor maps and localizing the robot.

Many other sensors are possible, such as mechanical scales to measure weight, gyroscopes to measure orientation, thermometers to measure temperature, radiation detectors, carbon monoxide detectors, smoke alarms, and so on.

2.5 Common Sensor Characteristics

Most sensors are characterized in terms of a *transfer function*, which relates the possible inputs (phenomena) to the outputs (sensor readings). In Section 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.

Several important terms and concepts will be introduced with respect to the transfer function. For simplicity here, suppose that the transfer function is a mapping $g : \mathbb{R} \to \mathbb{R}$, 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, study the preimages of sensor mappings in Section 3.3. This leads to the fundamental source of uncertainty covered in this tutorial. More uncertainty may arise, however, due to lack of *repeatability*. If the sensor used under the exact conditions multiple times, does it always produce the same reading?

An important process for most sensors is *calibration*. In this case, systematic (or repeatable) errors can be eliminated to improve sensor accuracy. For example, suppose we have purchased a cheap digital thermometer that has good repeatability but is usually inaccurate by several degrees. We can used 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.

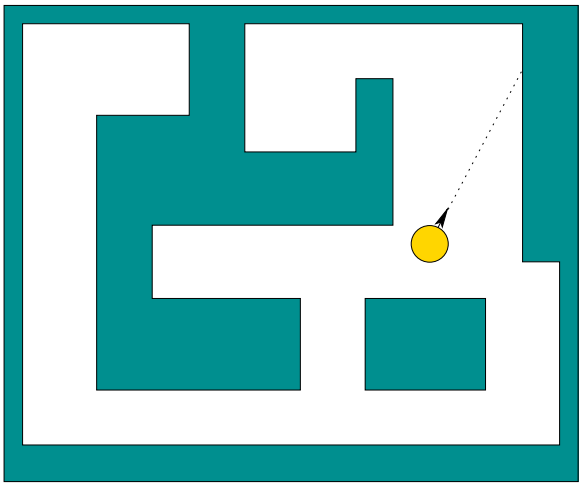


Figure 5: A mobile robot is placed in an indoor environment with polygonal walls. It measures the distance to the wall in the direction it is facing.

3 Virtual Sensors

Now that physical sensors have been described, we turn to making mathematical models of the information that is obtained from them. This leads to *virtual sensors* that could have many alternative physical implementations. The key idea in this section is to understand how two spaces are related:

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

2. The *observation space*, which is the set of possible sensor output values or observations.

We will define a *sensor mapping*, which indicates what the sensor is supposed to observe, given the cartoon-like description of the external world. For most sensors, a tremendous amount of uncertainty arises because the sensor does not observe *everything* about the external world. Understanding how to model and manipulate this uncertainty is the main goal of this section. Additional uncertainty may arise due to sensor noise or calibration errors, but it is important to consider these separately. Eventually, all sources of uncertainty combine, making it difficult or impossible to reason about them without understanding them independently.

3.1 Physical State Spaces

Consider the scenario shown in Figure 5, 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 sonar or laser 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. Do we already know the robot's configuration (position and orientation)? Do we have a precise geometric map of all of the walls? If we know both of these already, then we would learn absolutely nothing from the sensor observation. If we know the robot's configuration but do not have a map, then the sensor reading provides information about how the walls are arranged. Alternatively, if we have a map but not the configuration, then we learn something about the robot's position and orientation. If we have neither, then something is

learned about *both* the configuration and the map of walls. The purpose of defining the physical state space is to characterize the set of *possible* external worlds that are consistent with a sensor observation and whatever background information is given.

Since the physical state contains both configuration and map information, a common structure frequently appears for the physical state 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 state space would be the set of all (z, Z) such that $z \in Z$ and $Z \in \mathcal{Z}$.

3.1.1 A mobile robot among obstacles

No walls Return to Figure 5. If there were no walls, then the robot could move to any position $(q_x, q_y) \in \mathbb{R}^2$ and orientation $q_\theta \in [0, 2\pi)$. The physical state, denoted by x, is completely expressed by $x = (q_x, q_y, q_\theta)$. The physical state space, denoted by X, in this case is the set of all robot positions and orientations. We can imagine $X \subset \mathbb{R}^3$ by noting that $q_y, q_y \in \mathbb{R}$ and $x_\theta \in [0, 2\pi) \subset \mathbb{R}$. This will be perfectly fine for defining a sensor; however, we sometimes need to capture additional structure. Here, the fact that 0 and 2π are the same orientation has not been taken into account. Formally, we can place the robot at the origin, facing the x axis, and apply homogeneous transformation matrices to translate and rotate it [13, 14]. The set of all such transformations is called a *matrix group*. In particular, we obtain $X = SE(2)$, which is the set of all 3 by 3 matrices that can translate and rotate the robot. As is common in motion planning, we could alternatively write $X = \mathbb{R}^2 \times S^1$, in which S^1 denotes a circle in the topological sense and represents the set of possible orientations. Let $S^1 = [0, 2\pi]$ with a declaration that 0 and 2π are the same. Most of these issues are quite familiar in robotics, control theory, and classical mechanics; see [1, 3, 16, 17]. The discussion in this tutorial will be kept simple to avoid technicalities that are mostly orthogonal to our subject of interest.

Known map Now suppose that the robot has a perfect polygonal map of its environment. This constrains the robot position (q_x, q_y) to lie in some set $E \subset \mathbb{R}^2$ that has a polygonal boundary. The state space becomes $X = E \times S^1$, in which S^1 once again accounts all possible orientations.

One of several maps The robot is now told that one of k possible maps is the true one. For example, we may have a set $\mathcal{E}$ of five possible maps $\{E_1, E_2, E_3, E_4, E_5\}$. This can be imagined as having $k = 5$ copies of the previous state space. The state space X is the set of all pairs (q, E_i) in which $(q_x, q_y) \in E_i$ and $E_i \in \mathcal{E}$.

Unknown map If the map is completely unknown, then the robot may be told only the *map family*, which is an infinite collection. For example, $\mathcal{E}$ may be the set of all polygonal subsets[1] of $\mathbb{R}^2$. Every map can be specified by a polygon and describes a subset of $\mathbb{R}^2$. The state space X is the set of all pairs (q, E) in which $(q_x, q_y) \in E$ and $E \in \mathcal{E}$. Note that we can write $X \subset SE(2) \times \mathcal{E}$.

Numerous other map families can be made. Here are several thought-provoking possibilities, in which each defines $\mathcal{E}$ as a set of subsets of $\mathbb{R}^2$. Thus, $\mathcal{E}$ could be:

- The set of all connected, bounded polygonal subsets that have no interior holes (formally, they are *simply connected*).

[1] To be more precise, each subset must be closed, bounded, and simply connected.

- The previous set expanded to include all cases in which the polygonal region has a finite number of polygonal holes.
- All subsets of $\mathbb{R}^2$ that have a finite number of points removed.
- All subsets of $\mathbb{R}^2$ that can be obtained by removing a finite collection of nonoverlapping discs.
- All subsets of $\mathbb{R}^2$ obtained by removing a finite collection of nonoverlapping convex sets.
- A collection of piecewise-analytic subsets of $\mathbb{R}^2$.

Each map does not even have to contain homogeneous components. For example, each could be described as a polygonal region that has a finite number of interior points removed. Furthermore, three-dimensional versions exist for the families above. For example, $\mathcal{E}$ could be a set of polyhedral regions in $\mathbb{R}^3$.

In some of the examples above, obstacles such as points or discs are removed. We could imagine having an *augmented* map in which a label is associated with each obstacle. For example, if n disks are removed, then they may be numbered from 1 to n. This becomes a special case of the models considered next.

3.1.2 A bunch of bodies

Now consider placing other kinds of entities into an environment E, which may or may not contain 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 E and can be transformed using its own configuration parameters. For example, a body could be a point that is transformed by (q_x, q_y) parameters or a rectangle that is transformed by (q_x, q_y, q_θ) parameters. We can write $B(q_x, q_y, q_\theta) \subset E$ to indicate the set of points occupied by B when at configuration (q_x, q_y, q_θ). In general, bodies could be as complex as any robots considered in robot motion planning; however, this is too much of a digression for the tutorial; see [13, 14] for understanding how the configuration space of bodies is constrained when they are not points. Here, it will be assumed that all bodies are points, except for obstacles.

In this tutorial, bodies may have many different interpretations and uses. Here are terms and examples that appear all over the literature:

- **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.
- **Object:** A body that can be detected and manipulated by a robot. It can *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 stationary body that has an unknown location but is easy to recognize by a sensor directly over it.
- **Tower:** A body that transmits a signal, such as a cell-phone tower or a lighthouse.

Rather than worry about particular names of bodies, which are clearly arbitrary, it is more important to think about their mathematical characteristics. Think about these three important properties of a 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. Its configuration could nevertheless be unknown. 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, ..., B_n that are distinguishable simply by the fact that each is uniquely defined. We can now define any equivalence relation $\sim$ and say $B_i \sim B_j$ 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 models are possible, but are not considered here. (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.)

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 remainder of the tutorial, many different kinds of bodies will appear and it is crucial to pay attention to their properties rather than their particular names. In all cases, it will be assumed that bodies are contained in E.

3.1.3 Fields

A *field* is a function $f : \mathbb{R}^n \to \mathbb{R}^m$, in which n is the dimension of the environment ($n = 2$ or $n = 3$) and m could be any finite dimension. Usually, $m \leq n$.

As a first example, a map in Section 3.1.1 could equivalently be expressed as a function $f : \mathbb{R}^2 \to \{0, 1\}$ in which $f(q_x, q_y) = 1$ if and only if $(q_y, q_y) \in E$. This causes a clear division of $\mathbb{R}^2$ into an obstacle region and a collision-free region; however, it is sometimes useful to assign intermediate

values. Let the map be defined as $f : \mathbb{R}^2 \rightarrow [0, \infty)$ in which $f(q_x, q_y)$ yields an *altitude*. For an outdoor setting, f could describe a terrain map. In this case, $\mathcal{E}$ is a set of functions in which each $f \in \mathcal{E}$ satisfies some properties, such as a bound on the maximum slope. If there are no other obstacles, then the state space would be $X = SE(2) \times \mathcal{E}$.

Perhaps the most important example is the electromagnetic field generated by a radio transmitter. In a 2D environment, this is captured by a vector field $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$. Thus, a 2D vector is produced at every point in $\mathbb{R}^2$. A simplified version could be defined as an *intensity field*, $f : \mathbb{R}^2 \rightarrow [0, \infty)$, in which the scalar values represent the signal intensity (the magnitudes of the original vectors).

Fields can also be defined with the consideration of other obstacles. For example, waves may propagate through a world that is constrained to a polygonal region.

3.1.4 Introducing Time

Of course the world is not static. If the physical state space X is meant to be a cartoon-like description of the world, it only represents a single snapshot. Time will now be introduced to animate the world. Let T refer to an interval of time, in which the most convenient case is $T = [0, \infty)$. Starting from any physical state space X as defined above, we can obtain a *state-time space* $Z = X \times T$, in which each $z \in Z$ is a pair $z = (x, t)$ and x is the state at time t.

Since time always marches forward, we can consider the "animation" as a path through Z that is parameterized by time. This leads to a *state trajectory*, $\tilde{x} : T \rightarrow X$. The value $\tilde{x}(t) \in X$ represents the state at time t. The value $\tilde{x}(0)$ is called the *initial state*.

The configurations of bodies may change over time, but are continuous functions. In fact, they are usually differentiable, leading to time derivatives. For example $\dot{q} = dq/dt$ is *velocity* and $\ddot{q} = d^2q/dt^2$ is *acceleration*. Such quantities can be incorporated directly into the state to expand X into a *phase space* as considered in mechanics. For example, $x = (q, \dot{q})$ is the phase of a mechanical system in Lagrangian mechanics. However, the rest of this tutorial will avoid working directly with velocities.

Before time was introduced, $\mathcal{E}$ was introduced to represent possible maps. Now it is possible that the maps vary over time, along with configurations. This variation may or may not be predictable.

3.2 Virtual Sensor Models

Now that the state space X is defined, we can introduce numerous sensor models that are inspired by the physical sensors in Section 2.4, but are expressed abstractly in terms of X.

3.2.1 The Sensor Mapping

We define models of instantaneous sensors, which use the physical state to immediately produce an observation. Let X be any physical state space. Let Y denote the *observation space*, which is the set of all possible sensor observations. A virtual sensor is defined by a function

$$h : X \rightarrow Y, \tag{1}$$

called the *sensor mapping*, which is very much like the transfer function described in Section 2.5. The interpretation is that when $x \in X$, the sensor instantaneously observes $y = h(x) \in Y$. Equation 1 is perhaps the most important definition in this tutorial. Numerous virtual sensor models will now be defined in terms of it. These models can be physically implemented in several alternative ways using various sensors. If (1) seems too idealistic, considering that sensors may be unpredictable, do not worry. Sensor disturbances and other complications are handled in Section 3.5.

3.2.2 Basic Examples

Models 1, 2 and 3 will be useful for comparisons to other, more practical models.

Model 1 (Dummy Sensor)
At one extreme, a worthless sensor can be made by letting $Y = \{0\}$ with $h(x) = 0$ for all $x \in X$. This sensor never changes its output, thus providing no information about the external world. ■

Model 2 (Identity Sensor)
At the other extreme, we can define an "all knowing" sensor by setting $Y = X$ and letting $y = h(x) = x$. From a single observation, no uncertainty about the external world exists. ■

Model 3 (Bijective Sensor)
Let h be any bijective function from X to Y. By one interpretation, this sensor is as powerful as Model 2 because x can be reconstructed from y using the inverse $x = h^{-1}(y)$. In practice, however, it may be costly or impossible to compute the inverse of h. ■

The next two models are generic but useful in many settings.

Model 4 (Linear Sensor)
For a model that is in between the power of Models 1 and 2, suppose $X = Y = \mathbb{R}^3$. Let $y = h(x) = Cx$ for some 3 by 3 real-valued matrix C. In this case, x can be reconstructed from y if C has full rank. This is a special case of Model 3. More generally, if C has rank $k \in \{1, 2, 3\}$, then there is a $(3 - k)$-dimensional linear subspace of X that produces the same observation y. Linear sensors can be similarly defined for any $X = \mathbb{R}^n$ and $Y = \mathbb{R}^m$. In fact, this is the standard *output model* for linear systems in control theory [4]. ■

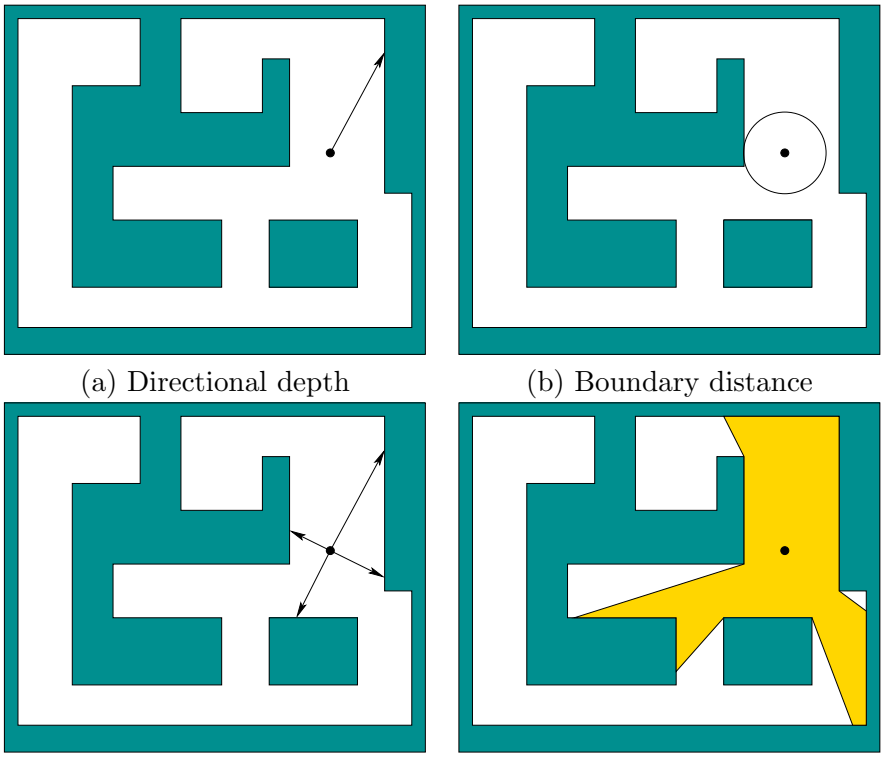
Model 5 (Projection Sensor)
This convenient sensor directly observes some components of X. For example, if $x = (x_1, x_2, x_3) \in \mathbb{R}^3$, then a projection sensor could yield the first two coordinates. In this case, we have $Y = \mathbb{R}^2$ and $y = h(x) = (x_1, x_2)$. ■

3.2.3 Depth Sensors

We now introduce an important family of sensor models that arise in mobile robotics. Using the state space models from Section 3.1.1, *depth sensors* base the observation on distance from the sensor to the boundary of E. The state space is $X \subset SE(2) \times \mathcal{E}$, in which each state $x \in X$ is represented as $x = (q_x, q_y, q_\theta, E)$ with $(q_x, q_y) \in E$ and $E \in \mathcal{E}$. For convenience, the notation $p = (q_x, q_y)$ and $\theta = q_\theta$ will be used.

Model 6 (Directional Depth Sensor)
How far away is the wall in the direction the robot is facing? Figure 6(a) shows a mobile robot facing a direction to the upper right. Let $b(x)$ denote the point on the boundary of E that is struck by a ray emanating from p and extended in the direction of θ. The sensor mapping

$$h_d(p, \theta, E) = \|p - b(x)\| \tag{2}$$



(a) Directional depth (b) Boundary distance

(c) K-directional depth (d) Omnidirectional depth

Figure 6: Several variations exist for depth sensors.

precisely yields the distance to the wall. This could be implemented using a sonar, shown in Figure 4(b), or a single laser/camera combination. ■

Model 7 (Boundary Distance Sensor)
How far away is the nearest wall, regardless of direction? As shown in Figure 6(b), this can be considered as the radius of the largest disk that can be placed in E, centered on the robot. The sensor mapping can be expressed in terms of h_d:

$$h_{bd}(p, \theta, E) = \min_{\theta' \in [0, 2\pi)} h_d(p, \theta', E) \tag{3}$$

Note that h_{bd} ignores θ, as expected. This sensor could be implemented expensively by using two SICK laser scanners (shown in Figure 4(l)) and reporting the minimum distance value. A cruder version could be made from an array of sonars. ■

Model 8 (Proximity Sensor)
Imagine that a light goes on when the robot is within a certain distance, $\epsilon > 0$, to the wall. This is easily modeled as

$$h_{p\epsilon}(p, \theta, E) = \begin{cases} 1 & \text{if } h_{bd}(p, \theta, E) \leq \epsilon \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

An array of simple infrared sensors could accomplish this. A directional version could alternatively be made by using h_d from (2) instead of h_{bd} above. ■

Model 9 (Boundary Sensor)
By reducing ϵ to 0, we obtain a sensor that indicates whether the robot is touching the boundary. This is called a *boundary* or *contact sensor*:

$$h_{bd}(p, \theta, E) = \begin{cases} 1 & \text{if } h_{bd}(p, \theta, E) = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Note that $h_{bd}(p, \theta, E) = h_{p0}(p, \theta, E)$. Again, a directional version can be made by substituting h_d for h_{bd}. This sensor could be implemented using contact sensors, as shown in Figure 4(a). ■

Model 10 (Shifted Directional Depth Sensor)
This model is convenient for defining the next two. It is simply a directional sensor that allows an offset angle ϕ between the direction that the robot faces and the direction that the sensor is pointing:

$$h_{sd\phi}(p, \theta, E) = \|p - b(p, \theta + \phi, E)\|. \tag{6}$$

In comparison to h_d in (2), only ϕ has been inserted. ■

Model 11 (K-Directional Depth Sensor)
Suppose there is a set of offset angles $\phi_1, \ldots, \phi_k$, which in most cases are regularly spaced. Figure 6(c) shows an example for which $k = 4$ and the directions are spaced at right angles. In this case, the observation is a vector $y = (y_1, \ldots, y_k)$ in which

$$y_i = h_i(p, \theta, E) = h_{sd\phi_i}(p, \theta, E). \tag{7}$$

■

Model 12 (Omnidirectional Depth Sensor)
In the limiting case, imagine letting k become infinite so that measurements are taken in all directions, as shown in Figure 6(d). In this case, the observation is an entire function (imagine an infinite-dimensional vector). We obtain $h_{od}(x) = y$, in which $y : S^1 \rightarrow [0, \infty)$ and

$$y(\phi) = h_{od\phi}(p, \theta, E). \tag{8}$$

This means that evaluating the function y at $\phi \in [0, 2\pi)$ yields the shifted directional distance $h_{od\phi}(p, \theta, E)$; see Figure 7. In practice, most sensors have a limited range of directions. In this case the domain of y can be restricted from S^1 to $[\phi_{min}, \phi_{max}]$ to obtain observations of the form $y : [\phi_{min}, \phi_{max}] \rightarrow [0, \infty)$. In practice, this corresponds closely to the dense measurements obtained from the SICK laser scanner, shown in Figure 4(l). That one scans over 180 degrees; however, 360-degree variants exist. ■

For all of the sensor models from Model 6 to 12, an important *depth-limited* variant can be made. When placed into large enough environments, a sensor might not be able to detect a wall that is too far away. Instead of a distance range $[0, \infty)$, we could have a range of distances from d_{min} to d_{max}. The following model illustrates the idea.

