Vision

So far we have considered applications of AI to game playing, which may require intelligence but probably have little bearing on behavior in the real world. This is *not* because of some intrinsic limitation on search and neural nets, but just because it seemed simpler to start with the kind of closed, well-defined world created by the rules of a game.

Getting a computer to thrive in the real world immediately raises the problem of getting information into the computer and allowing its behavior to change the things around it. Getting information in is the problem of *sensing;* behaving is the problem of *robotics.* To these we now turn. We will focus on computer vision. This is the sense modality most thoroughly studied, presumably because scientists are primates and have excellent vision themselves.

Biological vision systems are very good at finding moving objects in images. Indeed, the primitive visual systems of frogs and similar animals see very little except moving objects. Since the work of the psychologist J. J. Gibson, many have used the concept of *optical flow* to think about processing changing images. Imagine capturing the state of the image over a small time interval and watching where each part of the image moves. For example, imagine you're on a roller coaster, zooming down the track. Everything in the image is moving, except a point directly in front of you, the point you're headed toward. But if you're rounding a corner, then nothing in the image is stationary. At each point in the image, draw an arrow in the direction that piece of the image is moving, and make the arrow longer if that piece is moving faster. When I say "point," I mean mathematical point, so there is an infinite number of arrows; this is an abstract vector field. The flow field is useful in a couple of ways. If you are moving, the stationary point, if there is one, tells you what your immediate target is. Points nearer by are moving faster, so the field tells you the approximate distance of objects. If you are not moving, then the flow field tells you what is moving; if you're a frog, a small moving object is lunch. See figure 2.3a, showing the flow field a person might see as she exits a building.

It is possible to state a precise relationship between the way the image changes as you move across it spatially and the way it changes at a particular point over time. An example appears in figure 2.3b. Suppose

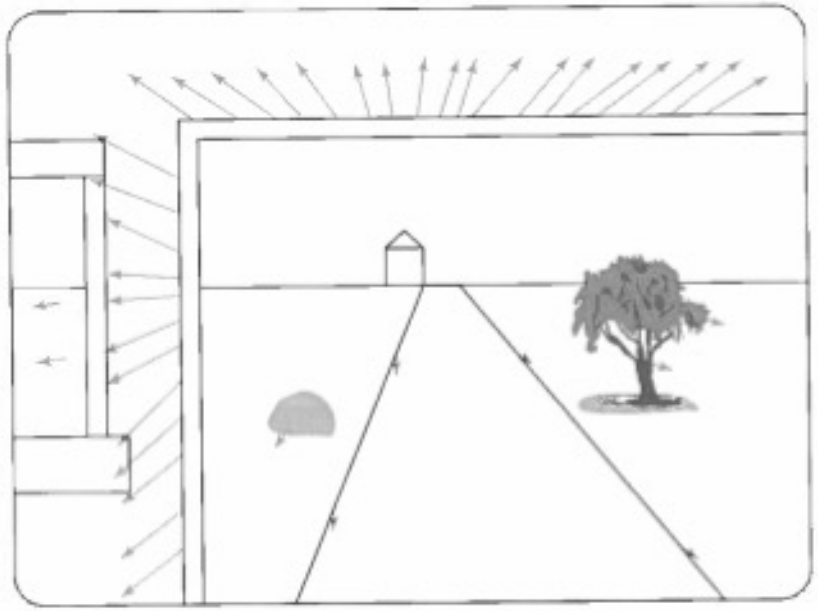


Figure 2.3a
Optical flow

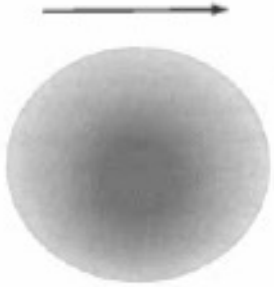
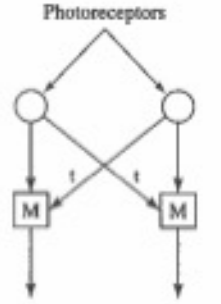


Figure 2.3b
Moving patch

an image consists of a dark patch moving over a white background. The patch is not uniform, but is a darker gray in the middle than at the edges. Consider what happens at a point P as the right edge of the patch moves over it. At one instant the brightness is $I(P, 0)$; after the patch has moved



