EXERCISES 26

ROBOTICS

26.1 Robots

[[need exercises]]

26.2 Robot Hardware

[[need exercises]]

26.3 What kind of problem is robotics solving?

[[need exercises]]

26.4 Robotic Perception

Exercise 26.MCLB

Monte Carlo localization is *biased* for any finite sample size—i.e., the expected value of the location computed by the algorithm differs from the true expected value—because of the way particle filtering works. In this question, you are asked to quantify this bias.

To simplify, consider a world with four possible robot locations: $X = \{x_1, x_2, x_3, x_4\}$. Initially, we draw $N \geq 1$ samples uniformly from among those locations. As usual, it is perfectly acceptable if more than one sample is generated for any of the locations X. Let Z be a Boolean sensor variable characterized by the following conditional probabilities:

$$\begin{aligned}
P(z \mid x_1) &= 0.8 & \qquad P(\neg z \mid x_1) &= 0.2 \\
P(z \mid x_2) &= 0.4 & \qquad P(\neg z \mid x_2) &= 0.6 \\
P(z \mid x_3) &= 0.1 & \qquad P(\neg z \mid x_3) &= 0.9 \\
P(z \mid x_4) &= 0.1 & \qquad P(\neg z \mid x_4) &= 0.9\,.
\end{aligned}$$

MCL uses these probabilities to generate particle weights, which are subsequently normalized and used in the resampling process. For simplicity, let us assume we generate only one new sample in the resampling process, regardless of N. This sample might correspond to any of the four locations in X. Thus, the sampling process defines a probability distribution over X.

a. What is the resulting probability distribution over X for this new sample? Answer this question separately for $N = 1, \ldots, 10$, and for $N = \infty$.

b. The difference between two probability distributions P and Q can be measured by the KL divergence, which is defined as

$$KL(P,Q) = \sum_i P(x_i) \log \frac{P(x_i)}{Q(x_i)} .$$

What are the KL divergences between the distributions in (a) and the true posterior?

c. What modification of the problem formulation (not the algorithm!) would guarantee that the specific estimator above is unbiased even for finite values of N? Provide at least two such modifications (each of which should be sufficient).

To answer this question, consider all possibilities for the initial samples before and after resampling. This can be done because there are only finitely many states. The result for $N = \infty$ is simply the posterior, calculated using Bayes rule.

```
int
main(int argc, char *argv[])
{
  // parse command line argument
  if (argc != 3){
    cerr << "Usage: " << argv[0] << " <number of samples>"
 << " <number of states>" << endl;
    exit(0);
  }

  int numSamples = atoi(argv[1]);
  int numStates = atoi(argv[2]);
  cerr << "number of samples: " << numSamples << endl
       << "number of states:  " << numStates  << endl;
  assert(numSamples >= 1);
  assert(numStates >= 1);

  // generate counter
  int samples[numSamples];
  for (int i = 0; i < numSamples; i++)
    samples[i] = 0;

  // set up probability tables
  assert(numStates == 4); // presently defined for 4 states
  double condProbOfZ[4] = {0.8, 0.4, 0.1, 0.1};
  double posteriorProb[numStates];
  for (int i = 0; i < numStates; i++)
    posteriorProb[i] = 0.0;
  double eventProb = 1.0 / pow(numStates, numSamples);

  //loop through all possibilities
  for (int done = 0; !done; ){

    // compute importance weights (is probability distribution)
    double weight[numSamples], totalWeight = 0.0;
    for (int i = 0; i < numSamples; i++)
      totalWeight += weight[i] = condProbOfZ[samples[i]];
    // normalize them
    for (int i = 0; i < numSamples; i++)
      weight[i] /= totalWeight;

    // calculate contribution to posterior probability
    for (int i = 0; i < numSamples; i++)
      posteriorProb[samples[i]] += eventProb * weight[i];
```

(a)

```
    // increment counter
    for (int i = 0; i < numSamples && i != -1;){
      samples[i]++;
      if (samples[i] >= numStates)
samples[i++] = 0;
      else
i = -1;
      if (i == numSamples)
done = 1;
    }
  }

  // print result
  cout << "Result:  ";
  for (int i = 0; i < numStates; i++)
    cout << " " << posteriorProb[i];
  cout << endl;


  // calculate asymptotic expectation
  double totalWeight = 0.0;
  for (int i = 0; i < numStates; i++)
    totalWeight += condProbOfZ[i];

  cout << "Unbiased:";
  for (int i = 0; i < numStates; i++)
    cout << " " << condProbOfZ[i] / totalWeight;
  cout << endl;

  // calculate KL divergence
  double kl = 0.0;
  for (int i = 0; i < numStates; i++)
    kl += posteriorProb[i] * (log(posteriorProb[i]) -
      log(condProbOfZ[i] / totalWeight));
  cout << "KL divergence: " << kl << endl;
}
```

(b)

Figure S26.1 Code to calculate answer to exercise 25.1.

a. The program (correctly) calculates the following posterior distributions for the four states, as a function of the number of samples N. Note that for $N = 1$, the measurement is ignored entirely! The correct posterior for $N = \infty$ is calculated using Bayes rule.