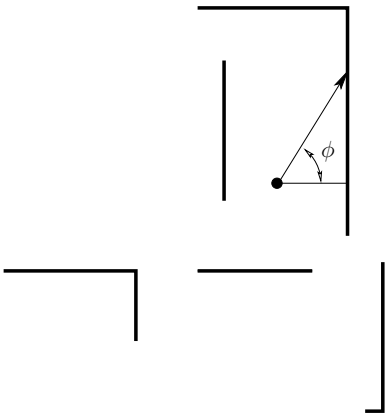


Figure 7: For the omnidirectional depth sensor, Model 12 a function $y : S^1 \rightarrow [0, \infty)$ is obtain in which each $y(\phi)$ is the depth in the direction $\theta + \phi$. The figure shows how the depth data appears for the environment in Figure 6(a) and $\theta = 0$.

Model 13 (Depth-Limited Directional Depth Sensor)
Model 6 can be modified to obtain a depth-limited version in which the sensor cannot give measurements when the distance is outside of the interval $[d_{min}, d_{max}]$ for some $d_{min}, d_{max} \geq 0$ with $d_{min} < d_{max}$. Let $d(x) = p - b(x)$. The sensor mapping is

$$h_{dd}(p, \theta, E) = \begin{cases} d(x) & \text{if } d_{min} \leq d(x) \leq d_{max} \\ \# & \text{otherwise,} \end{cases} \tag{9}$$

in which the symbol # indicates that the sensor cannot determine the distance. If the wall is too far away, most sensors will not report a value. For example, a sonar echo will not be heard. Thus, this model is realistic in many settings. ■

Section 3.2.4 covers many depth-limited sensors, but with a different purpose in mind. Rather than measuring depth, they are designed to detect bodies within their field of view. Depth-limited sensors also become important in Section 3.2.6, for defining gap sensors.

3.2.4 Detection Sensors

As the name suggests, this family models sensors that detect whether one of more bodies are within their sensing range. Physical examples include a camera, the occupancy detector of Figure 4(e), and the pressure mat of Figure 4(k).

Three fundamental aspects become important in detection sensor models:

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?

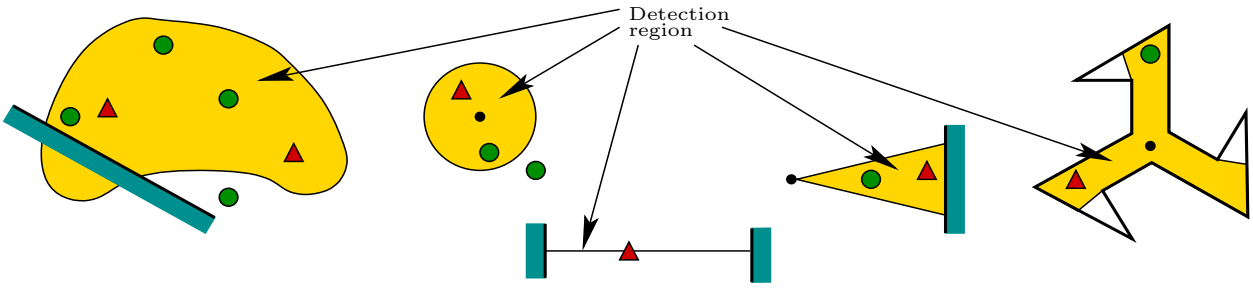


Figure 8: The detection regions may take many different shapes and may or may not be attached to a movable body.

If the answer is "no" to all three questions, then the simplest case is obtained: A stationary detection sensor that indicates whether at least one point body is within its range. For this case, let $V \subset E$ be called the *detection region*. Suppose that E contains one ore more point bodies that can move around. Note that V can be any shape, as shown in Figure 8.

We now present several models, starting with the simplest case and eventually taking into account all three complications above.

Model 14 (Static Binary Detector)
A simple detection model can now be defined in terms of V. Suppose that a single body moves in E and its position is denoted by p. The sensor mapping is

$$h(p, E) = \begin{cases} 1 & \text{if } p \in V \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

It simply indicates whether the body is in the detection region. Physically, this could correspond to a cheap occupancy sensor that is mounted on the wall. ■

There are three separate axes along which to generalize (10). Each will be handled separately, but all three generalizations can clearly be combined.

Model 15 (Moving Binary Detector)
Suppose the sensor can move, as in a camera that is mounted on a mobile robot. Let q denote the configuration of the body that is carrying the sensor. We now obtain $V(q) \subset E$ as the configuration-dependent detection region. The sensor mapping is

$$h(p, E) = \begin{cases} 1 & \text{if } p \in V(q) \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

■

Model 16 (Detecting Larger Bodies)
What if the body has some shape and is transformed by q' to obtain $B(q') \subset E$? Then we could, for example, make a static binary detector for general bodies:

$$h(q', E) = \begin{cases} 1 & \text{if } B(q') \cap V \neq \emptyset \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

The sensor detects the body if any part of it enters V. This is similar to the definition of configuration-space obstacle region, $\mathcal{C}_{obs}$, in motion planning [5, 13, 14]. An alternative definition would require the body to be contained in the detection region: $B(q') \subseteq V$. If the sensor can additionally move, then V in (12) is replaced with $V(q)$ and the state becomes $x = (q, q', E)$. ■

Now suppose there are multiple bodies. Let $P = \{p_1, \ldots, p_n\}$ denote a set of n point bodies that move in E. The state becomes $x = (q, p_1, \ldots, p_n, E)$ in which q is the sensor configuration.

Model 17 (At-Least-One-Body Detector)
This model detects whether there is at least one body in the detection region $V(q)$. The sensor mapping is

$$h(q, p_1, \ldots, p_n, E) = \begin{cases} 1 & \text{if for any } i,\ p_i \in V(q) \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

■

Model 18 (Body Counter)
Moving away from a binary sensor, the sensor could count the number of bodies in the detection region $V(q)$. The sensor mapping is

$$h(q, p_1, \ldots, p_n, E) = |P \cap V(q)|, \tag{14}$$

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

More generally, we can consider bodies that are partially distinguishable to the sensor. Let L be a set of class labels, attribute values, or feature values that can be assigned to bodies, as discussed in Section 3.1.2. Let ℓ be an assignment mapping $\ell : \{1, \ldots, n\} \rightarrow L$.

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

$$h_\lambda(p, E) = \begin{cases} 1 & \text{if for some } i,\ p_i \in V \text{ and } \ell(i) = \lambda \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

In a physical implementation, a camera could be used with computer vision techniques to classify and label bodies in the image. ■

Numerous other extensions and variations are possible. Here are some ideas: 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 components, the sensor might indicate which component was crossed, and 5) multiple detection sensors could be in use, each of which classifying bodies differently.

3.2.5 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*, a term introduced by Guibas [8]. A detection sensor only tells us which bodies are in view, whereas a relational sensor additionally indicates how they are arranged.

Let R be any relation on the set of all bodies. For a pair of bodies, B_1 and B_2, examples of $R(B_1, B_2)$ 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 full state: The configurations of the sensor and the bodies. We therefore write the relation as R_x and define it over the set $\{1, \ldots, n\}$, which includes the indices of the bodies. Using this notation for the "in front of" example, $R_x(i, j)$ means that body B_i is in front of B_j when viewed from the state $x = (q_s, q_1, \ldots, q_n)$, in which q_s is the sensor configuration and each remaining q_i is the ith body configuration.

Model 20 (Primitive Relational Sensor)
This sensor indicates whether the relation R_x is satisfied for two bodies B_i and B_j that are in the detection region:

$$h(x) = \begin{cases} 1 & \text{if } R_x(i, j) \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

■

Numerous instantiations of Model 20 can be used in combination to obtain *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_x for which the vertices are the set of bodies and a directed edge exists if and only if $R_x(i, j)$. As the state changes, the edges in G_x may change.

An important compound relational sensor will now be defined.

Model 21 (Linear Permutation Sensor)
Suppose there is a finite set of static point bodies in the plane that are considered as completely distinguishable landmarks. Consider a relation $\preceq_l$, for which $a \preceq_l b$ means that a appears to be to the left of b when viewed from the sensor position (q_x, q_y). 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 9(a). The binary observations that determine $\preceq_l$ can be combined to yield a single observation that is a linear ordering of the landmarks. In the example, the observation would be $y = (4, 2, 1, 3, 5)$. If the landmarks were capable of moving, then any permutation might be possible, and Y would be the set of all 5! permutations. ■

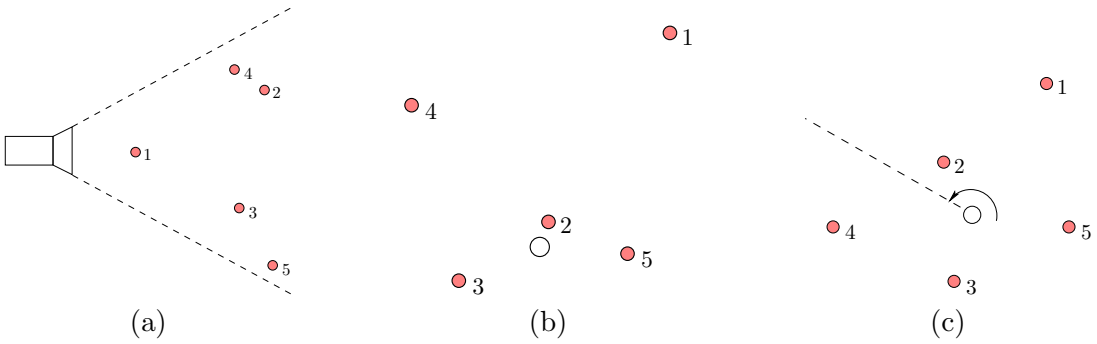


Figure 9: Three kinds of compound relational sensors: (a) The linear sensor observes that the landmarks are ordered from left to right as $(4, 2, 1, 3, 5)$. (b) This sensor sorts the landmarks closest to farthest, resulting in the observation $(2, 3, 5, 4, 1)$. (c) The cyclic sensor sweeps counterclockwise and yields the cyclic ordering $(1, 2, 4, 3, 5)$

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 a is to the left or right of b because they are perfectly aligned in the sensor view. Such cases can be handled by defining multiple relations. For example, one primitive could be $\preceq_l$, and an new one $\preceq_a$ could indicate whether they are aligned.

Model 22 (Distance Permutation Sensor)
Figure 9(b) shows how to obtain an alternative permutation based on sorting the bodies from nearest to farthest. In practice, imagine that each landmark 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 point p_k is to the left or right of a ray that starts at point p_i and pierces point p_j. This relation involves triples of points, and can be expressed as $R_x(i, j, k)$. This relation can be used to define the next model.

Model 23 (Cyclic Permutation Sensor)
We extend Model 21 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, a, b, and c, we can determine whether the cyclic permutation is (a, b, c) or (a, c, b) (note that others are equivalent, such as $(b, c, a) = (a, b, c)$). When the primitive observations are combined, the compound sensor in this case yields a cyclic permutation of the landmarks, as shown in Figure 9(c). ■

If the bodies are only partially distinguishable, then many interesting relational sensor variants arise.

3.2.6 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 $V(q)$, which is denoted as $\partial V(q)$. For most 2D

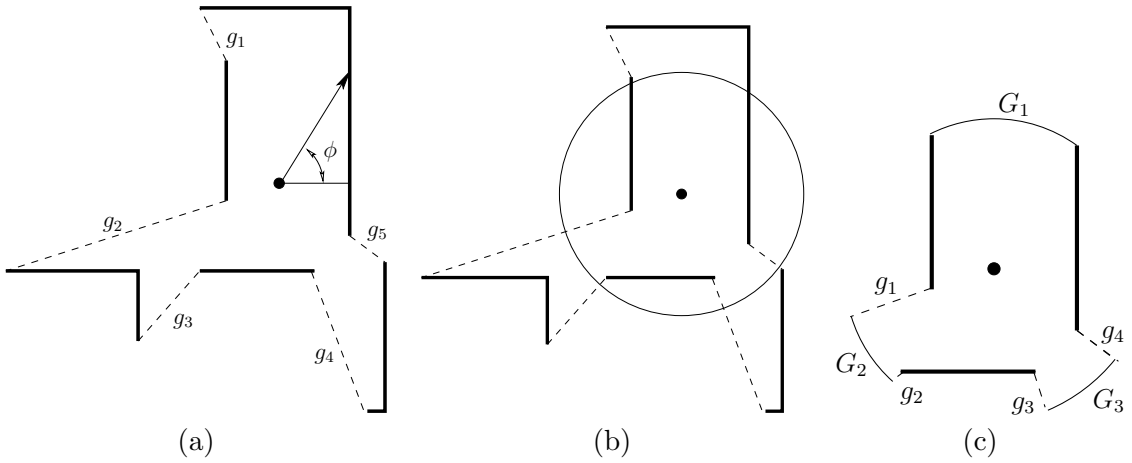


Figure 10: Gap sensor models: (a) Five discontinuities in depth are observed. (b) A limited range is considered. (c) Two kinds of gaps are obtained for limited range.

cases, $\partial V(q)$ is a closed curve. To motivate this model, recall Model 12, Figure 6(d), and Figure 7. The data from the omnidirectional depth sensor are depicted again in Figure 10(a), but this time discontinuities or *gaps* in the depth measurements are shown. When sweeping counter-clockwise, imagine a sensor that reports: A wall, then a gap g_1, then a wall, then a gap g_2, then a wall, and so on. 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 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 3.2.3, the complement of E can be treated as a static body, so that the observation alternates between gaps and the environment boundary.

Model 24 (Simple Gap Sensor)
This sensor has already been described using Figure 10(a). Suppose that a robot carries a sensor with an omnidirectional field of view and is placed into a nondegenerate environment E that bounded by a simply polygon and contains no interior obstacles. Treating the complement of E as a special body, say B_0, the gap sensor for Figure 10(a) observes

$$y = (B_0, g_1, B_0, g_2, B_0, g_3, B_0, g_4, B_0, g_5), \tag{17}$$

which is interpreted as a cyclic sequence. Since it is impossible to have two consecutive gaps, the B_0 components contain no information, and (17) can be simplified to $y = (g_1, g_2, g_3, g_4, g_5)$. Once again, this observation is cyclic; for example, $y = (g_3, g_4, g_5, g_1, g_2)$ is equivalent. ■

Model 25 (Depth-Limited Gap Sensor[2])
In reality, most sensors have limited range. Suppose that for an omnidirectional sensor, nothing can be sensed beyond some fixed distance, as shown in Figure 10(b). The resulting data from a depth sensor would appear as in Figure 10(c). There are two kinds of gaps: one 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

[2]This model is based on the one introduced in [15].

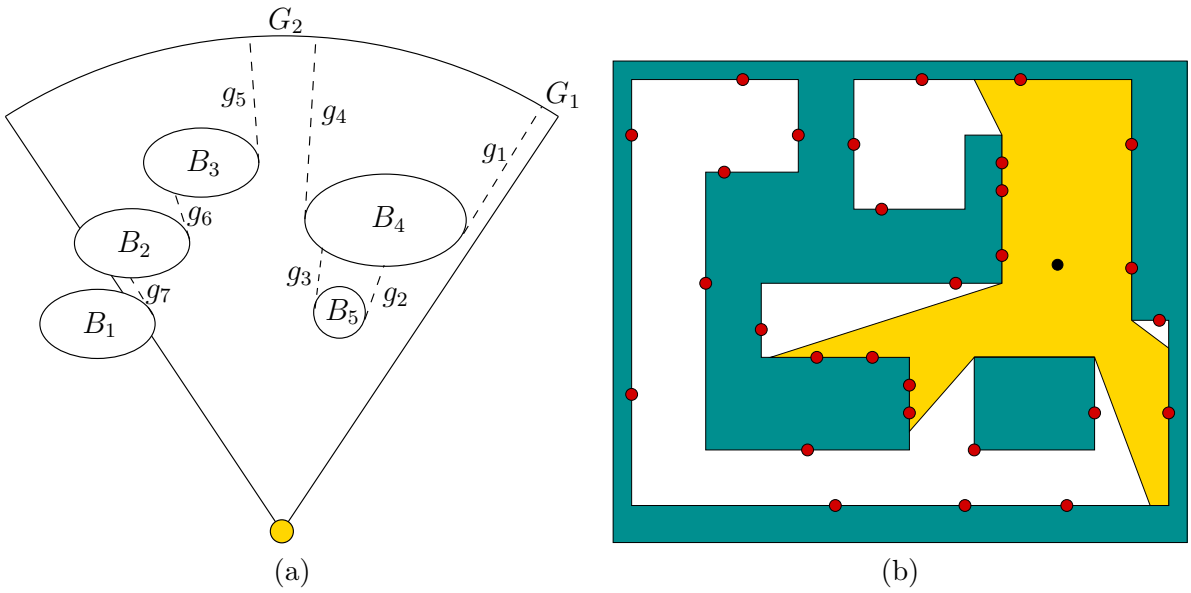


Figure 11: (a) A gap sensor among multiple bodies. (b) A sensor that counts landmarks between gaps.

G_i. The observation for the example in Figure 10(c) is $y = (B_0, G_1, B_0, g_1, G_2, g_2, B_0, g_3, G_3, g_4)$, which again is a cyclic sequence. In contrast to Model 24, the appearances of B_0 cannot be deleted without losing information. ■

Model 26 (Multibody Gap Sensor)
In the models so far, only one body, B_0, was considered. Now suppose there are multiple bodies, as shown in Figure 11(a). The sensor sweeps from right to left, and is not omnidirectional. In this case, the observation is a linear sequence,

$$y = (G_1, g_1, B_4, g_2, B_5, g_3, B_4, g_4, G_2, g_5, B_3, g_6, B_2, g_7, B_1). \tag{18}$$

■

For Model 26, it was assumed that the bodies are completely distinguishable. As in Model 19, it is once again possible assign labels to be bodies. In this case, Model 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 [7].

Model 27 (Landmark Counter)
Let E be a bounded environment 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 E. All bodies are assigned a common label, such as "feature", meaning that they are completely indistinguishable. When in the interior of E, 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 11(b) is $y = (3, 3, 4, 0, 1)$.

The model can be adapted 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, and 4) the bodies could be capable of motion. ■

3.2.7 Field Sensors

Recall from Section 3.1.3 that vector fields can be defined in the world. For the next models, suppose that the world is two-dimensional and a field $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is known. Furthermore, the particular E is known and is simply $E = \mathbb{R}^2$. Extensions that remove obstacles from E are straightforward. The state space here is simply $X = SE(2)$, which is parameterized as $x = (p, \theta)$.