"t" indicates short time delay

Figure 2.3c
Two motion detectors (from Sekuler et al., fig. 1C)

for a short length of time t, the brightness is reduced to $I(P, t)$. But if the dark patch doesn't change as it moves, it is possible to predict this change by looking at the piece that's coming. Letting v be the velocity of the patch, it will travel distance vt over the time interval t, so the brightness value that's at point vt to the left of P now will be over P after the interval. Calling this value $I(P - vt, 0)$, we can conclude

$$I(P, t) = I(P - vt, 0).$$

In other words, if we know v we can predict the change in brightness value at P by "peeking" to the left to see what's coming. However, what we want to do is the opposite: figure out the value of v by matching up the temporal brightness change at a single point in the image (the left side of the equation) with the spatial brightness change at a single point in time (the right side of the equation). The flow field is just the value of v at every point.[6]

Actually computing the field requires coping with the fact that we can't really list the value of the field at the infinite number of points in the image and points in time. One way to cope is to approximate by laying a grid over the image, and finding the value several times a second at every square of the grid. If the grid is fine enough, the resulting approximation will give you all the information you need. Not surprisingly, both the biological vision system and the digital vision system use such an approximation. The digital system represents the image and the flow field as arrays of numbers, and the biological system represents them by the

activity levels of arrays of cells in the retina of the eye and visual cortex of the brain. The computer's arrays are more regular, but that's a detail. The computer solves the equation numerically by performing repeated numerical operations at every grid square, or *pixel*, of the image. The brain of an animal solves it by allowing a neuron reacting to the brightness level at one place to compare it to the recent brightness stored at a neighboring neuron (Reichardt 1961, Sekuler et al. 1990). In figure 2.3c, two motion-detection cells, labeled "M," are connected to two nearby receptor cells with delays (labeled) in between. As a pattern moves across the two receptors, the M cells are able to compare the brightness at a point with the recent brightness at a nearby point. This is the biological system's way of comparing $I(P, t)$ with $I(P - vt, 0)$.

The key point is that both schemes rely on approximations of the underlying abstract mathematical objects. They work because the approximations are close enough. The details of the hardware, whether cells or silicon, are not important, provided they don't get in the way.

There are obvious differences between the two cases. The visual system must have many pairs of cells, one for each possible position, orientation, and velocity of a patch moving across the eye. A given motion detector can only tell you the motion at a certain point, in a certain direction, and at a certain speed.[7] To compute the whole field you have to provide a stupendous number of cells, each of which does a tiny bit of the computation. The digital computer may employ a stupendous number of silicon memory chips, but all the numbers pulled from those chips go through a small number of CPUs, each working blindingly fast. The digital computer exhibits what is called a *von Neumann architecture;*[8] the neural system has a *connectionist architecture,* because every interaction between two computations must be manifest in an explicit connection between the neural structures that perform them. But, as I have argued already, this is a difference that makes no difference, a matter of minor economic constraints on the materials at hand.

Another difference is that the computer implementation is the result of a deliberate attempt by vision researchers to solve an equation, whereas the biological system was designed by evolution—that is, not designed at all. So we must be cautious in claiming that the equation explains the structure of the biological system. Nonetheless, this kind of claim is not that unusual. Suppose that we are trying to explain photosynthesis, and we produce a thermodynamic analysis that shows that the maximum amount of energy that can be extracted from sunlight is X. Then we find a mechanism in plants that comes close to extracting that amount. The existence of the mechanism is explained by pointing out the need and the analysis. Of course, many gaps are left. We have to explain how the mechanism can have arisen from a series of mutations. We have to argue that the energy required to create and sustain the mechanism is less than X. We might even have to argue that the mechanism confers no other large benefit on the plant, whose existence might lead us to a different or complementary explanation of its evolution. This is a topic I will return to (in chapter 5).

Robotics

Vision and other sensor systems give an organism information about the world around it. The main use of this information is to guide the organism's motion through the world. The word "robot" is used to refer to a mechanical analogue: a creature that uses motors to control the motion of its body and limbs.

The problem of controlling motion using sensors is difficult for several reasons. One is that robots are more complex than traditional machines. An automobile engine contains many moving parts, but they're all moving along fixed trajectories. A piston goes back and forth in exactly the same way millions of times, causing the crankshaft to rotate the same way on each occasion. But consider an arm consisting of a shoulder, an elbow, a wrist, a hand, and several fingers. When the elbow and wrist are extended, the arm is the shape of a beam, and the shoulder must support the movement of a beam. If the arm is extended and holding a heavy object (such as a bag of garbage), the beam's physical characteristics change completely, but the shoulder muscles must still control its motion. When the elbow is bent, the same shoulder is the pivot for an object of an entirely different shape. When the elbow is bent and the hand is positioned near a table (to do some work on a watch, for instance), then the shoulder and elbow must move in such a way as to keep the hand near its position while allowing the fingers to manipulate things. So the general motion

problem is to move some parts of the arm-hand system in a desired motion, carrying loads of different sizes, while other parts obey constraints on their possible locations and velocities.

That's one bundle of difficulties. Another is that the world imposes constraints on the possible locations of the parts of the system. If you're working on an automobile engine, your arm must fit around the contours of the engine and its compartment.

But the worst problem is that it is not easy to obtain from the sensors the information required to control motion. We saw in the last section how hard it is to compute the optical flow. We are now asking for something much more precise: the exact three-dimensional structure of, say, an automobile engine. Extracting this information is possible, but it's not clear that you can get everything you need to guide an arm. There are several methods for extracing three-dimensional structure from an image. One is to use two eyes, and use the slight differences between the two images to compute depths of points; another is to shine a laser at different points of an object and measure how long it takes for the pulse to return. Obviously, the former method, *stereoscopic vision*, is used by animals (including us), and the latter, *laser rangefinding*, is not. There are less obvious techniques, such as noticing how texture and shadows change as a surface curves away from the eye. Optical flow gives depth information; objects that are closer tend to move faster across the eye. Using these techniques one can recover a *depth map*, which gives the distance from the eye of every pixel in the image. This is almost the same as a three-dimensional model of the object; it represents that shape from the point of view of the eye, and it's a straightforward mathematical operation to transform it to represent the shape from any other desired point of view.

Unfortunately, except for laser rangefinding, all of these techniques tend to yield inaccurate depth maps, and in any case a depth map is not a complete representation of the three-dimensional shape of the object being seen. It's almost one but not quite. By definition it says nothing about pieces of the object that are not visible, which includes at least half of it. (See figure 2.4, which shows the depth map generated from a 3/4 view of a bucket.) The eye can move to take in more of the object, but that requires knitting together depth maps taken from more than one point of view.

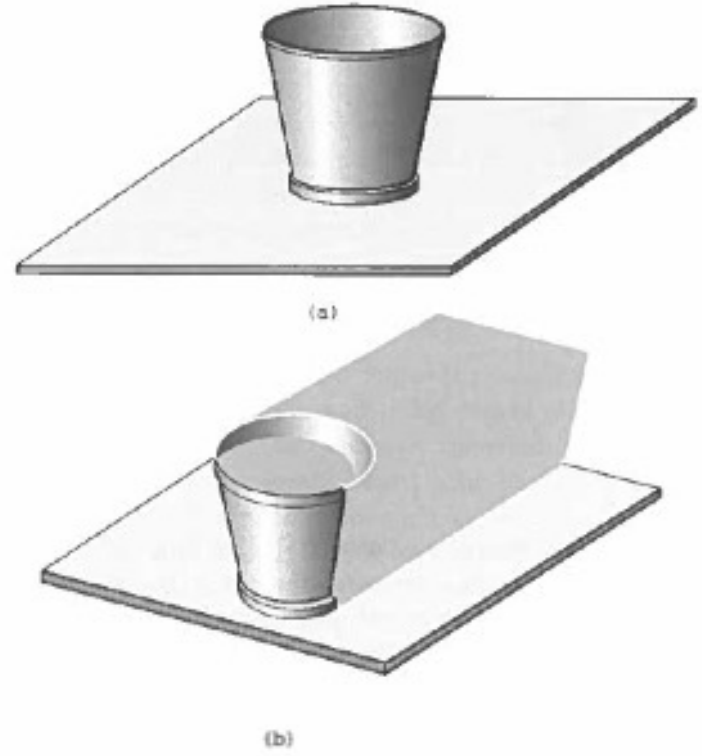


Figure 2.4
(a) Bucket. (b) Depth map from (a) (seen from the side)

It may be possible to solve these problems, but it may also be unnecessary. There isn't much hard evidence that the human visual-motor system constructs an accurate three-dimensional model of the objects in view before beginning to interact with them. As we interact with an object, we keep looking at it and are constantly acquiring new views of it. As we move our hands close to a part of the object, we can see our fingers getting closer to it without knowing exactly where either the object or the hand is. And, of course, we have our sense of touch to tell us when the hand (or the elbow) has bumped into something, whether we see the collision or not.