Model 28 (Direct Field Sensor)
This sensor observes the field vector. The sensor mapping is

$$h(x) = h(p, \theta) = (f_1(p), f_2(p)), \tag{19}$$

which yields a two-dimensional observation vector. ■

Model 29 (Direct Intensity Sensor)
This sensor provides the magnitude of the field vector. For radio signals, this could be achieved using a non-directional signal meter. The sensor mapping is

$$h(x) = h(p, \theta) = \|f(p)\|, \tag{20}$$

which yields a nonnegative real *intensity* value. ■

Model 30 (Intensity Alarm)
In the spirit of previous sensor models in the section, a binary sensor can be made that indicates when the intensity is above a certain threshold $\epsilon \geq 0$. The sensor mapping is

$$h(p, \theta) = \begin{cases} 1 & \text{if } \|f(p)\| \geq \epsilon \\ 0 & \text{otherwise.} \end{cases} \tag{21}$$

■

Model 31 (Transformed Intensity)
In most settings, it is unreasonable to expect to recover the precise magnitude. We might nevertheless have a sensor that returns higher values as the intensity increases. Let $g : [0, \infty) \rightarrow [0, \infty)$ be any strictly monotonically increasing smooth function. The sensor mapping is

$$h(x) = g(\|f(x)\|). \tag{22}$$

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 state space and g becomes part of the state (in other words, $x = (p, \theta, g)$). Such a sensor can still provide useful information. For example, if $y = h(x)$ is increasing over time, then we might know that we are closer to the radio transmitter, even though g is unknown. ■

Model 32 (Field Vector Observation)
This sensor directly measures the entire field vector $f(p)$; however, the vector is rotated based on the orientation θ. For example, if the field vector "points" in the direction $3\pi/4$ and $\theta = \pi/4$, then the sensor observes the vector as pointing at $3\pi/4 - \theta = \pi/2$. Let $R(\phi)$ be a 2×2 rotation matrix that induces a rotation by ϕ. The rotated vector observation is

$$h_{fv}(x) = R(-\theta)f(p). \tag{23}$$

If f is given and θ is unknown, then it can be determined using $h_{fv}(x)$. Likewise, if θ is known and f is unknown, then $f(p)$ can be determined from $f(p) = R(\theta)h_{fv}(x)$. ■

Now consider constructing a magnetic compass. If the field is known, as in the case of the earth's magnetic field, then we can look at the direction of the vector observed using Model 32 and infer the direction θ. The direction with respect to an arbitrary given field is given in the next model.

Model 33 (Field Direction Observation)
The direction obtained by the observation vector (24). Let $y' = h_{fv}(x)$. The sensor mapping is

$$y = h_{fdo}(x) = \text{atan2}(y'_2, y'_1), \tag{24}$$

in which $y \in [0, 2\pi)$ and atan2 is the two-argument arctangent function, common in many programming languages. ■

Model 34 (Ideal Magnetic Compass)
Suppose it is known that the field vectors are all directed to the north. This means $f(p) = (0, 1)$ for all $p \in \mathbb{R}^2$. This is, of course, not true of the earth's magnetic field, but we often pretend it is correct. To obtain a compass $y = h(x) = h_{fdo}(x) - \pi/2$, which adjusts for the angular difference between $\theta = 0$ and North, $\theta = \pi/2$. Under these idealized conditions, we should obtain $y = h(p, \theta) = \theta$. ■

Model 35 (Magic Compass)
Without even referring to fields, a kind of "magic" compass can be defined as

$$y = h(x) = h(p, \theta) = \theta. \tag{25}$$

This is a projection sensor, as defined in Model 5. It somehow (magically) obtains the orientation without using fields. ■

We can use Model 34 to simulate Model 35 if the perfect field is given. Using perfect calibration, any given field can be used to simulate this compass by simply transforming the angles produced by Model 33.

3.3 Preimages

3.3.1 The amount of state uncertainty due to a sensor

We have now seen many kinds of virtual sensors, all of which were of the form $h : X \to Y$. What does an observation y tell us about the external, physical state? To understand this, we should

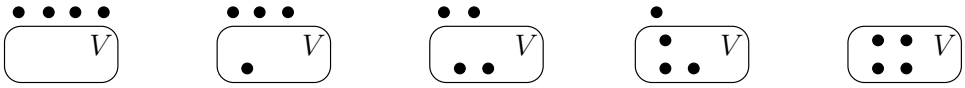


Figure 12: A fixed detection sensor among 4 moving points in $\mathbb{R}^2$ yields these 5 equivalences classes for the partition $\Pi(h)$ of X. In this model, the observation y is the number of points in V.

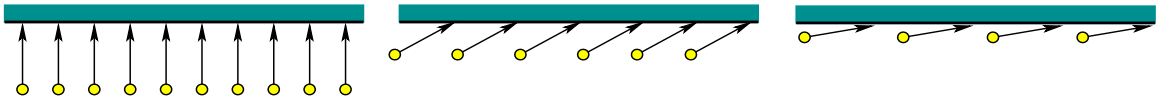


Figure 13: The preimage for a single-directional depth sensor is a two-dimensional subset of $SE(2)$, assuming the environment is given. Shown here are several robot configurations within the same preimage.

think about all states $x \in X$ that could have produced the observation. For a given sensor mapping h this is defined as

$$h^{-1}(y) = \{x \in X \mid y = h(x)\}, \tag{26}$$

and is called the *preimage* of y. If h were invertible, then h^{-1} would represent the inverse; however, because our sensor models are usually many-to-one mappings, $h^{-1}(y)$ is a subset of X, which yields all x that map to y.

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

The connection between h and $\Pi(h)$ is fundamental to sensing. As soon as X, Y, and a sensor mapping are defined, you should immediately think about how X is partitioned. The sets in $\Pi(h)$ can be viewed as equivalence classes. For any $x, x' \in X$, equivalence implies that $h(x) = h(x')$. These states are indistinguishable when using the sensor. In an intuitive way, $\Pi(h)$ gives the sensor's sensitivity to states, or the "resolution" at which the state can be observed. The equivalence classes are the most basic source of uncertainty associated with a sensor.

The following model provides a clear illustration.

Model 36 (Enumeration Sensor)
Suppose that n point bodies move in $\mathbb{R}^2$ and a detection sensor is stalled that counts how many points are within a fixed detection region V. The state space is $X = \mathbb{R}^{2n}$ and observation space is $Y = \{0, 1, \ldots, n\}$. The partition $\Pi(h)$ is formed by $n + 1$ equivalence classes. Figure 12 shows how these subsets of X could be depicted for the case of $n = 4$. If the sensor was 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. ■

Many more interesting partitions of X could be made from the sensors of Section 3.2. Recall the depth sensors of Section 3.2.3. First consider the case of a given polygonal environment, leading to $X = E \times S^1$. For Model 6, each $h^{-1}(y)$ is generally a two-dimensional subset of X 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 $E \times S^1$. For example, equivalent states along a single wall are depicted in Figure 13. Using the boundary sensor, Model 9, $\Pi(h)$, contains

only two classes: All states in which the robot is in the interior of E, and all states in which it is on the boundary of E. The omnidirectional depth sensor, Model 12, is quite powerful. This leads to very small preimages. In most cases, these correspond to the finite set of symmetry classes of the environment. Such symmetries are usually encountered in robot localization. For example, in environment at the extreme left of Figure 8, $h^{-1}(y)$ is a three-element set, corresponding to the three possible orientations at which the same observation could be obtained.

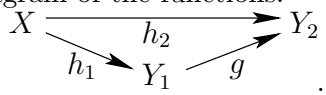
Now suppose that the environment is unknown, leading to $X \subset SE(2) \times \mathcal{E}$. Each $h^{-1}(y)$ contains a set of possible environment and robot configuration pairs that could have produced the observation. In the case of a boundary sensor, $h^{-1}(1)$ would mean "all environments and configurations in which the robot is touching a wall". For the omnidirectional sensor, $h^{-1}(y)$ indicates all ways that the environment could exist beyond the field of view of the sensor.

3.4 The Sensor Lattice

After seeing so many 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.

For all of the discussion in this section, assume that the state space X is predetermined and fixed. Let $h_1 : X \to Y_1$ and $h_2 : X \to Y_2$ be any two sensor models (recall the great variety from Section 3.2). We say that h_1 *dominates* h_2 if and only if $\Pi(h_1)$ is a refinement of $\Pi(h_2)$. This is denoted as $h_1 \succeq h_2$.

For some state $x \in X$, imagine receiving $y_1 = h_1(x)$ and $y_2 = h_2(x)$. If $h_1 \succeq h_2$, then $h_1^{-1}(y_1) \subseteq h_2^{-1}(y_2) \subseteq X$. This clearly means that h_1 provides at least as much information about x as h_2 does. Furthermore, using y_1, we could infer what observation y_2 would be produced by h_2. Why? Since $\Pi(h_1)$ is a refinement of $\Pi(h_2)$, then every $x \in h_1^{-1}(y_1)$ must produce the same observation $y_2 = h_2(x)$. This implies that there exists a function $g : Y_1 \to Y_2$ such that $h_2(x) = g(h_1(x))$, written as $h_2 = g \circ h_1$. Here is a diagram of the functions:



The existence of g implies that h_1's observations can be used to "simulate" h_2, without needing additional information about the state. One important point, however, is that it might be computationally impractical or infeasible to compute g in practice. The decidability and complexity of computing g lead to interesting open research questions for various sensing models.

Using the dominance relation $\succeq$, we can naturally compare many of the sensors in Section 3.2. Note that $\succeq$ is a partial ordering; most sensor pairs are incomparable. Figure 14 shows how some sensors are related. The most powerful sensor of Section 3.2.3 is the omnidirectional depth sensor because it induced the finest partition of X. We can use it to simulate all other sensors in that section. For the directional sensors, it is assumed that the directions are properly aligned. Since gaps are just discontinuities in the depth function, the depth sensors can even be used to simulate gap sensors, such as Models 24 and 25. Note that these relationships hold regardless of the particular collection $\mathcal{E}$ of possible environments. It does not matter whether the environment is given or is open to some infinite collection of possibilities.

Other sensors could be added to Figure 14. For example, the dummy sensor, Model 1, is dominated by all of these sensors. Furthermore, the identity sensor, Model 2 dominates all of these. The same is true of the bijective sensor, Model 3, since both induce the same partition of X.

What happens as we include more and more sensors, and continue to extend the diagram in Figure 14? It is truly remarkable that *all* possible sensors of the form $h : X \to Y$ over a fixed state

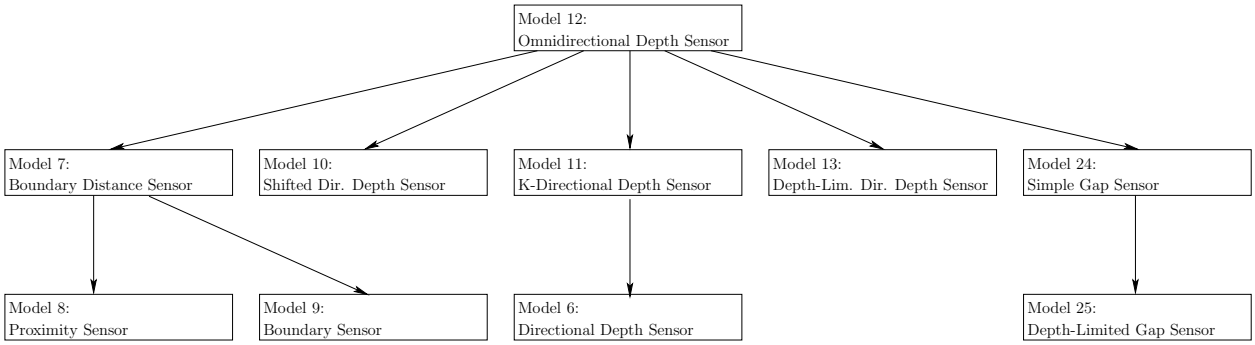


Figure 14: Several models from Section 3.2 are related using the idea of dominance, based on refinements of the partitions they induce over X. Models higher in the tree induce finer partitions. A lower sensor model can be "simulated" by any model along the path from the root of the tree to itself.

space X can be related in a clear way, and the tree extends into a lattice.

Note that Y is not fixed, meaning we could take any set Y and define any mapping $h : X \to Y$. Consider defining an equivalence relation $\sim$ on this enormous collection of sensors: We say that $h_1 \sim h_2$ if and only if $\Pi(h_1) = \Pi(h_2)$. For example, Models 2 and 3 are equivalent because the both induce the same partitions of X (all preimages are singletons). More precisely, Model 3 is a family of sensors, which includes Model 2; however, the entire family is equivalent.

If we no longer pay attention to the particular h and Y, but only consider the induced partition of X, 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 X.

The relationship between sensors in terms of dominance then leads to the well-known idea of a *partition lattice*, depicted in Figure 15 for the set $X = \{1, 2, 3, 4\}$. Recall that a lattice is a set together with a partial order relation $\succeq$ for which every pair of elements has a least upper bound and a greatest lower bound. 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.

Now observe that for any state space X, all possible sensors fit nicely into the partition lattice of X. Furthermore, $\succeq$ indicates precisely when one sensor dominates another. The tree depicted in Figure 14 is embedded in this lattice. The partition corresponding to the bijective sensor, Model 3, is at the top of the lattice because it is the finest partition possible. The dummy sensor, Model 1 is at the bottom of the lattice because it is the coarsest partition possible.

An important property of a lattice is that every pair of elements has a unique *greatest lower bound* (glb) and a unique *least upper bound* (lub). These have an interesting interpretation 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 state $x \in X$. Let $y_1, \ldots, y_4$, be the observations obtained by applying $h_1, \ldots, h_4$, respectively. An element of $\Pi(h_3)$ is obtained by intersecting preimages, $h_1^{-1}(y_1) \cap h_2^{-1}(y_2)$. There is a straightforward way to construct some representative h_3 from h_1 and h_2. Let $Y_3 = Y_1 \times Y_2$ and $h_3 : X \to Y_3$ be defined as $y_3 = (y_1, y_2) = (h_1(x), h_2(x))$. 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. Again starting from any $x \in X$, $\Pi(h_4)$ is the finest partition for which $h^{-1}(y_1) \cup h^{-1}(y_2) \subseteq h^{-1}(y_3)$.

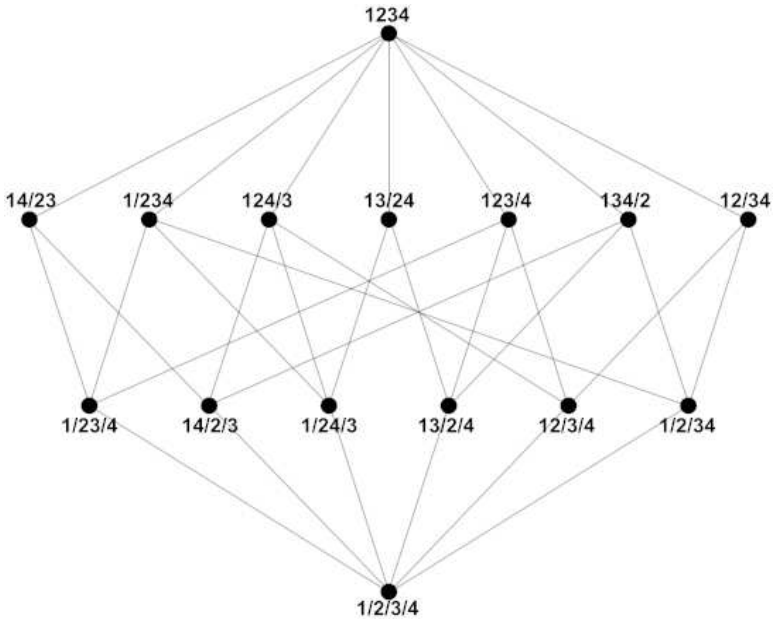


Figure 15: The partition lattice for a four-element set (copied from Wikipedia).

One way to "visualize" these relationships is to imagine the case in which $X = Y = \mathbb{R}^3$ and restrict the set of all sensor mappings to be only linear ones, $y = Cx$, as given in Model 4. If C has rank 2, then the preimages $h^{-1}(y)$ are lines in $\mathbb{R}^3$. Consider two linear sensors, with matrices C_1 and C_2 having rank 2. The glb produces preimages that are the intersection of two lines. The lines must always intersect because both preimages are observations of same state $x \in \mathbb{R}^3$. If the combined 3 by 6 matrix, $[C_1\ C_2]$, has rank three, then all preimages will be points, and the glb is a bijective sensor. The preimages for the lub in this case is the set of all planes in $\mathbb{R}^3$. Each plane is obtained by taking the union of the preimages, which forms a pair of intersecting lines.

3.5 Additional Complications

Up until now, all sensor models have been idealized because the observation is immediately and completely determined by applying $y = h(x)$. Although preimages $h^{-1}(y)$ reveal important information about sensing problems, additional concerns have so far been neglected. This section handles three independent issues: 1) the sensor observation might not be predictable, even if the state is known, 2) the observation might depend on state and time, and 3) the observation might depend on one or more previous states, rather than being instantaneous.

3.5.1 Nondeterministic Disturbance

Suppose that a sensor provides an observation instantaneously, but there is uncertainty about which observation will occur at state x. We consider two general models to address this: Nondeterministic and probabilistic. First, the nondeterministic model will be defined, which might alternatively be called *possibilistic*. In this case, the sensor mapping specifies a *set of possible observations* that could be obtained from some state $x \in X$.

Let an observation space Y be defined as before and let X be any state space. A *nondeterministic sensor mapping* is defined as

$$h : X \rightarrow \text{pow}(Y) \tag{27}$$

in which pow(Y) is the power set of Y, to yield any possible subset. For a state $x \in X$, the mapping h yields a set $h(x) \subseteq Y$. An alternative to (27) is to define a *nature action* that generates a disturbances and construct a function that maps states and nature actions into observations; see Section 11.1 of [14].

Before giving examples of (27), consider the effect on preimages. A reasonable definition of a preimage for a nondeterministic sensor mapping h is

$$h^{-1}(y) = \{x \in X \mid y \in h(x)\}. \tag{28}$$

If $h(x)$ is a singleton subset for all $x \in X$, then (28) reduces to the original preimage (26). As $h(x)$ grows in size, the preimages become larger. Rather than a partition of X, a *cover* of X is obtained, denoted by $\mathcal{C}(h)$. This means that the union of all $h(x)$ is equal to X, but the preimages are not necessarily disjoint.

Model 37 (One-Dimensional Position Sensor)
Let $X = Y = \mathbb{R}$. Imagine a sensor the measures the position along X. Let $\epsilon > 0$ be a bound in the maximum amount of measurement error. The nondeterministic sensor mapping is

$$h(x) = \{y \in Y \mid |x - y| \leq \epsilon\}. \tag{29}$$

For example, $h(2) = [2-\epsilon, 2+\epsilon]$. The actual observation produced by the sensor may be any value $y \in [2 - \epsilon, 2 + \epsilon]$.

The preimage of an observation y is

$$h^{-1}(y) = \{x \in X \mid |x - y| \leq \epsilon\}. \tag{30}$$

For example, $h^{-1}(5) = [5 - \epsilon, 5 + \epsilon]$. Clearly, the preimages of h yield a cover of $X = \mathbb{R}$. ■

Nondeterministic versions of the sensors in Section 3.2 can be easily constructed. A couple of examples are given here.

Model 38 (K-Dimensional Position Sensor)
Model 37 can be easily extended to k dimensions, usually with $k = 2$ or $k = 3$. Reusing $\epsilon > 0$, the sensor mapping could be

$$h(x) = \{y \in Y \mid \|x - y\| \leq \epsilon\}, \tag{31}$$

in which $\| \cdot \|$ is the Euclidean norm. In this case, $h(x)$ is a disc of radius ϵ, as are the preimages of h. ■

Model 39 (Faulty Detectors)
Consider modifying a static binary sensor given by (10). The sensor might produce a *false positive* by yielding $h(p, E) = 1$ even though $p \notin V$. In this case, the preimage would be $h^{-1}(1) = X$. If the sensor could also produce a *false negative* by yielding $h(p, E) = 0$ when $p \in V$, then $h^{-1}(0) = X$. These two preimages together cover X 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. ■

The notion of dominance from Section 3.4 can be extended to the nondeterministic sense. Once again, we must determine whether one sensor can "simulate" the other. We say that h_1 *dominates* h_2 another if there exists a mapping $g : Y_1 \rightarrow \text{pow}(Y_2)$ such that for all $y_1 \in h_1(x)$, $g(y_1) \subseteq h_2(x)$.