We can use the same approach in a robotic system, on a more modest scale. Suppose we want the robot to align a screwdriver with a screw. The screw is sticking up out of a surface, and the robot is supposed to

Figure 2.5
Screwdriver control

screw it down flush with the surface (figure 2.5). This requires lining up four points: the place where the screw enters the surface, the head of the screw, the blade of the screwdriver, and the handle of the screwdriver (points A, B, C, and D in figure 2.5). These four points are really defined in three-dimensional space, but all the robot knows is where they are in the image, a two-dimensional space. Under the expected range of motions, these four points will not change their appearance very much. So we can define them as visual patterns that the vision system should keep track of as the hand moves. Now our control algorithm can work as follows: Draw line segments $A'B'$ and $C'D'$, joining the centers of the visual patterns corresponding to points A and B, and C and D, respectively. If the two line segments are not collinear, move the screwdriver a little bit (the exact motion depends on the discrepancy in direction of $A'B'$ and $C'D'$, as well as how far the line containing $A'B'$ is from the line containing $C'D'$; the details are beyond the scope of this book). After the motion, take another picture. Presumably $A'B'$ is closer to the line containing $C'D'$, so the operation is repeated, until the two segments are collinear.

This procedure will not guarantee that the physical points A, B, C, and D are collinear. If they lie in a plane, and that plane is parallel to the line of sight, then they will look collinear no matter what their true alignment. Any variation in position in that plane will be invisible to the

camera. However, if we use *two* cameras looking from slightly different directions, then it is possible to align the screwdriver with the screw using only measurements in the image (Hager 1997), without ever having to infer the three-dimensional position of any of the objects involved.

A very different problem in robotics is the problem of map learning. Many animals do an amazingly good job of getting from one place to another. Bees find their way to flower beds, birds find their way south in the winter, and many mammals, including humans, roam a large territory without getting lost. The field of *map learning* has the goal of giving robots a similar set of skills.

The word "map" may have misleading connotations. One pictures a folded piece of paper with a schematic picture of the world on it. Such a picture might indeed be a necessary component of a map, but it omits an important factor, namely, what a point on the map looks like when you're there. Without this information, you will know which way to turn at every point on the trip, but you won't know when you're at that point. The only reason a road map is usable is that it shows the names of streets and roads, and, if you're lucky,[9] there are street signs displaying the same names. That's normally all you need to know about what the place looks like. In environments less structured than street networks, you must store more and different information about what the places look like when you're there.

Animals use a variety of techniques for figuring out where they are (Gallistel 1990; Muller et al. 1991). One technique is *dead reckoning*, in which the animal keeps track of every step and turn it takes after leaving home, adding up all those little motions to figure out what its net motion from home has been. It can then find its way back by turning in the direction of home and heading straight there. Robots can do this, too, but not as well. They do it by counting things like the number of times their wheels have turned. A typical robot changes its heading by having one wheel rotate more than the other, so a difference in the rotation counts for the two wheels translates into a change in the direction the robot is moving. Unfortunately, even though the robot can measure fairly small fractions of a wheel rotation, slight errors in the measurement translate into significant errors in the robot's estimate of its heading. Every time a robot turns it loses track of its orientation by a couple of degrees, so a few

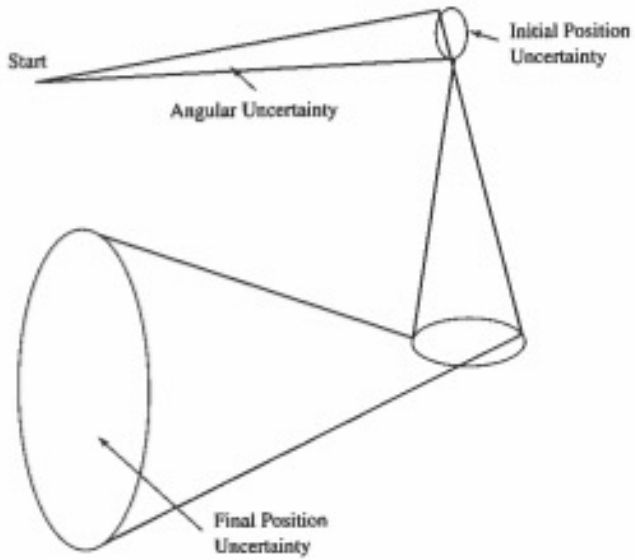


Figure 2.6
Positional uncertainty due to angular uncertainty

turns leave it rather disoriented. If the robot travels a long distance in an uncertain direction, a seemingly small direction uncertainty can translate into a large position uncertainty. In figure 2.6, the robot has no uncertainty at all about the distance it travels, and it knows the amount it turns to within 10 degrees. Nonetheless, after three turns and moves, its positional uncertainty (represented by an ellipse surrounding all the places where it might be) is enormous.

Hence a robot cannot rely entirely on dead reckoning to tell it where it is. It must visit locations more than once, and remember what they look like, or how they appear to nonvisual sensors. Then, if dead reckoning tells it that it might be at a previously visited location, it can compare its current appearance with the stored picture (or sonar recording) to verify.

Much of the research in this area uses sonar to measure the shape of the robot's immediate environment. Pulses of ultrasound are sent out and the time required for the pulse to bounce off something and return is recorded. The idea is that if there is an object in front of the sonar,

the time recorded tells the robot how far away that object is. Sonar has the advantage that it is cheap and simple. Many commercially available robots come equipped with rings of sonars so that signals can be sensed in all directions simultaneously. The quality of the information sonar returns is not very good, however. A sonar beam is 30 degrees wide, so getting a pulse back tells you there is an object but gives its direction to within only 30 degrees. The time it takes to record a pulse depends in complex ways on the composition and shape of obstacles.

Vision potentially provides a lot more information. A medium-sized black-and-white image can contain over 10,000 pixels, each encoded with a byte of information. A ring of 16 sonars provides about 16 bytes. A sonar recording of the environment is a low-resolution, one-dimensional row of dots. A picture is worth . . . , well, you get the idea. Many places look pretty much alike to a sonar. Every place looks different to a camera.

The problem is that the *same* place looks different to a camera. If you take a picture of a place, then come back and take another picture, they will look different. For one thing, some of the contents of the image will be different. The lighting may have changed. But the biggest changes will be due to the fact that the camera will not be in exactly the same place. Many of the lines will be at different angles. Objects will be foreshortened differently.

An algorithm developed by Hemant Tagare (Tagare et al. 1998) can cope with some of these problems. The idea is to treat the change in an image due to the change in camera position as a random perturbation of the image. If the camera hasn't moved too far, then there are limits to how the image can have changed. Let's assume that the camera is mounted horizontally and is located in the same plane now as when the picture was taken. Let's also assume that the lighting conditions haven't changed much. These assumptions are reasonable for indoor scenes taken from a robot rolling around on a flat, horizontal floor. Under these conditions, there are three ways the scene can be perturbed: (1) the camera axis won't be parallel to its orientation when the picture was originally taken; (2) the camera will be closer to or further from the objects in front of it; (3) some objects may have been added or removed; in particular, even if the same objects are present, some objects in the foreground may block different parts of the background image than they did originally (figure 2.7a,b).