Model 40 (Inaccurate Directional Depth)
Recall Model 6. For any $\epsilon \geq 0$, we can define a mapping

$$h_\epsilon(p, \theta, E) = \{y \in [0, \infty) \mid |||p - b(x) - y||| \leq \epsilon\}. \tag{32}$$

Note the similarity to (29). For this model, h_ϵ dominates $h_{\epsilon'}$ if and only if $\epsilon \leq \epsilon'$. ■

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 (27), in which $h(x)$ yields a set of possible observations in Y. The models in this section place a probability density over $h(x)$. A convenient way to express this is

$$p(y|x), \tag{33}$$

which is a probability density function over Y, but conditioned on the particular state, $x \in X$. Unfortunately, this representation hides the underlying sensor mapping. Using h that we can declare $p(y|x) = 0$ for all $y \notin h(x)$, which is powerful information that is not reflected in (33). 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.

Model 41 (Probabilistic 1D Position Sensor)
We first make a probabilistic variant of Model 37. Assume the error density is Gaussian with zero mean and variance σ^2. The probability density function is

$$p(y|x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-y)^2}{2\sigma^2}}. \tag{34}$$

This function is maximized when $x = y$, which corresponds to the case of no disturbance. ■

Model 42 (Probabilistic General Position Sensor)
Now consider a probabilistic variant of Model 38. Assume the error density is Gaussian with zero mean and Σ as a $k \times k$ covariance matrix. The density function is

$$p(y|x) = \frac{1}{|\Sigma|^{1/2}(2\pi)^{k/2}} e^{(y-x)^T \Sigma^{-1}(y-x)}. \tag{35}$$

■

Model 43 (Probabilistic Detectors)
Revisiting Model 39, simply attach probabilities to false positives and false negatives. For a false positive, we define $p(y = 1 \mid p \notin V)$. The condition $p \notin V$ could be replaced with a more precise

location for p, to allow conditional probabilities that vary because of the state. Note that the "correct" positive is obtained by subtracting the above probability from one. Likewise, a false negative is defined by $p(y = 0 \mid p \in V)$. ■

Model 44 (Probabilistic Directional Depth)
A generalization of Model 40, again assuming a zero-mean Gaussian density for disturbances, is

$$p(y|p, \theta, E) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-\|p-b(x)\|)^2}{2\sigma^2}}. \tag{36}$$

■

3.5.3 Sensors Over State-Time Space

Recall from Section 3.1.4, that once time is introduced, the state-time space $Z = X \times T$ is used to fully describe any situation. In this case, the sensor should logically be defined over Z, rather than X. This means that a combination of state $x \in X$ and time $t \in T$ produces an observation y. Thus, the basic sensor mapping (1) is replaced by

$$h : Z \rightarrow Y, \tag{37}$$

for which we write $y = h(z)$, or equivalently, $y = h(x, t)$.

Here is a simple example.

Model 45 (Perfect Clock)
This sensor simply reports the current time:

$$y = h(z) = h(x, t) = t. \tag{38}$$

■

The basic sensing examples from Section 3.2.2 can be easily extended by replacing X with Z. They could, for example, report time in addition to their existing observation. Here is an example that extends Model 14.

Model 46 (Detector With Time Stamp)
Use the same model for V, E, and p as in Model 46. Let $Y = \{0, 1\} \times T$. The sensor mapping $h : Z \times Y$ is

$$h(p, E, t) = \begin{cases} (1, t) & \text{if } p \in V \text{ at time } t \\ (0, t) & \text{otherwise.} \end{cases} \tag{39}$$

■

All of the concepts from Section 3.3 extend naturally from X to Z. A preimage under the sensing model in (37) is

$$h^{-1}(y) = \{(x, t) \in Z \mid y = h(x, t)\}, \tag{40}$$

Now consider partitions $\Pi(h)$ over Z. A weak sensor may partition Z into large chunks of state-time space. Following Section 3.4, a sensor h_1 dominates another h_2 if and only if its partition $\Pi(h_1)$ of Z is a refinement of $\Pi(h_2)$. In the same way as for X, a partition lattice over Z is obtained.

The concepts from Sections 3.5.1 and 3.5.1 can be adapted here to yield nondeterministic and probabilistic sensor models. This results in $h : Z \rightarrow \text{pow}(Y)$ for the nondeterministic case and $p(y|z)$ for the probabilistic case.

3.5.4 History-Based Sensors

As a natural transition to the temporal filters of Section 4, we consider one final extension to the sensor models. It might be the case that the sensor output depends on a *history* of previous states. The most common examples in practice are odometers, such as the wheel encoder in Figure 4(e). They accumulate changes over time and report the aggregate amount, such as total distance traveled. The relationship to Section 4 is that the sensors here could be realized by employing a filter that uses information from instantaneous sensors (such as $h : X \rightarrow Y$). In other words, a history-based sensor usually contains a built-in filter. This should become clearer after reading Section 4.

Let a *state trajectory* up to time t be denoted as $\tilde{x} : [0, t] \rightarrow X$. The set of all possible trajectories for any possible $t \in T$ is called the *trajectory space* and is denoted by $\tilde{X}$. For a *history-based sensor*, the sensor mapping is

$$h : \tilde{X} \rightarrow Y. \tag{41}$$

In this case, a given state trajectory $\tilde{x}$ produces an observation $y = h(\tilde{x})$.

Once again, the notions of preimages, partitions, and the sensor lattice naturally extend. In this case, X is simply replaced by $\tilde{X}$. For example,

$$h^{-1}(y) = \{\tilde{x} \in \tilde{X} \mid y = h(\tilde{x})\}, \tag{42}$$

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

Some examples of history-based sensors follow.

Model 47 (Linear Odometer)
Suppose we would like to measure how far a robot has traveled. Let (v_x, v_y) represent the instantaneous velocity of a planar robot. A history-based sensor could integrate the magnitude of velocity obtain the distance:

$$y = \theta_0 + \int_0^t \sqrt{v_x^2 + v_y^2} ds. \tag{43}$$

This model implies that v_x and v_y are components of state (as is common in classical mechanics and modern control theory). For a robot moving in a given planar environment, the state could be represented as $x = (p_x, p_y, \theta, v_x, v_y)$. ■

Model 48 (Angular Odometer)
Recall the wheel encoder of Figure 4(e). An idealized model of this sensor can be made by considering the limit after making smaller and smaller holes along the disc. This results in a perfect *angular odometer*, which is modeled as

$$y = \theta_0 + \int_0^t \dot{\theta}(s) ds, \tag{44}$$

in which y measures the net orientation change from some starting orientation. What would the sensor report if $|\dot{\theta}(s)|$ is integrated instead? ■

In practice, sensors cannot actually produce instantaneous observations. Using a history-based sensor, the delay can be explicitly modeled:

Model 49 (Delayed Measurement)
Suppose a sensor measures the state perfectly, but it takes one unit of time to output the result. This can be modeled as

$$y = \begin{cases} \tilde{x}(t-1) & \text{if } t \geq 1 \\ \# & \text{otherwise,} \end{cases} \tag{45}$$

in which # means that the state cannot yet be determined. A delayed version of any sensor of the form $h : X \rightarrow Y$ or $h : Z \rightarrow Y$ can be made in this way. ■

Model 50 (Discrete-Time Odometer)
Without referring directly to velocities, a history-based sensor can be constructed that estimates the distance traveled by comparing positions reported at various times. Consider some $\Delta t > 0$, corresponding to a fixed time increment. Let $\tilde{p}(t)$ denote the robot position in $\mathbb{R}^2$ at time t; this can be determined from $\tilde{x}(t)$.

The sensor mapping is

$$h(\tilde{x}) = \sum_{i=1}^{\lceil t/\Delta t \rceil} \|\tilde{p}(i\Delta t) - \tilde{p}((i-1)\Delta t)\|. \tag{46}$$

For a state trajectory $\tilde{x} : [0, t] \rightarrow X$, the total distance traveled is estimated. The quality of the estimate depends on how small Δt is selected. This sensor is essentially constructed as a temporal filter, which will be covered in Section 4.2. ■

4 Filtering

A *filter* combines information from multiple observations to efficiently keep track of state properties that are needed for inference tasks. Although it comes with different historical connotations, an alternative term is *sensor fusion*. The observations may come from different sensors, which either appear in the same location or are distributed around the same environment. In these cases, a *spatial filter* sews the information together to make a coherent view. Alternatively, the observations may arrive sequentially over time from the same sensor. This leads to a *temporal* or *causal filter*, which incrementally updates its internal information for each new observation. A filter may even combine both spatial and causal elements. In the most general settings, we must resolve observations taken at different times from various sensors distributed around the environment.

4.1 Spatial Filters

Here we suppose that some sensors have been distributed around the environment and each has produced an observation. How is this information interpreted? The answer is mostly provided by analyzing preimages, which were introduced in Section 3.3.

4.1.1 A general triangulation principle

Suppose that several instantaneous sensors produce their observations at the same time. Techniques have been widely used for hundreds of years for combining information from these to obtain important quantities such as distance, longitude, and latitude. Most of this section is a generalization of the ancient idea of triangulation, in which observing the angles between pairs of distant

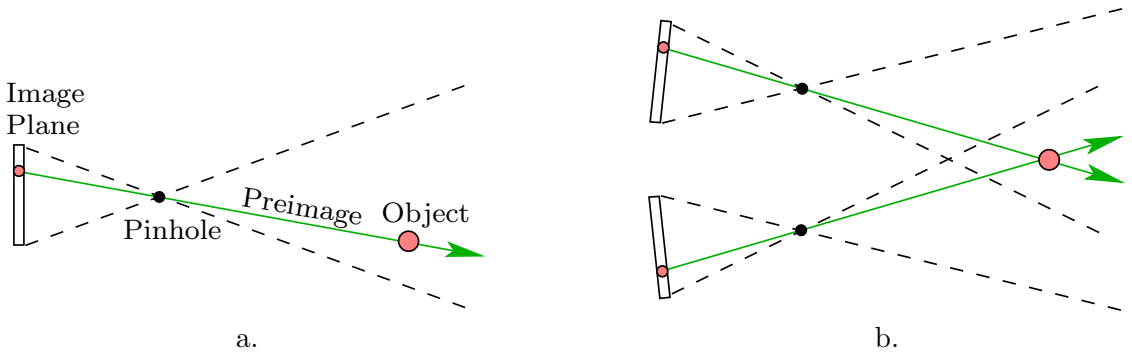


Figure 16: a) Using the pinhole camera model, the preimage of a point object is an infinite ray that that connects from the pinhole to the object. b) From matching the same object in two camera images, its location can be determined by intersecting the two preimage rays.

features has been used to infer location. We now develop this idea in a general way using concepts of Section 3.

Consider any n sensor mappings $h_i : X \rightarrow Y_i$ for i from 1 to n. If each produces an observation $y_i \in Y_i$ at some common instant, then what are the possible states? The *triangulation*[3] of the observations is denoted by $\Delta(y_1, \ldots, y_n)$. It is determined by intersecting the preimages (26) of each sensor to obtain

$$\Delta(y_1, \ldots, y_n) = h_1^{-1}(y_1) \cap h_2^{-1}(y_2) \cap \cdots \cap h_n^{-1}(y_n), \tag{47}$$

which is a subset of X. It is the set of all states that could possibly have produced the n observations simultaneously.

Here are some important, classical examples of triangulation.

Filter 1 (Stereopsis)
Figure 16(a) shows a small object appearing in an image under the pinhole camera model. This can be imagined as an object in $X = \mathbb{R}^3$ with a 2D image, or an object in $X = \mathbb{R}^2$ with a one-dimensional image. The sensor mapping $h : X \rightarrow Y$ is the standard perspective projection model, in which $y \in Y$ represents the location of the object in the image. The preimage $h^{-1}(y)$ is a ray that extends outward from the pinhole and through the object in $\mathbb{R}^3$. Figure 16(b) illustrates the principle of stereopsis, commonly used to locate objects in human vision and computer vision systems. In this case, the preimages are intersected to reveal the precise object location. This assumes that the positions and orientations of the cameras are known. ■

Filter 2 (Ancient Triangulation)
Figure 17 shows classical triangulation, which is a technique used for thousands of years by ancient Greeks, Egyptians, and Chinese. The sensor mapping provides the angle between a pair of landmarks, as observed from the sensor location. To understand the preimage, depicted in Figure 17(a), imagine moving around in the plane while holding the angle between two landmarks fixed. What curve do you trace? It turns out to be a circular arc; however, there are two arcs depending on whether you are "in front of" or "behind" the landmarks. The example shown is for angle

[3]This is completely different from triangulations in computational geometry, topology, or meshing.

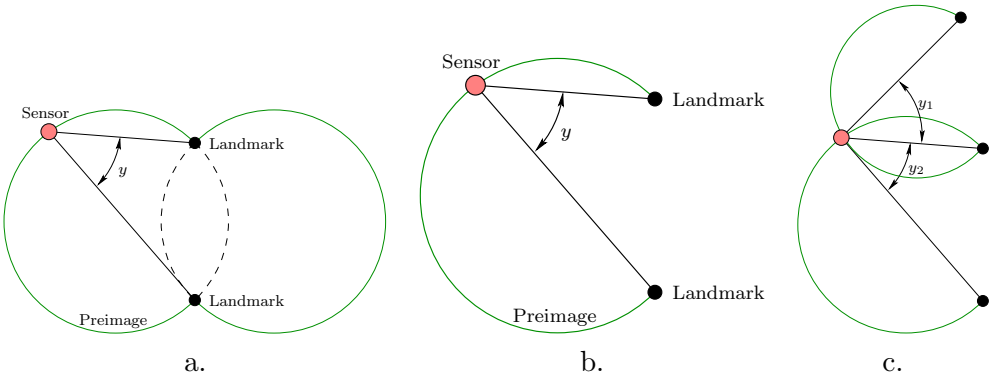


Figure 17: The ancient triangulation technique uses observations of the angle between pairs of landmarks. (a) Holding the angle between two landmarks fixed, the preimage is a portion of two circles. (b) If it is known which landmark is to the left, then half of the original preimage is eliminated to obtain a circular arc. (c) With three landmarks, with known left-to-right order, the circular arcs are intersected to determine the sensor location.

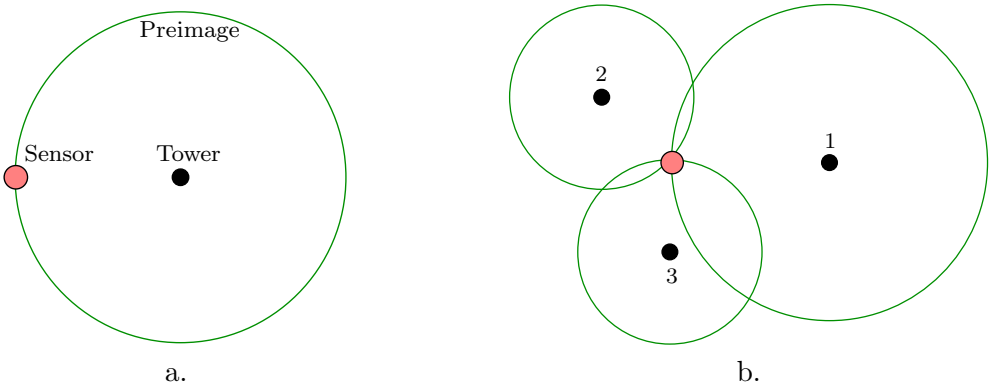


Figure 18: The principle of trilateration enables the sensor location to be determined from observations of distances to known towers. a) From a single tower, the sensor could be anywhere along a circle. b) Using three towers, the position is uniquely determined.

observations $y < \pi/2$. If $y = \pi/2$, then the two arcs fuse into a single circle. If $y > \pi/2$, then the preimages appears as the dashed lines in Figure 17(a). If there is not front/behind ambiguity, then one arc can be eliminated, as depicted in Figure 17(b). If there are three landmarks, then two angles are obtained and the preimage arcs are intersected to obtain the precise sensor location, shown in Figure 17(c). ■

Filter 3 (Trilateration)
Figure 18 shows the principle of *trilateration*. Rather than observing the angle between landmarks, imagine that a sensor observes the distance to a landmark. If both the landmark and sensor have synchronized clocks, then a virtual sensor can be constructed that estimates the distance based on the *time of arrival* (TOA) of the signal. This assumes a known propagation speed for sound waves or radio signals. Suppose that $X = \mathbb{R}^2$ and the landmark location is known. The sensor mapping

$h : X \rightarrow Y$ yields a distance $y = h(x) \in (0, \infty)$. The preimages $h^{-1}(y)$ are circles of radius y, centered at the landmark.

Now consider determining the precise sensor location. If there are two landmarks, the intersection of preimages yields a pair of points because two circles intersect at two points in general. If there are three landmarks, then three circles are intersected to obtain the precise sensor location. This is the principle of trilateration.

If $X = \mathbb{R}^3$, then the preimages become spheres. In this case, four landmarks are needed: A pair of intersected spheres yields a circle, three spheres intersected yields two points, and four finally yield a unique point. ■

Filter 4 (Hyperbolic Positioning)
One shortcoming of trilateration is the precise clock synchronization needed to determine the signal time of arrival. Suppose that the towers are synchronized but the sensor is not. If the towers send their signals at the same time, then the sensor can instead use a chronometer to measure the differences in their arrival times. This is called *time difference of arrival (TDOA)*. If the signal propagation speed is known, then the virtual sensor yields the relative distances for each pair of towers. For example, it might tell us that Tower 1 is 432 meters closer to the sensor than Tower 2. What is the preimage in this case? The set of all points in the plane in which one tower is a fixed distance closer than another fall along a hyperbolic curve. The preimage in the three-dimensional case is one sheet of a hyperboloid. The method of *hyperbolic positioning* involves intersecting the hyperbolic preimages from multiple observations, one for each pair of towers, to obtain the precise location. This technique was used in the Decca Navigator System in World War II to locate ships, and in modern times it can be applied to localize a cell phone user using multiple cell towers. ■

How much can be learned about $x \in X$ from the observations? For any two sensors $h_1 : X \rightarrow Y_1$ and $h_2 : X \rightarrow Y_2$, consider their associated partitions $\Pi(h_1)$ and $\Pi(h_2)$. Let their observations be combined to produce one stronger sensor $h_3 : X \rightarrow Y_1 \times Y_2$, defined as $h_3(x) = (h_1(x), h_2(x))$. In terms of partitions, $\Pi(h_3)$ is just the common refinement of $\Pi(h_1)$ and $\Pi(h_2)$. Furthermore, h_3 is just the greatest lower bound (glb) in the sensor lattice over X; recall Section 3.4.

Based on the filter models just presented, it is natural to wonder how information improves when sensors are combined. This depends on the properties of the preimages. Suppose, for example, that the sensor mappings are linear $y_i = C_i x$, with $Y = \mathbb{R}^{m_i}$ and $X = \mathbb{R}^n$. Each $h_i^{-1}(y_i)$ is a hyperplane through the origin of X, with its dimension depending on m_i and the rank of C_i. Let every C_i have full rank. If $m_i = n$, then $h_i^{-1}(y_i)$ indicates a unique $x \in X$ with a single observation. If $m_i = 1$, then $\Delta(y_1, \ldots, y_k)$ produces a unique $x \in X$ only if $k = n$ and the C_i column vectors are linearly independent. Moving away from this linear example, we generally consider nonlinear preimages. However, the properties of *preimage dimension* and *sensor mapping independence* remain critical to characterizing $\Delta(y_1, \ldots, y_n)$.

4.1.2 Handling disturbances

In most applications of the triangulation principle, sensor observations are not perfectly predictable, even when the state is given. Therefore, it is crucial to consider disturbances. For the case of nondeterministic sensor mappings (27), the enlarged preimage (28) is obtained. In that case, (47) is simply applied to the new preimages. One problem, however, is that the intersection of preimages might not be small enough to determine the state, regardless of how many observations are obtained. For example, suppose Filter 3 considers disturbances on the distance measured to the tower. If

upper and lower bounds are specified on the distance, then each preimage is an annulus. If the distance error is at most $\epsilon > 0$, then the annulus has thickness 2ϵ. The intersection of several annuli will usually not result in a point.