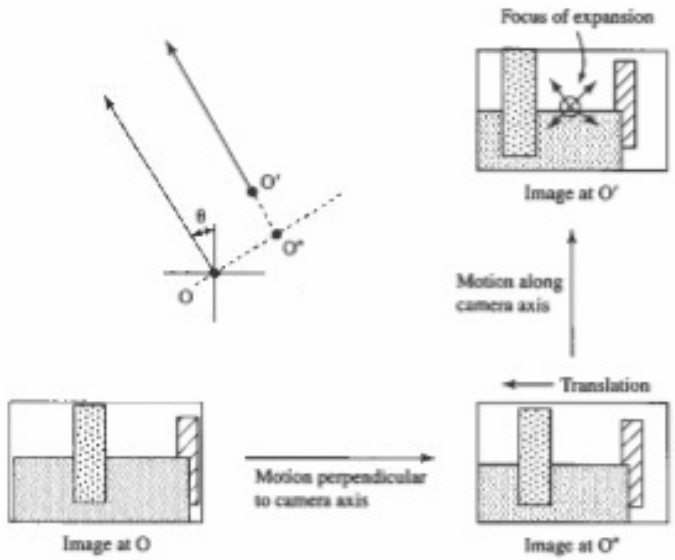


Figure 2.7a
Camera shift

The first kind of perturbation is handled by taking a wide-angled picture when a location is first visited. We don't use a wide-angled camera, but instead take a series of ordinary pictures and "glue" them together. If the resulting *panorama* spans 160 degrees, and the camera's field of view is 30 degrees, then the later picture can be taken with the axis rotated by up to 50 degrees (left or right) and it will match some part of the panorama.

The second kind of perturbation is more tricky. We don't know if the camera moved away from or toward the objects in the scene. Even if we did, the amount that a pixel would shift depends on the distance of the object whose surface it occurs on. In fact, this is a variant of the optical-flow problem we considered earlier, the difference being that we don't get to track pixels over time; if the camera had moved smoothly and directly from its old to its new position, we could compute the flow, but all we have is two snapshots. Nonetheless, for each pixel in one image, we can place constraints on the pixels it "could have flowed from" in the other. We call this set of pixels the *source region* for that pixel. If we maintain

Basically, correlating two images allowing for small perturbations

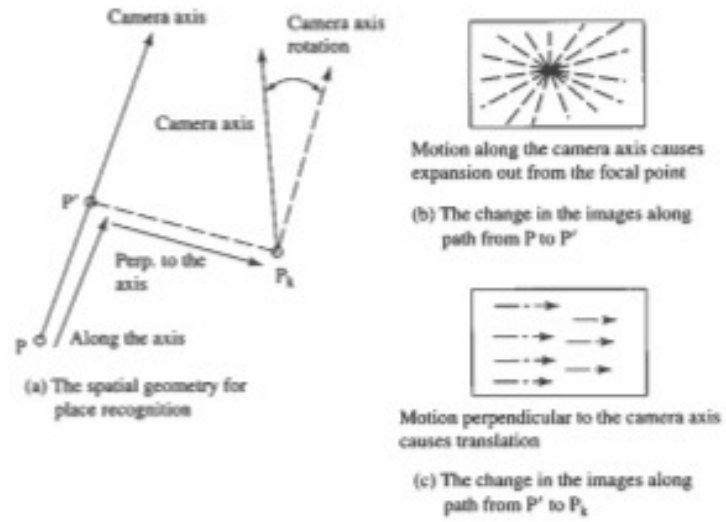


Figure 2.7b
Image perturbations due to camera shift

the idealized picture of the camera rotating and then moving in or out, and we correctly guess the rotation, then the pixel in the center of the image will be the same as in the old image; its source region is one pixel. As we look at pixels further from the center, the optical geometry of the camera means that the source regions get bigger. We can draw a diagram that gives, for some representative pixels, the source region in the other image (figure 2.8). For each point with a cross-hair over it, the enclosing quadrilateral with curved side gives the source region for that point.

The third kind of perturbation is unmodelable. Some pixels in the two images simply don't correspond to each other. We call these *outliers*. The more there are, the harder it is to see the similarity of the two images.

We put all this together by providing an estimate for the probability that the image can have arisen as a random perturbation of a slice of the panorama. For each pixel, we look at the pixels in its source region. If they are all of roughly the same gray level, then this pixel's brightness had better be close to it. If they exhibit a wide range of gray levels, then this pixel's brightness must fall in that range. The algorithm examines every

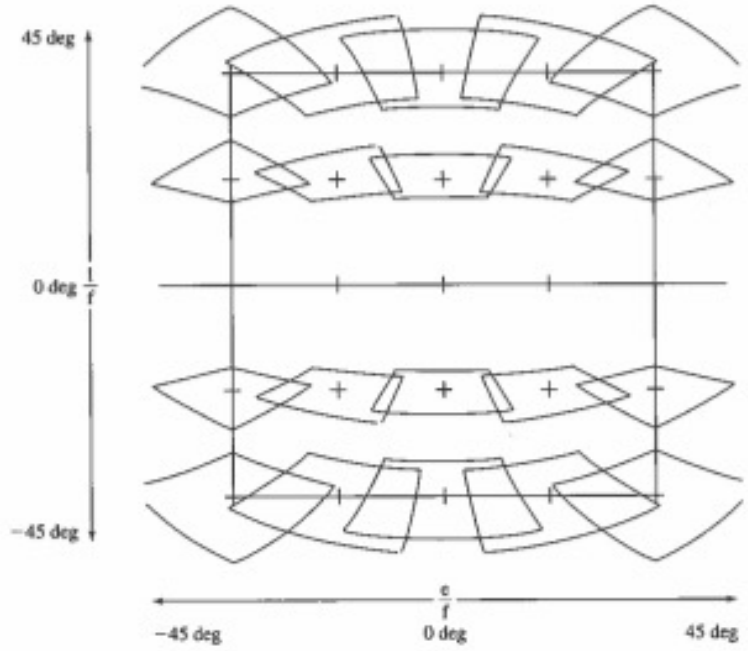


Figure 2.8
Pixel source regions (from Tagare and McDermott 1998)

pixel, and computes the square of the difference between it and the middle of the range as a proportion of the size of the range. If the difference is very high, then the algorithm classifies the pixel as an outlier. When every pixel has been examined, we arrive at two numbers that characterize how good the match is between two images: the outlier count, and the average difference between nonoutliers. If both these numbers are low, then the chances are good that the new image was taken from about the same place as the panorama.

This algorithm is not foolproof. There cannot be a foolproof algorithm in a world where some pairs of locations look very similar. But it often provides a strong clue to a robot that it has returned to a previously visited location. With enough such clues the robot can build a map.

There have been several approaches to robot map building (Kuipers and Byun 1991; Mataric 1990; Kunz et al. 1997; Engelson and McDermott

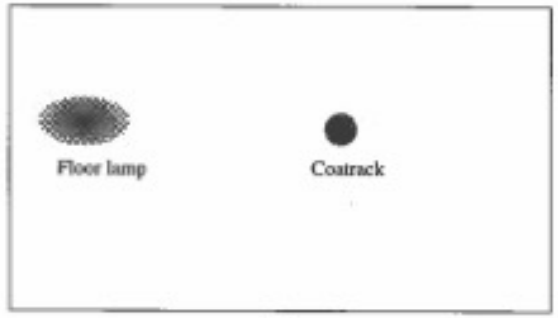
1992; Kriegman and Taylor 1996). A recent piece of work (Thrun et al. 1998) relies, like many recent research efforts in AI, on the theory of probability. That's because the inputs to the system yield "clues" to the shape of the world but not perfect information. It turns out that such clues can be modeled well using the language of *subjective probability*, in which propositions are assigned "degrees of belief" between 0 and 1. A proposition with subjective probability 1 is believed to be true with certainty; one with probability 0 is believed to be false with certainty; if you believe a proposition with degree p, you should be willing to bet p dollars against $1 - p$ dollars that the proposition is true (Savage 1954).

Thrun et al. represent a map as an assignment of probabilities of the presence of landmarks at every point in an area. A *landmark* is a recognizable entity; the case of there being no landmark at a point is treated as the presence of a special "null landmark" there. For example, in figure 2.9a, after some exploration a robot might assign probability 0.99 to the presence of the null landmark at every point except those near the two indicated points. At one of these the robot identified a coatrack, at the other a floor lamp. It is more certain of the exact location of the coatrack, so there is a smaller cluster of higher probabilities there. In figure 2.9b we see a more realistic example, in which the space consists of a set of corridors. Many areas cannot be visited at all, because they are blocked by walls and unopened doors. The landmarks in this case consist of panoramas that are matched at various points.