This issue could motivate the consideration of disturbances probabilistically. The generalized triangulation principle can be adapted to the probabilistic case, in which the original set intersection is replaced by Bayes' rule. The probabilistic analog to (47) is

$$p(x|y_1, \ldots, y_n) = \frac{p(y_1|x)p(y_2|x) \cdots p(y_n|x)p(x)}{p(y_1, \ldots, y_n)}. \tag{48}$$

If we forget about making normalized probability density functions and define $p(y_i|x) = 1$ if and only if $y_i \in h(x)$, then (48) is equivalent to (47). It can therefore be considered as an extension, assuming that such probabilities are realistically available.

In the probabilistic setting, it becomes reasonable to use many more observations than were minimally needed in Filters 1 to Filter 4. This allows as much information as possible to be applied to reduce ambiguity. In many problems the goal is to estimate the state x that is producing the observations. Suppose $\hat{x}$ is a guess of the state. Let $d_i(\hat{x}, y_i)$ denote the Euclidean distance in X from $\hat{x}$ to the nearest point on the preimage $h^{-1}(y_i)$. State estimation can now be formulated as an optimization problem: Try to find $\hat{x}$ that minimizes the distances d_i. How are all d_i optimized simultaneously? One popular idea is to convert them to a scalar criterion, which often results in the *least squares* optimization problem:

$$\min_{\hat{x} \in X} \sum_{i=1}^{n} d_i^2(\hat{x}, y_i). \tag{49}$$

This optimization is equivalent to *maximum likelihood estimation* in the probabilistic setting if the measurement errors are assumed to be distributed as zero-mean Gaussian densities. Otherwise, it can be simply viewed as a way to select $\hat{x}$ by reducing total error.

4.1.3 Spatial filters over state-time space

Recall from Section 3.5.3 that a sensor can also be defined over state-time space Z to obtain $h : Z \rightarrow Y$. In this case, the concepts discussed so far extend naturally over time. We use the preimage definition (40) for triangulation in (47). In this case, a spatiotemporal filter can be made that gains information simultaneously about state and time. An important example of this is GPS units. There is a difficult time synchronization problem, which makes it hard to use the simple trilateration model (Filter 3). GPS units therefore intersect preimages in Z to infer both position and time, simultaneously. Minimally, one additional satellite is needed to sufficiently reduce the dimension of the set of possible $z = (x, t)$ values. This scheme generally requires 5 satellites; however, 4 is often cited the minimum number because the constraint that people stand on the earth is used to eliminate one. More satellites may be used, however, resulting in an optimization similar to (49) to overcome disturbances.

4.2 Temporal Filters

This section introduces filters that combine measurements over time.

4.2.1 The Inference Problem

Recall from Section 3.5.4 the notion of a state trajectory $\tilde{x} : [0, t] \rightarrow X$ and the space $\tilde{X}$ of all state trajectories. Now suppose that over the interval of time $[0, t]$, we have gathered and recorded sensor

observations. If this could be done continuously, we obtain what would look like a "observation trajectory". Since observations may jump around erratically, rather than appearing as motions in the physical world, we instead call them the *observation history*, which is defined as a function $\tilde{y} : [0, t] \rightarrow Y$.

When presented with $\tilde{y}$, there are two fundamental questions:

1. What is the set of state trajectories $\tilde{x} : [0, t] \rightarrow X$ that might have occurred?

2. What is the set of possible current states, $\tilde{x}(t)$?

Using a single observation $y = h(\tilde{x}(t))$, we could answer the second question by the preimage $h^{-1}(y)$ from (26). However, based on an entire history $\tilde{y}$, we may able to further narrow down the set of possibilities. This will be the purpose of a temporal filter.

Before preceding to the filter details, we first introduce some concepts that help to illuminate the two fundamental questions above. Suppose that a sensor $h : X \rightarrow Y$ is given and is applied over an interval of time $[0, t]$. For every $t' \in [0, t]$ some observation $\tilde{y}(t') = h(\tilde{x}(t'))$ is obtained. This means that we can use h, applied over $[0, t]$, to define a mapping

$$H : \tilde{X} \rightarrow \tilde{Y}, \tag{50}$$

in which $\tilde{Y}$ is the set of all possible observation histories. For any sensor mapping $h : X \rightarrow Y$, the mapping H is automatically induced.

To understand the two fundamental questions, we now only need to look at preimages of H. Compare this to (26):

$$H^{-1}(\tilde{y}) = \{\tilde{x} \in \tilde{X} \mid \tilde{y} = H(\tilde{x})\}. \tag{51}$$

This is the "answer" to the first fundamental question: It yields the set of all state trajectories that could have produced $\tilde{y}$. Computing or even explicitly representing this answer could be extremely challenging.

The second fundamental question can also be addressed using H. Here is one way to express the answer:

$$\{x \in X \mid \exists \tilde{x} \in H^{-1}(\tilde{y}) \text{ such that } \tilde{x}(t) = x\} \tag{52}$$

A simple example will now be presented.

Example 1
Consider the upper part of Figure 19(a), in which there are four edges in an embedded graph that represents the state space X. For convenience, the edges are labeled from a to d. A point body travels around inside of the graph, and we would to determine where it has gone using a simple sensor that measures only the horizontal coordinate. In other words, for any $p \in X$, with coordinates $p = (p_x, p_y)$, the sensor yields $y = h(x) = p_x$.

Consider the preimages of h for various observations. If the body moves in edge d, then the state can be immediately inferred from the sensor observation. The preimage $h^{-1}(y)$ consists of a single point in d. If the body moves in the other edges, then $h^{-1}(y)$ contains three points, one inside of each of edge a, b, and c.

Now think about the two fundamental questions above. Imagine that the body has been moving around in X for some time, and we want to reason about where it has been. Given $\tilde{y} : [0, t] \rightarrow Y$, what trajectories are possible, assuming the body must move on a continuous path? Any trajectory portion that places the body in edge d can be completely determined from the observations. There is ambiguity only when the body is in other edges. If the body moves from edge d to the left, then it can only enter edge b or c. The observation preimage indicates that the body could be in a, b,

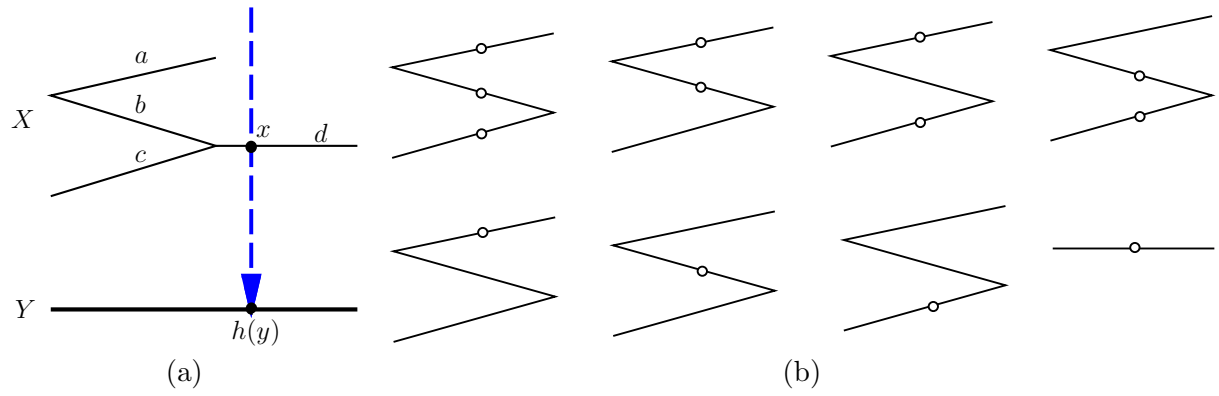


Figure 19: (a) Imagine trying to infer the location of a point on a planar graph while observing only a single coordinate. This simple example involves a point moving along a graph that has four edges. When the point is on the rightmost edge, there is no uncertainty; however, uncertainty exists when the point travels along the other edges. (b) The various possible I-states are shown, based on sensor preimages.

or c; however, the history that the body came from d eliminates the possibility of being in edge a. This nicely answers the second basic question, which is formulated mathematically in (52).

For the first question, we consider entire possible trajectories. For this simple problem, the set is always finite. For a simple example, imagine a trajectory that starts the body in the midpoint of d at time $t = 0$. The body then moves left to the midpoint of edge b or c, arriving at time $t = 1/2$. The body then returns to the right, reaching the midpoint of edge d at time $t = 1$. Based on $\tilde{y} : [0, 1] \rightarrow Y$, the preimage $H^{-1}(\tilde{y})$, defined in (51), contains only two possible trajectories. One trajectory moves the body up to edge b and then returns. The other moves the body down to edge c and returns. Note that the particular timing information is not ambiguous because using $\tilde{y}$, we know exactly which the time the body must be in various positions. The only ambiguity is which edge, b or c, contains the body during part of the trajectory. If the sensor did not report all of the timing information, then $H^{-1}(\tilde{y})$ could be infinite. Imagine, for example, the possible trajectories if the sensor reports the horizontal position only once every 0.1 seconds. Any continuous trajectory segment is possible in between the observations. ■

In the spirit of Section 3.4, we can talk about the partition of $\tilde{X}$, denoted by $\tilde{\Pi}$, that is induced by H. Note that $\tilde{\Pi}(h_1)$ is a refinement of $\tilde{\Pi}(h_2)$ if and only if $h_1 \succeq h_2$. This means that a more powerful sensor produces a better filter, which we would expect.

4.2.2 The structure of a temporal filter

Discretely indexed observation histories Suppose that for some sensor model $h : X \rightarrow Y$, observations arrive at discretely over time, to yield a sequence $\tilde{y} = (y_1, \ldots, y_k)$ observations. Each y_i will be said to correspond to a *stage* i. Depending on the particular model, it may or may not be known when each y_i occurred; however, it will always be assumed that y_{i+1} is obtained at a later time than y_i for every i from 1 to $k - 1$.

In the previous model of observation histories $\tilde{y} : [0, t] \rightarrow Y$, the state was *always* being observed by the sensor. Under this new model, $\tilde{y}$ provides observations at some discrete points along time.

In between these observations, the sensor does not make any other observations. Based on this situation, two distinguishing features of temporal filters arise:

1. Since observations arrive incrementally, we define filter information that can be updated incrementally.

2. A model is needed of how the state might change over time, particularly when no observations are available.

Let $\mathcal{I}$ be any set, and call it an *information space*[4] (or *I-space*) and let any $\iota \in \mathcal{I}$ be called an *information state* (or *I-state*). A *filter* has two components:

1. A given *initial I-state*, denoted by $\iota_0 \in \mathcal{I}$.

2. A *transition function* ϕ of the form $\phi : \mathcal{I} \times Y \rightarrow \mathcal{I}$. In particular, the structure is

$$\iota_k = \phi(\iota_{k-1}, y_k), \tag{53}$$

in which the *new* I-state ι_k is determined from the *previous I-state* ι_{k-1} and *new* observation y_k. When convenient, (53) will be shifted one stage forward to equivalently obtain $\iota_{k+1} = \phi(\iota_k, y_{k+1})$

Some generic, straightforward examples are now given.

Filter 5 (Sensor Feedback)
As a trivial special case, imagine a filter that maintains only the most recent observation. In this case $\mathcal{I} = Y$ and $\iota_k = \phi(\iota_{k-1}, y_k) = y_k$. Note that this filter did not even use the previous I-state ι_{k-1}. This implies that it does not even require the initial I-state ι_0, which has been left unspecified. ■

Filter 6 (Stage Counter)
Another simple example is to report the current stage. In this case, $\mathcal{I} = \mathbb{N} \cup \{0\}$, the set of nonnegative integers. We have $\iota_0 = 0$ and the filter is

$$\iota_k = \iota_{k-1} + 1. \tag{54}$$

■

Filter 7 (Simple State Estimator)
Suppose that $\mathcal{I} = X$ and the goal is to reconstruct the state from observations. To make the filter simple, suppose that $X = \mathbb{R}^2$ and a history-based sensor mapping is defined as

$$y_k = h(x_k, x_{k-1}) = x_k - x_{k-1}. \tag{55}$$

The initial I-state ι_0 is the initial state, $\iota_0 \in X$. The following filter perfectly recovers the state:

$$\iota_k = \iota_{k-1} + y_k. \tag{56}$$

By adding the observations, a telescoping sum is produced that results in $\iota_k = x_k$. Note that this filter refers back to the state space. Such reference is critical to most useful filters, and will be considered more carefully in Section 4.2.3. ■

[4] A more mathematically accurate name here would be *information set* because no topology is implied.

Note that any filter can be extended to form a mapping over all observation histories. Suppose that $\tilde{y}_k = (y_1, \ldots, y_k)$ is provided. Starting with ι_0, we can apply ϕ iteratively to obtain ι_k. This implies that any filter (53), once given an initial I-state ι_0, can be converted into a mapping $\kappa : \mathcal{I} \times \tilde{Y} \rightarrow \mathcal{I}$. This appears as

$$\iota_k = \kappa(\iota_0, \tilde{y}_k). \tag{57}$$

We could more generally consider proposing any I-space $\mathcal{I}$ and any mapping $\kappa : \mathcal{I} \times \tilde{Y} \rightarrow \mathcal{I}$; however, in this section we are only interested in ones that can be constructed incrementally by iterating (53).

4.2.3 Including motion models

We would like to use a model about the evolution of states to help construct more informed filters. Filter 7 referred to the state, but did not need any assumptions about how states change.

We can introduce a *state transition function*, $f : X \rightarrow X$, which correctly predicts x_{k+1} from x_k with the equation $x_{k+1} = f(x_k)$. Let X_k denote a given subset of X that means the set of possible states at stage k. We will sometimes write $X_k(...)$ to denote the set of possible states at stage k based on all information inside of the parentheses. If the current state is not known, but is restricted to some subset $X_k \subseteq X$, then we can apply f to every $x \in X_k$ to determine a *forward projection*:

$$X_{k+1}(X_k) = \{x_{k+1} \in X \mid x_k \in X_k \text{ and } x_{k+1} = f(x_k)\}. \tag{58}$$

For many problems, perfect predictability is too restrictive. Perhaps we only know a set of possible future states. In this case, a *nondeterministic state transition function* is obtained, which is of the form $F : X \rightarrow \text{pow}(X)$. The forward projection for this model becomes:

$$X_{k+1}(X_k) = \{x_{k+1} \in X \mid x_k \in X_k \text{ and } x_{k+1} \in F(x_k)\}. \tag{59}$$

Probabilistic models could also be introduced, to obtain a *probabilistic transition function* $p(x_{k+1}|x_k)$. The forward projection in this case becomes a marginalization: x_k is not actually given, but is replaced by some density $p(x_k)$. This yields:

$$p(x_{k+1}) = \sum_{x_k \in X} p(x_{k+1}|x_k)p(x_k). \tag{60}$$

It could be the case that our characterization of future states depends on some actions chosen by ourselves or other bodies in the environment. Suppose that these actions are known and are recorded in the same way as a sensor observation. In this case, let U be an *action space* and let $u_k \in U$ be an *action* applied at stage k. The particular u_k simply becomes a new parameter in the state transition functions. In the predictable case, we obtain $f : X \times U \rightarrow X$ and $x_{k+1} = f(x_k, u_k)$. In the nondeterministic case, we obtain $F : X \times U \rightarrow \text{pow}(X)$. In the probabilistic case, we obtain $p(x_{k+1}|x_k, u_k)$. These are standard models used in control theory. The forward projections presented above naturally extend to allow u_k to appear in the conditions.

It is assumed that one action is chosen at each stage. Let $\tilde{u}_k = (u_1, \ldots, u_k)$ be called an *action history*. These actions may be chosen externally, but the filter is able to obtain $\tilde{u}$, or the system we are designing may itself choose the actions to force desired behaviors to happen in the external, physical world. This is the subject of Section 5. In either case, the total information available at stage k for constructing a filter in some I-space $\mathcal{I}$ is ι_0, $\tilde{y}_k$, and $\tilde{u}_{k-1}$.

Taking the additional action information into account, the filter structure from (53) is extended to

$$\iota_k = \phi(\iota_{k-1}, u_{k-1}, y_k). \tag{61}$$

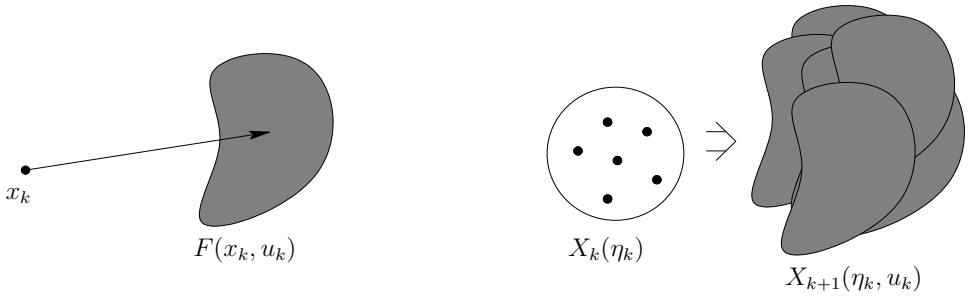


Figure 20: The first step in computing the nondeterministic I-state is to take the union of $F(x_k, u_k)$ over all possible $x_k \in X_k(\eta_k)$. It is the effect of applying an action when the current state is not precisely known.

Filter 8 (History I-Space Filter)
One special kind of I-space will now be defined. Let $\eta_k = (\tilde{y}_k, \tilde{u}_{k-1})$ be called the *history I-state* at stage k. It corresponds to all information available at stage k. Let $\mathcal{I}_{hist}$ be the set of all possible η_k for all possible $k \geq 1$. For $k = 1$, we obtain $\eta_1 = \tilde{y}_1$ because there is no u_0. A trivial filter can be defined over $\mathcal{I}_{hist}$ as

$$\eta_k = \phi(\eta_{k-1}, u_{k-1}, y_k). \tag{62}$$

In each iteration, the history I-state $\eta_{k-1} = (\tilde{u}_{k-1}, \tilde{y}_{k-1})$ is simply extended to include the new information u_{k-1} and y_k, which directly forms η_k. To initialize (62) properly, let $\eta_0 = \emptyset$ and define the history I-space to include $\emptyset \in \mathcal{I}_{hist}$. ■

4.2.4 Nondeterministic filters

Once a state transition function has been determined, a generic filter can be defined that keeps track of the set of possible states at every stage, given all available information. For this filter, we define $\mathcal{I}_{ndet} = \text{pow}(X)$, in which $\mathcal{I}_{ndet}$ is called the *nondeterministic I-space*. Using all available information $\eta_k = (\tilde{u}_{k-1}, \tilde{y}_{k-1})$ at stage k, we denote an I-state as $X_k(\eta_k)$, which is a subset of X. Thus, $X_k(\eta_k) \in \mathcal{I}_{ndet}$.

See Section 11.2.2 of [14] for a detailed derivation and presentation of the filter appearing here. The general form of the generic nondeterministic filter is

$$X_{k+1}(\eta_{k+1}) = \phi(X_k(\eta_k), u_k, y_{k+1}). \tag{63}$$

Let $X_1 \subseteq X$ denote the initial I-state. We thus interpret X_1 as ι_0, and $X_1 \in \mathcal{I}_{ndet}$. The filter uses the sensor mapping $h : X \to Y$ and state transition function $F : X \times U \to \text{pow}(X)$. After the first observation, the set of possible states is $X_1(y_1) = X_1 \cap h^{-1}(y_1)$. In words, this simply intersects the initial possible states with the preimage due to y_1.

Now suppose inductively that $X_k(\eta_k)$ has been given, as appearing in (63). First consider taking into account u_k. This yields:

$$X_{k+1}(\eta_k, u_k) = \bigcup_{x_k \in X_k(\eta_k)} F(x_k, u_k). \tag{64}$$

This can be considered as the set of all states that can be reached by starting from some state in $X_k(\eta_k)$ and applying an action $u_k \in U$. See Figure 20.

The next step is to take into account the observation y_{k+1}. This information alone indicates that x_{k+1} lies in the preimage $h^{-1}(y_{k+1})$. Therefore, an intersection is performed to obtain

$$X_{k+1}(\eta_{k+1}) = X_{k+1}(\eta_k, u_k, y_{k+1}) = X_{k+1}(\eta_k, u_k) \cap h^{-1}(y_{k+1}). \tag{65}$$

The completes the detailed specification of (63).

After starting with the initial subset of X, the nondeterministic I-states at any stage can be computed by iterating (64) and (65) as many times as necessary. Note, however, that this generic filter may have high complexity or might not even be computable. This depends on the particular structure of the problem and on how the I-states are encoded.

4.2.5 Probabilistic filters

A generic probabilistic (or Bayesian) filter can also be made. We define $\mathcal{I}_{prob}$ as the *probabilistic I-space*, which is the set of all probability density functions over X. It is assumed that such density functions exist, which might be possible for some state spaces. Using η_k, we denote an I-state as $p(x_k|\eta_k)$ and note that $p(x_k|\eta_k) \in \mathcal{I}_{prob}$. The filter takes the general form

$$p(x_{k+1}|\eta_{k+1}) = \phi(p(x_k|\eta_k), u_k, y_{k+1}), \tag{66}$$

which should be compared to the nondeterministic version (63).

To construct the filter, we use the sensor model $p(x_k|y_k)$ and the state transition model $p(x_{k+1}|x_k, u_k)$. The initial density, corresponding to ι_0, is a given prior $p(x_1)$. The filter presented here is derived in Section 11.2.3 of [14] using the same notation. Other sources include [5, 11, 19].

Assume for now that X is discrete. The filter (66) can be nicely expressed by first considering the effect of u_k, followed by y_{k+1}. Starting from $p(x_k|y_k)$ and using u_k in the state transition model, we obtain

$$\begin{aligned} p(x_{k+1}|\eta_k, u_k) &= \sum_{x_k \in X} p(x_{k+1}|x_k, u_k, \eta_k) p(x_k|\eta_k) \\ &= \sum_{x_k \in X} p(x_{k+1}|x_k, u_k) p(x_k|\eta_k). \end{aligned} \tag{67}$$

Taking this result and applying the sensor model on y_{k+1} yields

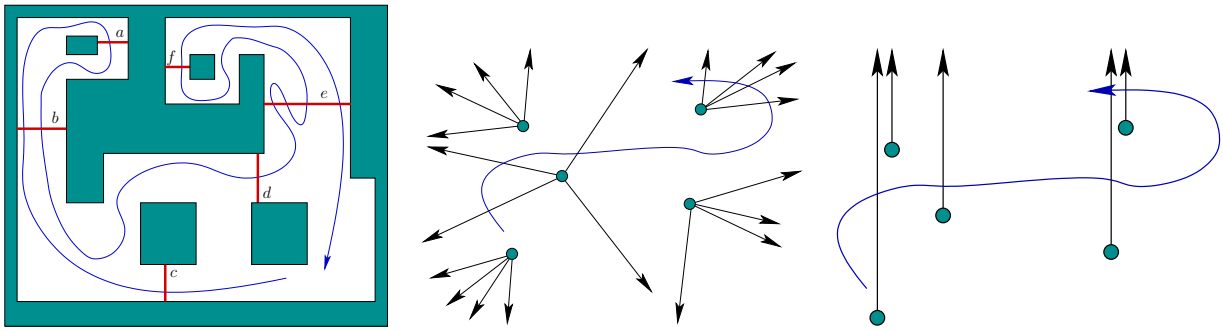
$$p(x_{k+1}|y_{k+1}, \eta_k, u_k) = \frac{p(y_{k+1}|x_{k+1}, \eta_k, u_k) p(x_{k+1}|\eta_k, u_k)}{\sum_{x_{k+1} \in X} p(y_{k+1}|x_{k+1}, \eta_k, u_k) p(x_{k+1}|\eta_k, u_k)}. \tag{68}$$

If X is not discrete, then the summations above are replaced with integrals.

Once again, there may be consderable computational challenges when implementing these filters. Many sampling-based techniques, such as particle filters, have been developed and used in practice to implement them. For their application in robotics, see [19].

Although not covered in detail here, one of the most famous and useful examples of a Bayesian filter is the *Kalman filter*. In this case, every probability density function is a Gaussian. Every $p(x_k|\eta_k)$ can therefore be specified by the mean μ and covariance Σ. The general structure of the Kalman filter is

$$(\mu_{k+1}, \Sigma_{k+1}) = \phi((\mu_k, \Sigma_k), u_k, y_{k+1}). \tag{69}$$



a. b. c.

Figure 21: (a) Imagine placing some beam-shaped detection sensors around an indoor environment. (b) Virtual beams could even arise by noticing when towers become aligned in an image while exploring a field. (c) Alternatively, virtual beams could arise from a robot using a compass: A virtual beam is crossed whenever a target is directly north.

4.3 Combinatorial Filters

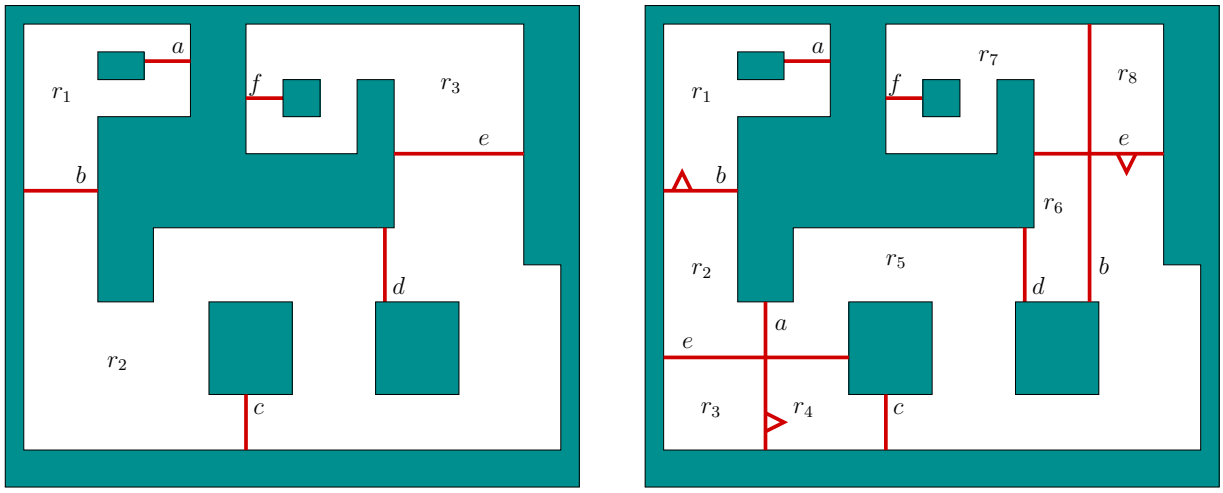
The filters described in Sections 4.2.4 and 4.2.5 maintain as much information as possible regarding possible states. The resulting I-spaces $\mathcal{I}_{ndet}$ and $\mathcal{I}_{prob}$ are too unwieldy in many situations. An alternative is to start with a particular task and try to determine what information can be discarded by the filter, while still maintaining task-critical information. This leads to a new family of filters based on designing small I-spaces that extract and maintain weak geometric, topological, or combinatorial information. They will generally be referred to as *combinatorial filters*.

4.3.1 Obstacles and beams

This section considers what information can be obtained from the knowledge that one or more moving bodies pass through detection beams. Consider the problem in Figure 21(a). An unpredictable point body moves inside of a given polygonal region E in the plane. The state space is $X = E \subset \mathbb{R}^2$. There are several static binary detectors, as in Model 14, and for each the detection region V is a line segment for which each endpoint connects to ∂E. Each detector is called a *beam* and is uniquely labeled a, b, c and so on. Rather than output 0 or 1 as in (10), we modify h so that each outputs 0 or its own label instead of 1. We can combine the beams into a single sensor that produces a label whenever the corresponding beam is crossed. If the body is not touching any beams, then the combined sensor continuously yields 0. An observation history $\tilde{y}$ will therefore be compressed to indicate the sequence of beams that were crossed over time. For example $\tilde{y} = cbabdeeefe$ for the state trajectory in Figure 21(a).

There are many ways to physically implement this model. We could, for example, use the safety beams shown in Figure 4(h). We could even obtain "virtual beams" through an entirely different physical sensor model. For example, Figure 21(b) shows 5 landmarks (imagine them as towers) in the plane. If a robot moves in the plane and uses a camera, it could detect when a pair of landmarks is perfectly aligned. This would correspond to having beams placed as shown in the figure. Alternatively, Figure 21(c) shows how virtual beams would appear if a robot has a compass and can determine precisely when a landmark is directly north of its position.

Now think about what can be inferred after receiving a sequence of beam crossings. Clearly, the sensing model is too weak to recover the state. For the example in Figure 22(a), suppose the



a. b.

Figure 22: (a) A simple example, which leads to three regions, r_1, r_2, and r_3. (b) Beams may be directional, may intersect, and may be indistinguishable.

beams divide E into three two-dimensional regions, r_1, r_2, r_3. Let $R = \{r_1, r_2, r_3\}$. For this simple example, if the initial region is given, then the new region can be determined after each beam is crossed. This assumes that the body never touches a beam without crossing it completely.

Filter 9 (Simple Region Filter)
Consider any environment $E \subseteq \mathbb{R}^2$ in which a finite collection of beams is arranged so that: 1) every beam either touches ∂E at each end or shoots off to infinity, 2) every beam is uniquely labeled, 3) no pair of beams intersects. Let R be the set of two-dimensional regions formed by taking the maximal connected regions that can be traversed by the body without crossing a beam (this was applied to generate the three regions in Figure 22(a)).

For this general problem, we can make a simple filter that keeps track of the current region, assuming it was known initially. Let $\mathcal{I} = R$ and specify $\iota_0 = r_0$ as the initial region. Using the filter template (53), we obtain $\phi : R \times Y \rightarrow R$ and

$$r_k = \phi(r_{k-1}, y_k). \tag{70}$$

The next region r_k can be easily determined once r_{k-1} and y_k (the most recent beam crossed) are given. ■

We now consider a more complicated problem, such as the one shown in Figure 22(b). Suppose that the initial region is not known. Furthermore, we will allow a much more complicated collection of beams. We still require all beams to have linear detection regions for which both ends reach ∂E. However, beams may or may not have each of the following three properties:

1. Beams may or may not be *distinguishable*. Two or more beams could produce the same label.

2. Beams may or may not be *disjoint*. In other words, a pair of beams may intersect in the interior of E.

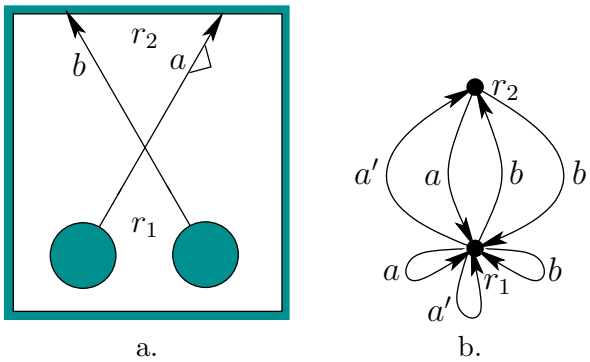


Figure 23: (a) An example that has two intersecting beams, one directed and one undirected. (b) The corresponding multigraph G.

3. Beams may or may not be *directed*. If a beam is directed, then we know which direction that body went when crossing. The beams in Figure 22(a) were undirected. For some beams in Figure 22(b), the beam's natural direction is shown with a triangular pointer placed on the beam. A body may traverse such beams in the natural direction or in the opposite direction. This information can be observed. For example, we may receive a if beam a is crossed in the natural direction, or receive a' for crossing in the opposite direction.

In spite of this complicated problem, the set R of regions can be defined as before. Each $r \in R$ is a connected two-dimensional region in $\mathbb{R}^2$ in which the body can travel without crossing a beam. We can define a multigraph G as follows. Every vertex in G corresponds to a region in $\mathbb{R}^2$. A directed edge is made from $r_1 \in R$ to $r_2 \in R$ if and only if the body can cross a single beam to go from r_1 to r_2. The beam label is placed on the edge. If the beam is undirected, then directed edges are made in both directions with the same label, say a for beam a. If the beam is directed, then the label is a in one direction and a' in the other. Note that a loop edge may be formed if a beam can be crossed while remaining in the same region. Figure 23 shows a simple example in which there are two beams and two regions.

Filter 10 (Nondeterministic Region Filter)
We now describe a filter that keeps track of the set of possible current regions. In this case, $\mathcal{I} = \text{pow}(R)$, and $\iota_0 = R_0$, a set of possible initial regions. The multigraph G is given (it can be computed from a description of E and the beam locations). A simple example is shown in Figure 23.

The method keeps track possible regions by *marking* the corresponding vertices of G. Initially, *mark* every vertex in R_0; all other vertices are *cleared*. The filter proceeds inductively, yielding $R_{k+1} = \phi(R_k, y_{k+1})$. At stage k, the marked vertices are precisely those corresponding R_k. Suppose that y_{k+1} is observed, which extends the sensor word by one observation. For each marked vertex, look for any outgoing edge labeled with y_{k+1}. For each one found, the destination vertex is marked. Any vertex that was marked at stage k but did not get marked in stage $k+1$ becomes *cleared*. Note that the total number of marked vertices may increase because from a single vertex there may be multiple edges that match y_{k+1}. Also, this approach works for the case of partially distinguishable beams because the match is based on the observation y_{k+1}, rather than the particular beam. The set of marked vertices yields R_{k+1}. ■

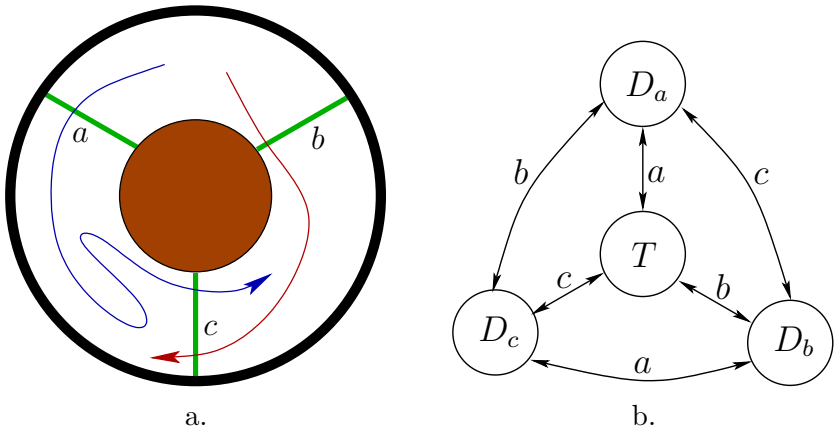


Figure 24: (a) Two bodies move in an environment consisting of three regions and three undirected beams. (b) A simple filter with only four I-states is sufficient to keep track of whether the two bodies are together in some region.

Filter 11 (Multiple Body Filter)
What if there are multiple bodies moving in E? Filter 10 could be generalized to keep track of which room each body might be in. Consider the example in Figure 24 in which two bodies move in an annulus-shaped environment E. The state space is $X = E^2 \subset \mathbb{R}^4$. Since there are two agents, there are 9 possible combinations: $(1,1)$, $(1,2)$, $(1,3)$, $(2,1)$, $(2,2)$, $(2,3)$, $(3,1)$, $(3,2)$, and $(3,3)$. In terms of the filter template (53), we let $\mathcal{I} = \text{pow}(R \times R)$. Initially, ι_0 is given, which is the set of possible initial room combinations. For example, someone may tell us that they are both in R_1, in which case $\iota_0 = \{(1,1)\}$.

For simplicity here, assume that beams cannot be crossed simultaneously by two bodies. To make the filter, we can directly extend the method of Filter 10. Let G^2 be the multigraph formed by taking the Cartesian product $G \times G$ in the sense that the vertices correspond to all ordered pairs of regions. Each edge in G^2 is formed if a transition from one ordered pair to another is possible after a single observation y_{k+1}. Once G^2 is formed, the method of propagating markers over the vertices, used in Filter 10, can be adapted and used here. In each iteration, the set of possible region pairs is maintained.

If there are n bodies in the environment, then the method can be extended by forming an n-fold Cartesian product of R to obtain $\mathcal{I}$ and a n-fold product of G to obtain G^n. If the number of bodies is unknown but bounded by n, then the disjoint union of I-spaces and G^i is formed, for each i from 0 to n. If the number is bodies is unbounded, then the method as described fails. ■

Of course the scheme described in Filter 11 is computationally prohibitive if there are many bodies and regions. Asymptotically, the number of vertices in G^n is exponential in n. There are many opportunities to dramatically reduce the filter complexity depending on the particular information of interest. This remains an open research problem: Determine what information can be maintained by a region filters that have much lower complexity. This is similar to the task of minimizing automata using Nerode equivalence classes in the classical theory of computation [10].

Filter 12 (Two-Bit Filter)
A simple example is presented in Figure 24. Two bodies move in a simple environment and we are interested only in one particular question: Are the two bodies together in a region, or are they

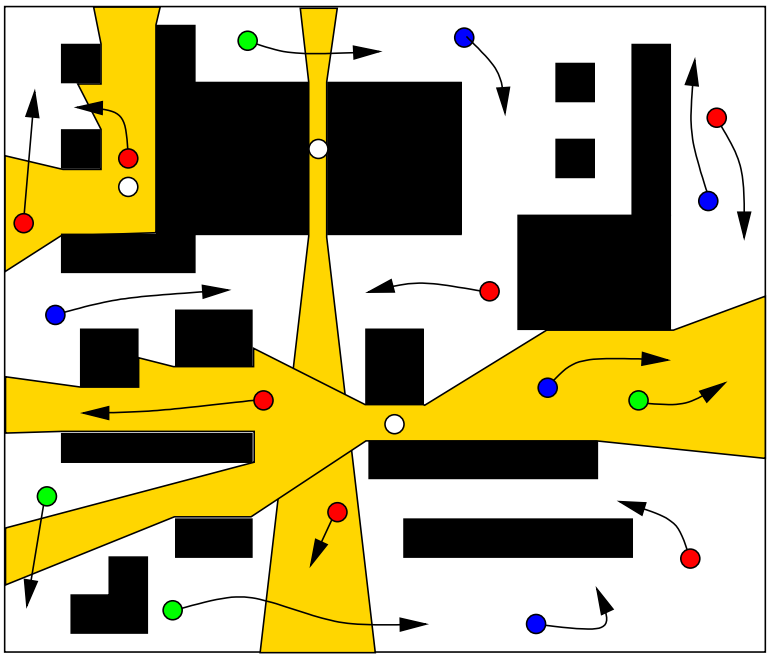


Figure 25: Imagine trying to keep track of bodies outside of the field of view–in the shadows.

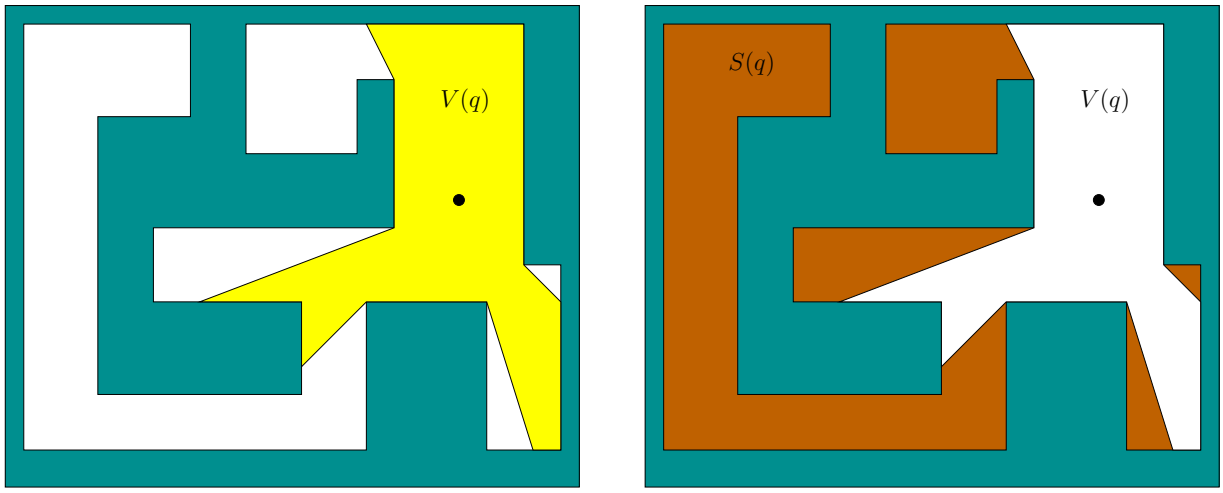
separated by a beam? It turns out that a particular filter can be designed for this question, and it is dramatically simpler than the generic one from Filter 11. The I-space is $\mathcal{I} = \{T, D_a, D_b, D_c\}$ and the filter is depicted in Figure 24(b). The T I-state means that the bodies are together. Each D_x I-state means that they are in neighboring regions and are separated by beam x. The automaton in Figure 24 defines the filter $\iota_k = \phi(\iota_{k-1}, y_k)$ by indicating the transitions caused by each specific y_k. If the initial I-state is given, then this simple "two-bit" filter can always, correctly answer the question about whether the two bodies are together (in the T I-state). ■

Filter 12 dramatically reduces the complexity of Filter 11 by finding the perfect I-space. How can this be done for other problems? Many exciting problems for future research remain!