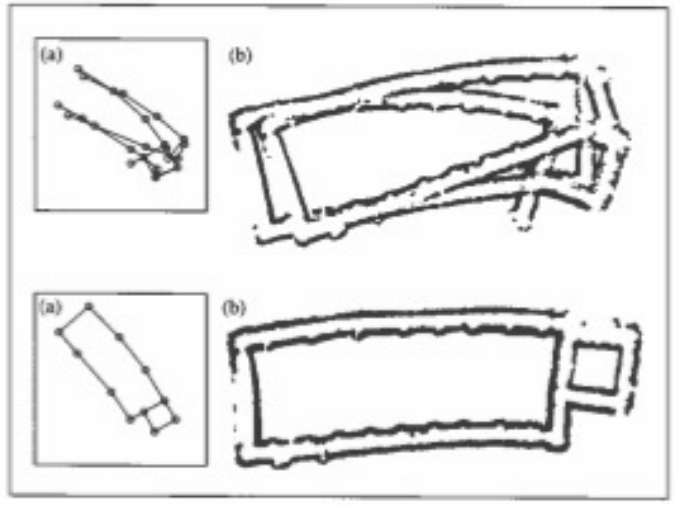
A robot can use a map like this in a couple of ways. One is for *route planning*. Given a destination and an origin, the robot can compute a path through the map that will allow it to get to the destination. Another is *location estimation*. As the robot moves, it can update its position by comparing what it sees with the map. Because the map is represented probabilistically, we can think of this as the problem of finding the location L such that the following quantity is maximized:

$$P(\textit{robot at } X \mid \textit{map } M \,\&\, \textit{observations } D),$$

where $P(A \mid B)$ is the *conditional probability* of proposition A given proposition B. X is a location, M is a probabilistic map, and D is a set of data from observations taken as the robot moves around, including dead-reckoning data and the landmarks seen along the way. Conditional



(a)



(b)

Figure 2.9
(a) Simple probabilistic map. (b) More complex map (from Thrun et al. 1998)

probability is a convenient way of representing changes in judgments of the plausibility of various beliefs as information is gathered. The probability that your ticket will win the lottery, $P(win)$, might be $\frac{1}{1,000,000}$; but after several digits have been selected, and they all match your ticket, you're justified in getting excited, especially if your ticket survives until there's just one digit left to select. At that point the relevant number is $P(win|all\ but\ one\ digit\ match)$, which is $\frac{1}{10}$.

Sometimes the robot is just given the map to begin with, but it can also use probability techniques to learn a map by wandering around and noticing landmarks. We pose this problem as that of finding a map M so as to maximize

$P(map\ M \mid observations\ D)$,

where now D is the set of observations collected as the robot wanders.

Conditional probability is related to unconditional probability by the following relationship

$$P(A \mid B) = P(A\&B)/P(B).$$

For instance, in the case of our lottery, the probability $P(all\ but\ one\ digit\ matched) = \frac{1}{100,000}$, so that

$$P(win \mid all\ but\ one\ digit\ matched) = \frac{\frac{1}{1,000,000}}{\frac{1}{100,00}} = \frac{1}{10}.$$

Given this relationship between conditional and unconditional probabilities, we can infer:

$$P(A\&B) = P(A \mid B)P(B) = P(B \mid A)P(A)$$

and hence

$$P(A \mid B) = \frac{(P(B \mid A)P(A))}{P(B)}.$$

This equation is known as Bayes's Theorem. It is very handy when you want to compute $P(A \mid B)$ and you happen to know $P(B \mid A)$. In our map example we can apply Bayes's Theorem to derive this:

$$P(map\ M \mid observations\ D) = \frac{P(D \mid M)P(M)}{P(D)}$$

The term $P(D \mid M)$ is the probability of observations given the map. Because the map specifies more or less where everything is, it turns out that this is not too hard to compute; it corresponds to asking, what would I see if I stood at a given point in the map and looked in a certain direction? $P(M)$ is the *prior probability* of a map. Some maps might be considered less plausible than others even if you had no data. For instance, you might not know where an acquaintance's house is, but you might rate it as unlikely that the house is in the river (although not impossible). $P(D)$ is the probability, over all possible maps, of the data that we happened to

observe. These two quantities may sound difficult to compute, but we can get around the difficulties. The exact form of $P(M)$ is not too important. Perhaps some simple rule would suffice that assigns higher probability to maps with landmarks spaced at a certain ideal distance. $P(D)$ doesn't depend on M at all, so if we want the M that maximizes $P(M \mid D)$ and don't care about the exact *value* of $P(M \mid D)$, we can just leave $P(D)$ out. (I have omittted many technical details.)

Thrun et al. tested their algorithm by moving the robot around by hand for fifteen minutes, telling it where various landmarks were. They then ran their algorithm to find the map that maximized the probability of making the observations that were actually made. The map that resulted was accurate to within inches.