In addition to trying to simplify the region filters, we might also want to make them more robust by tolerating disturbances in the beams. There may be false positives and false negatives, as described in Model 39. If we learned probabilistic models of these disturbances, then Filter 10 can be adapted to the probabilistic case. The I-space is the set of all probability density functions over R. This would follow the template of Section 4.2.5, resulting in a probabilistic filter that starts with a prior $p(r_1)$ and iteratively computes $p(r_{k+1}|\tilde{y}_{k+1})$ using only $p(r_k|\tilde{y}_k)$ and y_{k+1}. At each stage, the filter maintains a probability density over the possible regions that contain the body, based on the entire observation history.

4.3.2 Shadow information spaces

For these filters, imagine that several robots carry detection sensors (from Section 3.2.4) and move through a common environment that contains numerous moving point bodies, as shown in Figure 25. Suppose that q represents the configuration of all robots and $V(q) \subseteq E$ is their combined detection region in some environment E. The topic of interest is the part of E that is not visible to the sensors at a particular instant. We define $S(q) = E \setminus V(q)$ and call it the *shadow region*. See



a. b.

Figure 26: (a) The detection region for a single robot that carries a detection sensor. (b) The corresponding shadow region, which has 5 connected components.

Figure 26. Assume that if any body enters $V(q)$ it is detected by the sensors. Now think about portions of $S(q)$ within which bodies become trapped: Within any connected component of $S(q)$, a body cannot leave without being detected. We therefore consider a family of filters based on analyzing the connected components of the shadow region $S(q)$ and how they change over time.

A shadow region can generally be partitioned into a finite set of such components, called *shadow components*. As the robots move, the particular $S(q)$ gradually changes. The only changes of interest to us are the following combinatorial events:

1. **Disappear:** A shadow component vanishes, which eliminates a hiding place for the bodies.
2. **Appear:** A shadow component appears, which introduces a new hiding place for the bodies.
3. **Split:** A shadow component splits into multiple shadow components.
4. **Merge:** Multiple shadow components merge into one shadow component.

These are the only events that will concern us. To keep this tutorial simple, assume that: 1) no two events occur simultaneously, 2) a shadow component splits into at most two components, and 3) at most two components may merge into one. The events are illustrated in Figure 27.

Each time period over which no combinatorial events occur can be referred to as a stage. During a stage, if the particular shape or size of a shadow component varies, it will not be of interest. Each shadow component during a stage will be denoted s, and an entire set of n shadow components at stage k is denoted

$$S_k = \{s_1, s_2, \ldots, s_n\}. \tag{71}$$

We start with S_1 as the initial shadow components. Based on a combinatorial event, we obtain a transition from S_k to S_{k+1}. Typically, most components remain unchanged. For the disappear event, $S_{k+1} = S_k \setminus \{s\}$ for some component s. For an appear event, some new s is used to obtain $S_{k+1} = S_k \cup \{s\}$. For the split and merge events, three components become involved in the change. Let $M(s, s', s'')$ denote a *merge relation*, which is true when s and s' merge to form s''. Likewise, let $S(s, s', s'')$ denote a *split relation*, which is true when s splits to form s' and s''.

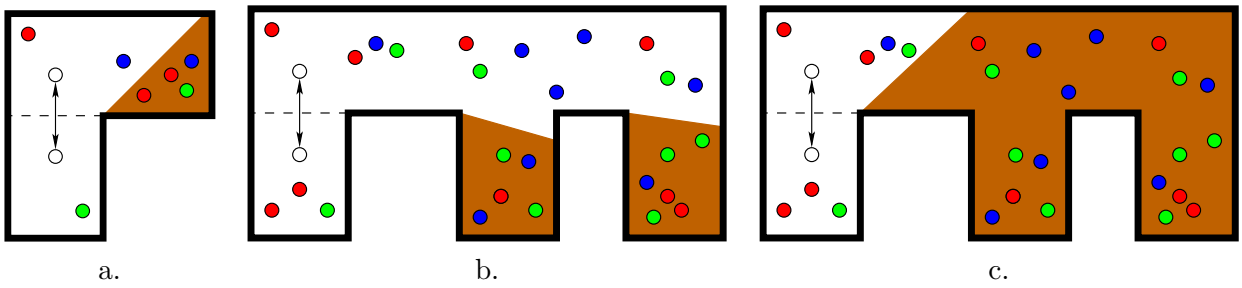


Figure 27: (a) A disappear or appear event occurs, depending on the direction of motion. (b,c) An split or merge event is illustrated, depending on the direction of motion.

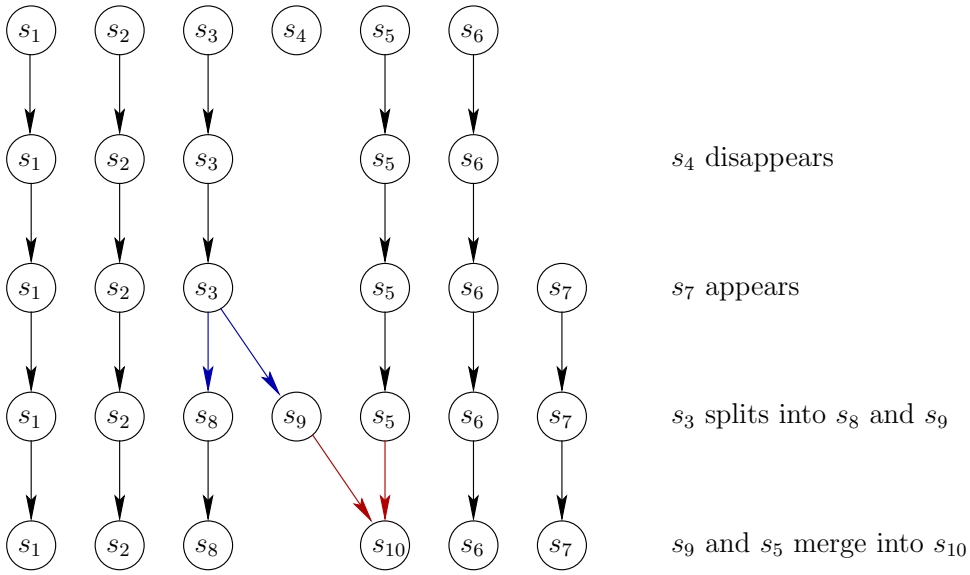


Figure 28: This example shows five stages, caused by each of the four combinatorial events.

Figure 28 shows a simple example.

We call the complete sequence $(S_1, \ldots, S_k)$, together with the specification of the split and merge relations, a *shadow sequence I-state*, and denote it by σ_k. Let $\mathcal{I}_{sseq}$ denote the *shadow sequence I-space*, which corresponds to all possible σ_k for a given problem. We assume that for whatever given problem, the mapping from $\mathcal{I}_{hist}$ to $\mathcal{I}_{sseq}$ exists and can be applied to yield σ_k from the observation and action histories.

We now define some filters that keep track of information over $\mathcal{I}_{sseq}$.

Filter 13 (Pursuit-Evasion Filter)
Suppose we would like to keep track of whether each shadow component is known not to contain any agents. This is useful, for example, to record the status of a pursuit strategy when solving a visibility-based pursuit-evasion problem [6, 9, 18].

The filter needs only to maintain a single bit per component:

- "0" means that there is definitely no body in s_1
- "1" means that could be a body in s_1

It will obtain all of the information it needs from the shadow sequence I-state. For each set S_k of shadow components, we associate a mapping $b_k : S_k \rightarrow \{0, 1\}$ which assigns a status bit to each shadow component. The initial mapping b_1 is given. If there might be an evader in a shadow component and we have no idea which one, then a common initial assignment is $b_1(s) = 1$ for every $s \in S_1$.

Now consider the operation of the filter when transitioning from σ_k to σ_{k+1}. Assume that b_k is already computed, and we have to determine b_{k+1}. The only additional information used comes from the split and merge relations from stage k to $k+1$. For any shadow component that appears, we assign $b_{k+1}(s) = 0$. For any that disappears, there is no assignment to make. The split and merge relations are utilized to handle the other two event types. If $S(s, s', s'')$, then $b_{k+1}(s') = b_k(s)$ and $b_{k+1}(s'') = b_k(s)$. If $M(s, s', s'')$, then $b_{k+1}(s) = 0$ if and only if $b_k(s') = 0$ and $b_k(s'') = 0$.

To gain an intuition for this filter, imagine that we are searching for a "lost" moving body which is outside of the field of view of the sensors. We do not know where it might be initially, which motivates assigning a value of 1 to every shadow component. As shadow events occur, we must update the bits so that every component $s \in S_{k+1}$ that *might* contain a body has $b_{k+1}(s) = 1$. Any component for which we are sure a body does not lie obtains the 0 status. If components merge, then we can assign 0 only if both original components are certain to not contain a body. ■

Filter 14 (Count Bounding Filter)
For this filter, we want to keep track of how many bodies there are in every shadow component. This simply replaces the each b_k function of Filter 13 with two functions $\ell_k : S_k \rightarrow C$ and $u_k : S_k \rightarrow C$ in which $C = \mathbb{N} \cup \{0, \infty\}$. The function ℓ_k is the *lower bound* on the number bodies in each shadow component. Likewise, u_k is the *upper bound*. As the names suggest, we require that $\ell(s) \leq u(s)$ for any shadow component s.

The first functions ℓ_1 and u_1 are assumed to be given. If no information is available for a shadow component s, we could assign $\ell_1(s) = 0$ and $u_1(s) = \infty$. Now consider the incremental operation of the filter. If a component s appears, it receives $\ell_{k+1}(s) = u_{k+1}(s) = 0$. If a component disappears, there is nothing to do. If $S(s, s', s'')$, then $\ell_{k+1}(s') = 0$, $\ell_{k+1}(s'') = 0$, $\ell_{k+1}(s') = u_k(s)$, and $u_{k+1}(s'') = u_k(s)$. If $M(s, s', s'')$, then $\ell_{k+1}(s'') = \ell_k(s) + \ell_k(s')$ and $u_{k+1}(s'') = u_k(s) + u_k(s')$.

So far the filter seems to lose important information. Let c, c', and c'' be the actual number of bodies in s, s', and s'', respectively. Even though these quantities are unknown, if $S(s, s', s'')$, then we must have $c = c' + c''$. Likewise, if $M(s, s', s'')$, then $c + c' = c''$. The filter should keep track of this information as well.

An interesting interpretation can be made. Let S'_k be the set of all shadow components observed up to stage k, even including ones that have disappeared. Let m be the number of elements in S'_k. For each i from 1 to m, an unknown integer represents the number of bodies in a particular shadow component. An m-dimensional vector v of integers can represent the number of bodies in every component. The set of all possible v is a subset of $\mathbb{Z}^m$ (each $\mathbb{Z}$ is the set of all integers). For component of v, upper and lower bounds are determined, which means that v is restricted to an axis-aligned rectangular subset of $\mathbb{Z}^m$. Furthermore, constraints of the form $c = c' + c''$ and $c + c' = c''$ due to splits and merges further restrict the set of possible v. Since all constraints are linear, the set of allowable v all lie in a polytope in $\mathbb{R}^m$. To ask particular questions about how many bodies are in a particular shadow component, taking all constraints into account, an integer linear program is obtained. Efficient solutions to this particular program are described in [21].■

Filter 15 (Team Maintenance Filter)
Filter 14 can be extended easily to the case of partially distinguishable bodies. Suppose that each

body belongs to a *team*. There could be only one team, which means they are all indistinguishable. If each is on a unique team, then they are all fully distinguishable. As an example in between, suppose they are classified as men or women. This yields two teams, and bodies are assumed to be indistinguishable inside of each team.

For this problem, we simply make a "vector version" of Filter 14, with one part for each team. For each shadow component and team, an upper and lower bound is maintained. All filter updates are handled for each team independently [21]. ■

An important complication can be easily considered in the filters above, and it is important to practical implementations. A body may pass in or out of the detection region $V(q)$, in which case we obtain additional information to be used in the filters. In this case, $V(q)$ may be treated as a special component for which the exact count on the number of bodies is known. Consider extending Filter 14. If we observe a body entering from shadow component s into $V(q)$, then we should decrement by one the upper and lower bounds associated with s. Likewise, if a body leaves $V(q)$ and enters some shadow component s, then we could increment its bounds. Such details are worked out in [21].

4.3.3 Gap navigation trees

We now develop a filter that is closely related to the shadow I-space. If a single robot is placed into a simply connected environment with a gap sensor, Model 24, then every gap corresponds directly to a shadow component. For example, when the robot is placed as in Figure 26(a), there are 5 gaps and each one corresponds to a shadow component.

Return to the shadow sequence I-state. We can similarly define a *gap sequence I-state*. The notions of combinatorial events and stages from Section 4.3.2 are used here. Rather than (71), we obtain

$$G_k = \{g_1, g_2, \ldots, g_n\}, \tag{72}$$

in which G_k is the set of gaps that exist over stage k, which is an interval of time during which no combinatorial events occur. We start with G_1, and have transitions from G_k to G_{k+1} in the same way as with shadow components: Gaps may disappear, appear, split, or merge. Furthermore, the merge and split relations are used here. This results in a *gap sequence I-state* γ_k and *gap sequence I-space*, $\mathcal{I}_{gseq}$.

As part of γ_k, we additionally assume that the split and merge relations can be inferred from the observation history. Here we obtain $M(g, g', g'')$ and $S(g, g', g'')$, defined in the same way as for shadows in Section 4.3.2. Note that this may or may not be possible in practice, depending in particular on how the gap sensor is implemented. In other words, to determine $M(g, g', g'')$, there needs to be sufficient information to infer that g'' was formed precisely as g and g' merged.

Filter 16 (Gap Navigation Tree)
We describe a filter over an I-space $\mathcal{I}_{trees}$ of rooted trees. Each tree captures some critical structure of the environment. Initially, for γ_1, the I-state $\iota_0 \in \mathcal{I}_{trees}$ consists of a root node that is connected to one child node for every gap in G_1. Every child vertex is labeled with its corresponding gap name.

The construction of the tree will now be described inductively. Assume that a tree ι_k has been computed by the filter. Using the new gaps G_{k+1}, a new tree ι_{k+1} is formed. Assuming that only one event occurs from stage k to $k+1$, it must either be an appear, appear, split, or merge. The filter is defined by describing the "surgery" that performed on ι_k to obtain ι_{k+1}. For an appear

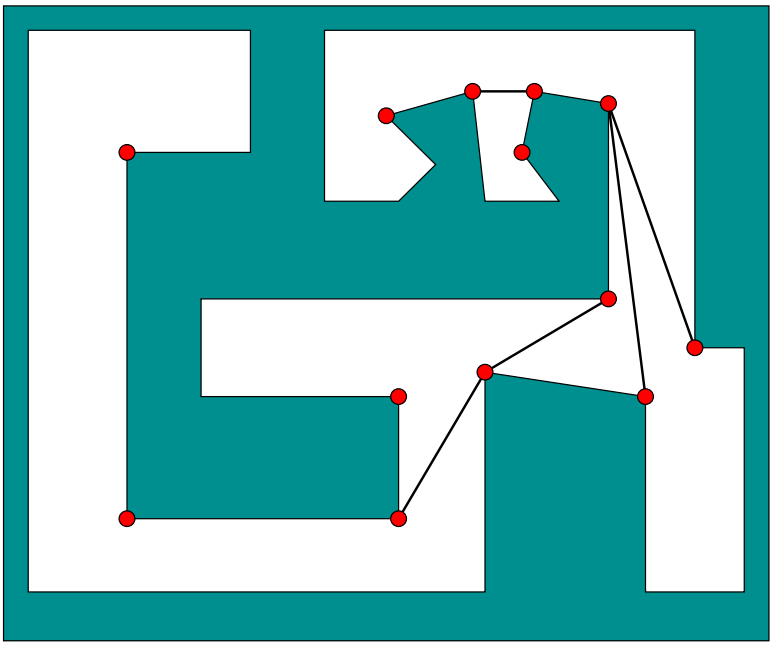
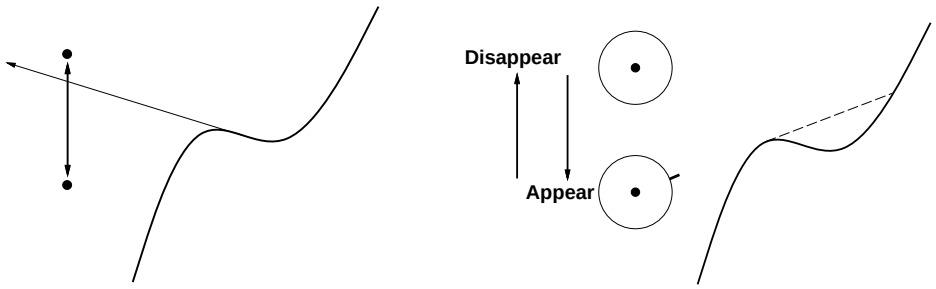


Figure 29: The shortest-path graph.

event, a new child is added from the root and given the label of the new gap. For a disappear event, the corresponding child is deleted from ι_k to obtain ι_{k+1}.

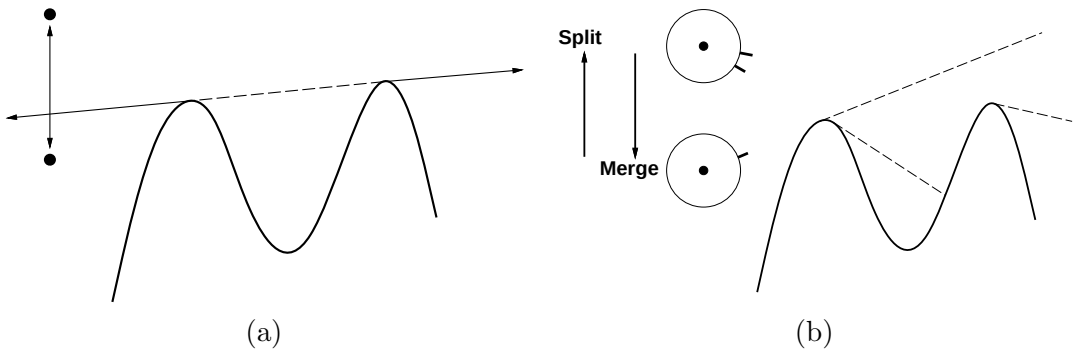
For a merge event $M(g, g', g'')$, consider the two subtrees corresponding to the two gaps g and g'. A new child of the root is inserted with label g''. The subtrees corresponding to g and g' are moved from the root and attached to g'' to indicate that these were merged into g''. In the case of a split event, the process works in reverse. For $S(g, g', g'')$, if there are already subtrees corresponding to g' and g'', then these are attached as children of the root when g is deleted. If there are no subtrees labeled g' and g'', then new child nodes corresponding to g' and g'' are attached to the root. More details appear in [14, 20]. ■

What is actually being recorded by the tree in Filter 16? The critical events are actually caused



(a) (b)

Figure 30: (a) The robot crosses a ray that extends from an inflectional tangent. (b) A gap appears or disappears from the gap sensor, depending on the direction.



(a) (b)

Figure 31: (a) The robot crosses a ray that extends from a bitangent. (b) Gaps split or merge, depending on the direction.

by generalized *inflections* and *bitangents*. Figures 30 and 31 show these cases. If the robot crosses an inflection ray, then an appear or disappear event occurs, depending on which direction it crosses. If the robot crosses a bitangent ray, then a split or merge event occurs, again depending on the direction. It turns out that the inner segments of bitangents are part of a well-known structure called the *shortest-path graph* (or sometimes called the *reduced visibility graph* [13]). An example is shown in Figure 29. Consider all pairs p, p' of points in a simply connected polygonal environment E. For each pair, there is a unique shortest path, which happens to be piecewise linear. Except for the initial and final path segments, all other segments must be either polygon edges or bitangent edges. Every nonsmooth point along the path corresponds to a reflex vertex along ∂E (interior angle greater than π).

If the robot explores enough of the environment, it is shown in [20] that the tree produced by Filter 16 encodes a portion of the shortest-path graph that is sufficient for optimal navigation to any place in E from the current robot location. This is depicted in Figure 32. Distance-optimal navigation can be performed using this filter, and is briefly described in Section 5.3.

5 Planning

Planning is a subject too vast to thoroughly cover here. Rather than give a complete introduction to planning, this tutorial briefly shows how to view planning from the perspective of I-spaces, using the filtering concepts from Section 4. For a more complete introduction to planning that fits with the notation of this tutorial, see [14]. Other helpful references include [5, 13].

This section views all planning problems as a search or optimization in an I-space, rather than the state space (unless the I-space happens to coincide with the state space). When there is substantial uncertainty due to sensors, planning occurs of an appropriate I-space. The challenging task is to design the system and resulting I-space so that task-solving plans can be efficiently computed and successfully executed. Ideally, we would like to accomplish tasks without having to fully reconstruct or estimate the full external state.

5.1 Plans and Execution

Let $\mathcal{I}$ be any I-space. Suppose that a state transition model from Section 4.2.3 is available that includes an action space U. Furthermore, suppose that a filter ϕ has been defined of the form (61).

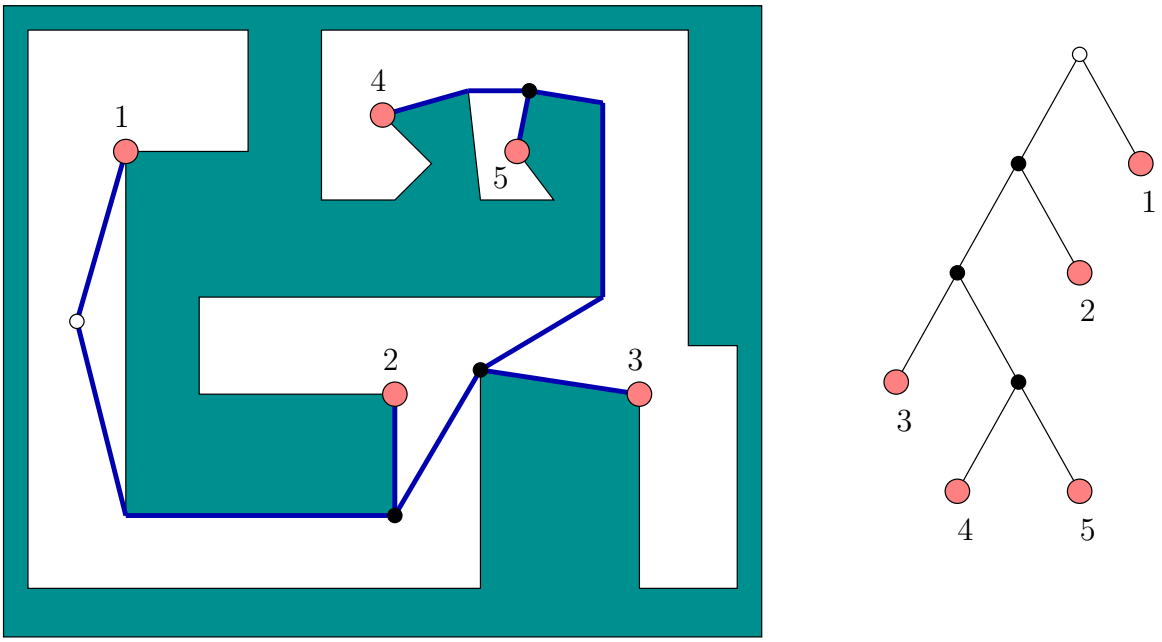


Figure 32: The gap navigation tree captures the structure of the shortest paths to the current robot location. The robot position is shown on the left. The tree on the right characterizes precisely how the shortest to the robot location are structured.

This produces a new I-state $\imath_k \in \mathcal{I}$ from the previous I-state $\imath_{k-1} \in \mathcal{I}$, previous action $u_{k-1} \in U$, and new observation $y_k \in Y$.

Generally, the planning problem is to chose each u_k so that some predetermined goal is achieved. Let $G \subset \mathcal{I}$ be called a *goal region* in the I-space. Starting from an initial I-state $\imath_0 \in \mathcal{I}$, what sequence of actions $u_1, u_2, \ldots$, will lead to some future I-state $\imath_k \in G$? Since future observations are usually unpredictable, it may be impossible to specify the appropriate action sequence in advance. Therefore, a more complete way to define the action selections is to define a *plan*:

$$\pi : \mathcal{I} \rightarrow U. \tag{73}$$

In this case, an action is determined from every I-state. During execution of the plan, the filter is executed, I-states are generated, and actions get automatically applied using π.

Using a filter ϕ, the execution of a plan can be expressed as

$$\imath_k = \phi(\imath_{k-1}, y_k, \pi(\imath_{k-1})), \tag{74}$$

which makes the filter no longer appear to depend on actions. The filter runs autonomously as the observations appear.

The main challenge is to construct a plan that achieves the desired goals. This topic is beyond the scope of the tutorial. The main issues that arise repeatedly in planning are:

- **Predictability:** Are the effects of actions predictable in the I-space? If not, then plans may be considered in which actions depends on future, unpredictable I-states. Otherwise, execution may look like a predictable path in the I-space.

- **Reachability:** Is the goal region even reachable from the initial I-state? In other words, do there even exist actions that will take us to the goal? Also, based on the I-space, does there

exist a plan that can reach the goal? If there is unpredictability, then we might additionally require that the goal is *guaranteed* to be reachable, over all possible disturbances.

- **Optimality:** If there are many plausible alternative plans, then what cost criteria should be formulated, and which plans are optimal with respect to them? Do optimal plans even exist?

- **Computability:** Given a description of the problem, can an algorithm be determined that automatically computes a useful plan? In many cases, a plan is designed by a clever human; however, automated planning is desirable in many circumstances. If a plan is theoretically computable, there are still practical issues such as algorithm complexity (running time and space) and implementation difficulty.

5.2 Important Generic Examples

Several examples are given here in which plans are described over various I-spaces that appeared in Section 4. They are widely used in robotics, planning, and control theory. Their description here, however, may look unusual because we based all of them on I-space concepts.

Example 2 (State Feedback Plans)
Suppose we have a filter that produces a reliable estimate of x_k using η_k. Assume the filter fits the general form (61), in which the I-space is $\mathcal{I} = X$ and ι_k is the estimate of x_k. In this case, a plan as expressed in (73) becomes $\pi : X \rightarrow U$.

Once the filter is running, there is no need to worry in the planning stage about uncertainty with regard to the current state. All sensing uncertainty is the problem of the filter. This is a standard approach throughout control theory and robotics: Produce a good state-estimating filter and then produce a plan or policy that uses state feedback. This enables the two issues of sensing and planning to be decoupled. Although a useful approach in many settings, we are most interested in this tutorial in ways to analyze both together, leading to simpler I-spaces, filters, and planning problems. ■

Example 3 (Open-Loop Plans)
In this case, we use Filter 6, which simply counts the number of stages. Recall the simple update equation (54) and I-space $\mathcal{I} = \mathbb{N}$. A plan is expressed as $\pi : \mathbb{N} \rightarrow U$. This can be interpreted as a specifying a sequence of actions:

$$\pi = (u_1, u_2, u_3, \ldots). \tag{75}$$

Such plans are often called *open loop* because no significant sensor observations are being utilized during execution. However, it is important to be careful, because some implicit time information is certainly being used: It is known that u_3 is being applied later than u_2, for example.

In (75), the actions appear to execute forever. In practice, the plan may terminate after a finite number of stages. See Chapter 2 of [14] for discussions of termination issues. ■

Example 4 (Sensor-Feedback Plans)
Now suppose that Filter 5 is applied, which produces only the most recent sensor observation y_k. In this case, a plan becomes $\pi : Y \rightarrow X$. It is wonderfully simple if such a plan can solve a useful task. For most tasks, however, some history of observations is needed. ■

Example 5 (Plans Over History I-Space)
Recall Filter 8, which simply reports the complete history of all observations and actions obtained so far. In this case, a plan is $\pi : \mathcal{I}_{hist} \rightarrow U$, which appears to be the most powerful plan possible. Every action depends on all possible information that can be utilized. There are several drawbacks, however. Since $\mathcal{I}_{hist}$ is large, it maybe be difficult or impossible to even represent an interesting plan. Furthermore, it may be hard to determine that the plan is in fact achieving a goal. Expressing the goal in this I-space may also be impractical because there is no direct connection to the state space. ■

5.3 Problem-Specific Examples

The examples of Section 5.2 used generic I-spaces that do not necessarily take into account problem-specific information to reduce the overall complexity of the planning problem. Using the concepts of this tutorial, we encourage the following overall process:

1. Design the system, which includes the environment, bodies, and sensors.
2. Define the models, which provide the state space X, the sensor mapping h, and the state transition function f.
3. Select an I-space $\mathcal{I}$ for which a filter ϕ can be practically computed.
4. Take the desired goal, expressed over X, and convert it into an expression over $\mathcal{I}$.
5. Compute a plan π over $\mathcal{I}$ that achieves the goal in terms of $\mathcal{I}$.

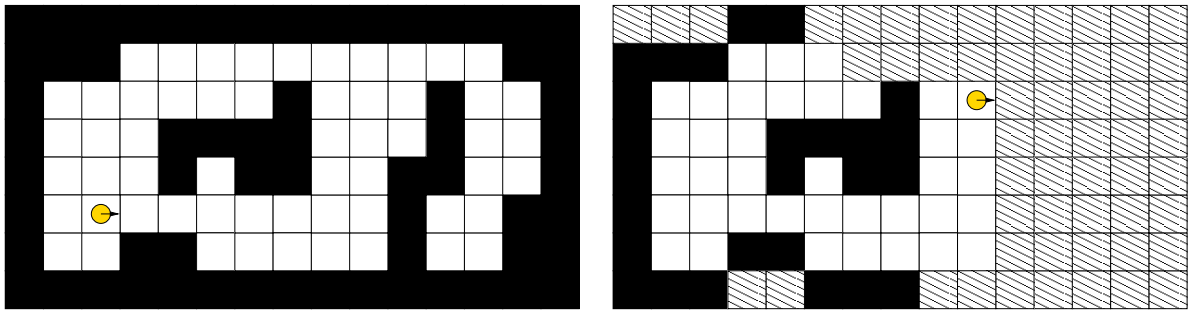
Ideally, these steps should all be taken into account together; otherwise, a poor choice in an earlier step could lead to artificially high complexity in later steps. Worse yet, a feasible solution might not even exist. Consider how Steps 4 and 5 may fail. Suppose that in Step 3, a simple I-space is designed so that each I-state is straightforward and efficient to compute. If we are not careful, then Step 4 could fail because it might be impossible to determine whether particular I-states achieve the goal. For example, Filter 6 simply keeps track of the current stage number. In most settings, this provides no relevant information about what has been achieved in the state space. Suppose that Step 4 is successful, and consider what could happen in Step 5. A nice filter could be designed with an easily expressed goal in $\mathcal{I}$; however, there might exist no plans that can achieve it. It could be the case that the problem is impossible to solve in the physical world under any circumstances, but a more common problem is that sufficient progress cannot be made in $\mathcal{I}$. This could mean, for example, that when actions are applied as in (74), the desired I-states are not even reachable.

Some examples that follow the approach are given here. Each is only briefly described, and we refer the reader to other sources for more details.

Example 6 (Maze Searching)
An example that was well ahead of its time and nicely follows the framework above is the maze searching algorithm by Blum and Kozen [2]. The robot is placed into a discrete environment, in which coordinates are described by a pair (i, j) of integers, and there are only four possible orientations (such as north, east, west, south). The state space is

$$X = \mathbb{Z} \times \mathbb{Z} \times D \times \mathcal{E}, \tag{76}$$

in which $\mathbb{Z} \times \mathbb{Z}$ is the set of all (i, j) positions, D is the set of 4 possible directions, and $\mathcal{E}$ is a set of environments. Every $E \in \mathcal{E}$ is a connected, bounded set of "white" tiles and all such possibilities



(a) (b)

Figure 33: (a) A discrete grid problem is made in which a robot is placed into a bounded, unknown environment. (b) An encoding of a partial map, obtained from some exploration. The hatched lines represent unknown tiles (neither white nor black).

are included in $\mathcal{E}$; an example appears in Figure 33(a). All other tiles are "black". Note that $\mathbb{Z} \times \mathbb{Z} \times D$ can be imagined as a discrete version of $SE(2)$, which was represented in Section 3.1.1 as $\mathbb{R} \times \mathbb{R} \times S^1$.

The robot is initially placed on a white tile, in an unknown environment, with an unknown orientation. The task is to move the robot so that every tile in E is visited. This strategy could be used to find an lost "treasure", which has been placed on an unknown tile. Only two actions are needed: 1) move FORWARD in the direction the robot is facing, and 2) ROTATE the robot 90 degrees counterclockwise. If the robot is facing a black tile and FORWARD is applied, then a sensor reports that it is blocked and the robot does not move.

Consider what kind of filters can be made for solving this task. The most straightforward one is for the robot to construct a partial map of E and maintain its position and orientation with respect to its map. A naive way so attempt this is to enumerate all possible $E \in \mathcal{E}$ that are consistent with the history I-state, and for each one, enumerate all possible $(i, j) \in \mathbb{Z} \times \mathbb{Z}$ and orientations in D. Such a filter would live in an I-space $\mathcal{I} = \text{pow}(\mathbb{Z} \times \mathbb{Z} \times D \times E)$, with each I-state being a subset of $\mathcal{I}$. An immediate problem is that every I-state describes a complicated, infinite set of possibilities.

A slightly more clever way to handle this is to compress the information into a single map, as shown in Figure 33(b). Rather than be forced to label every $(i, j) \in \mathbb{Z} \times \mathbb{Z}$ as "black" or "white", we can assign a third label, "unknown". Initially, the tile that contains the robot is "white" and all others are "unknown". As the robot is blocked by walls, some tiles become labeled "black". The result is a partial map that has a finite number of "white" and "black" tiles, with all other tiles being labeled "unknown". An I-state can be described as two finite sets W (white tiles) and B (black tiles), which are disjoint subsets of $\mathbb{Z} \times \mathbb{Z}$. Any tiles not included in W or B are assumed to be "unknown".

Now consider a successful search plan that uses this filter. For any "unknown" tile that is adjacent to a "white" tile, we attempt to move the robot onto it to determine how to label it. This process repeats until no more "unknown" tiles are reachable, which implies that the environment has been completely explored.

A far more interesting filter and plan are given in [2]. Their filter maintains I-states that use only logarithmic memory in terms of the number of tiles, whereas recording the entire map would use linear memory. They show that with very little space, not nearly enough to build a map, the environment can nevertheless be systematically searched. For this case, the I-state keeps track of only one coordinate (for example, in the north-south direction) and the orientation, expressed with

two bits. An entire plan is defined in [2] that is guaranteed to visit all white tiles using only this information. ■

Example 7 (Gap Navigation Trees)
Suppose that we would like to fully explore a continuous, planar environment, in a way that is similar to visiting all of the white tiles in Example 6. The map together with position and orientation is replaced in the current setting by a tree.

We start with Filter 16 and additionally introduce a motion model. The idea is that the robot can "chase" a gap, which means that an action u can be given so that the robot moves in the direction of the gap until a critical event occurs: The gap will either disappear or split [14, 20].

The first task is to make a plan that explores the whole environment. Recall the "unknown" labels from Example 6. The analogous issue here is that each leaf of the tree might contain a part of the environment that has not yet been explored. This means that if we chase that gap, we do not know whether it will disappear or split. We therefore augment Filter 16 so that it records a single bit of information for each leaf node. If we are certain that the gap would disappear, then we label the corresponding leaf node as "primitive". The search strategy is to chase any ancestor of a non-primitive leaf node, causing a sequence of splits, and terminating with a critical event for the leaf node. If it splits, then a descendent is chased. If it disappears, then a new non-primitive leaf node is selected and the plan continues. It is shown in [20] that in a simply connected environment, this plan always terminates with all leaves labeled as "primitive". At this point, the entire environment has been explored.

Once the entire environment has been explored, the tree can also be used for optimally moving the robot between locations. Since there are no coordinates in which to express the goals, the sensing model and filter are slightly augmented so that an object can be placed in E and is detected when it is visible from the robot. As the object disappears from view, it is simply recorded in the tree as if it were a disappearing gap. To return to the object optimally in terms of the shortest possible Euclidean distance traveled, the plan chases every ancestor of the object [20]. ■

Based on this tutorial, it should be clear that many more planning solutions can be developed by following this overall design process. There are many exciting opportunities for future research.

Acknowledgments

This work is supported in part by and NSF grant 0904501 (IIS robotics), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

References

[1] V. I. Arnold. *Mathematical Methods of Classical Mechanics, 2nd Ed.* Springer-Verlag, Berlin, 1989.

[2] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proceedings Annual Symposium on Foundations of Computer Science*, pages 132–142, 1978.

[3] F. Bullo and A. D. Lewis. *Geometric Control of Mechanical Systems.* Springer-Verlag, Berlin, 2004.

[4] C.-T. Chen. *Linear System Theory and Design.* Holt, Rinehart, and Winston, New York, 1984.

[5] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press, Cambridge, MA, 2005.

[6] B. Gerkey, S. Thrun, and G. Gordon. Clear the building: Pursuit-evasion with teams of robots. In *Proceedings AAAI National Conference on Artificial Intelligence*, 2004.

[7] B. Gfeller, M. Mihalak, S. Suri, E. Vicari, and P. Widmayer. Counting targets with mobile sensors in an unknown environment. In *ALGOSENSORS*, July 2007.

[8] L. Guibas. Sensing, tracking, and reasoning with relations. *IEEE Signal Processing Magazine*, 19(2):73–85, 2002.

[9] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. Visibility-based pursuit-evasion in a polygonal environment. *International Journal of Computational Geometry and Applications*, 9(5):471–494, 1999.

[10] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[11] P. R. Kumar and P. Varaiya. *Stochastic Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1986.

[12] A. Ladd, K. E. Bekris, A. Rudys, G. Marceau, L. E. Kavraki, and D. S. Wallach. Robotics-based location sensing using wireless ethernet. In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, pages 227–238, Atlanta, 2002.

[13] J.-C. Latombe. *Robot Motion Planning.* Kluwer, Boston, MA, 1991.

[14] S. M. LaValle. *Planning Algorithms.* Cambridge University Press, Cambridge, U.K., 2006. Also available at http://planning.cs.uiuc.edu/.

[15] L. Murphy and P. Newman. Using incomplete online metric maps for topological exploration with the gap navigation tree. In *Proceedings IEEE International Conference on Robotics & Automation*, 2008.

[16] S. Sastry. *Nonlinear Systems: Analysis, Stability, and Control.* Springer-Verlag, Berlin, 1999.

[17] M. W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control.* Wiley, New York, 2005.

[18] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888, October 1992.

[19] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics.* MIT Press, Cambridge, MA, 2005.

[20] B. Tovar, R Murrieta, and S. M. LaValle. Distance-optimal navigation in an unknown environment without sensing distances. *IEEE Transactions on Robotics*, 23(3):506–518, June 2007.

[21] J. Yu and S. M. LaValle. Tracking hidden agents through shadow information spaces. In *Proceedings IEEE International Conference on Robotics and Automation*, 2008.