N	$p(\text{sample at } s_1)$	$p(\text{sample at } s_2)$	$p(\text{sample at } s_3)$	$p(\text{sample at } s_4)$
$N=1$	0.25	0.25	0.25	0.25
$N=2$	0.368056	0.304167	0.163889	0.163889
$N=3$	0.430182	0.314463	0.127677	0.127677
$N=4$	0.466106	0.314147	0.109874	0.109874
$N=5$	0.488602	0.311471	0.0999636	0.0999636
$N=6$	0.503652	0.308591	0.0938788	0.0938788
$N=7$	0.514279	0.306032	0.0898447	0.0898447
$N=8$	0.522118	0.303872	0.0870047	0.0870047
$N=9$	0.528112	0.30207	0.0849091	0.0849091
$N=10$	0.532829	0.300562	0.0833042	0.0833042
$N=\infty$	0.571429	0.285714	0.0714286	0.0714286

b. Plugging the posterior for $N=\infty$ into the definition of the KL Divergence gives us:

N	$KL(\hat{p}, p)$	N	$KL(\hat{p}, p)$
$N=1$	0.386329	$N=7$	0.00804982
$N=2$	0.129343	$N=8$	0.00593024
$N=3$	0.056319	$N=9$	0.00454205
$N=4$	0.029475	$N=10$	0.00358663
$N=5$	0.0175705	$N=\infty$	0

c. The proof for $N=1$ is trivial, since the re-weighting ignores the measurement probability entirely. Therefore, the probability for generating a sample in any of the locations in S is given by the initial distribution, which is uniform.

For $N=2$, a proof is easily obtained by considering all $2^4=16$ ways in which initial samples are generated:

number	samples		probability of sample set	$p(z\|s)$ for each sample		weights for each sample		probability of resampling for each location in S			
1	0	0	$\frac{1}{16}$	$\frac{4}{5}$	$\frac{4}{5}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{16}$	0	0	0
2	0	1	$\frac{1}{16}$	$\frac{2}{5}$	$\frac{4}{5}$	$\frac{3}{9}$	$\frac{6}{9}$	$\frac{1}{24}$	$\frac{1}{48}$	0	0
3	0	2	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{4}{5}$	$\frac{1}{9}$	$\frac{8}{9}$	$\frac{1}{18}$	0	$\frac{1}{144}$	0
4	0	3	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{4}{5}$	$\frac{1}{9}$	$\frac{8}{9}$	$\frac{1}{18}$	0	0	$\frac{1}{144}$
5	1	0	$\frac{1}{16}$	$\frac{4}{5}$	$\frac{2}{5}$	$\frac{6}{9}$	$\frac{3}{9}$	$\frac{1}{24}$	$\frac{1}{48}$	0	0
6	1	1	$\frac{1}{16}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{16}$	0	0
7	1	2	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{2}{5}$	$\frac{1}{5}$	$\frac{4}{5}$	0	$\frac{1}{20}$	$\frac{1}{80}$	0
8	1	3	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{2}{5}$	$\frac{1}{5}$	$\frac{4}{5}$	0	$\frac{1}{20}$	0	$\frac{1}{80}$
9	2	0	$\frac{1}{16}$	$\frac{4}{5}$	$\frac{1}{10}$	$\frac{8}{9}$	$\frac{1}{9}$	$\frac{1}{18}$	0	$\frac{1}{144}$	0
10	2	1	$\frac{1}{16}$	$\frac{2}{5}$	$\frac{1}{10}$	$\frac{4}{5}$	$\frac{1}{5}$	0	$\frac{1}{20}$	$\frac{1}{80}$	0
11	2	2	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	$\frac{1}{16}$	0
12	2	3	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	$\frac{1}{32}$	$\frac{1}{32}$
13	3	0	$\frac{1}{16}$	$\frac{4}{5}$	$\frac{1}{10}$	$\frac{8}{9}$	$\frac{1}{9}$	$\frac{1}{18}$	0	0	$\frac{1}{144}$
14	3	1	$\frac{1}{16}$	$\frac{2}{5}$	$\frac{1}{10}$	$\frac{4}{5}$	$\frac{1}{5}$	0	$\frac{1}{20}$	0	$\frac{1}{80}$
15	3	2	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	$\frac{1}{32}$	$\frac{1}{32}$
16	3	3	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	$\frac{1}{16}$
sum of all probabilities								$\frac{53}{144}$	$\frac{73}{240}$	$\frac{59}{360}$	$\frac{59}{360}$

A quick check should convince you that these numbers are the same as above. Placing this into the definition of the KL divergence with the correct posterior distribution, gives us 0.129343.

For $N = \infty$ we know that the sampler is unbiased. Hence, the probability of generating a sample is the same as the posterior distribution calculated by Bayes filters. Those are given above as well.

d. Here are two possible modifications. First, if the initial robot location is known with absolute certainty, the sampler above will always be unbiased. Second, if the sensor measurement z is equally likely for all states, that is $p(z|s_1) = p(z|s_2) = p(z|s_3) = p(z|s_4)$, it will also be unbiased. An *invalid* answer, which we frequently encountered in class, pertains to the algorithm (instead of the problem formulation). For example, replacing particle filters by the exact discrete Bayes filer remedies the problem but is not a legitimate answer to this question. Neither is the use of infinitely many particles.

Exercise 26.MCLI

Implement Monte Carlo localization for a simulated robot with range sensors. A grid map and range data are available from the code repository at `aima.cs.berkeley.edu`. You should demonstrate successful global localization of the robot.

Implementing Monte Carlo localization requires a lot of work but is a premiere way to gain insights into the basic workings of probabilistic algorithms in robotics, and the intricacies inherent in real data. We have used this exercise in many courses, and students consistently expressed having learned a lot. We strongly recommend this exercise!

The implementation is not as straightforward as it may appear at first glance. Common problems include:

- The sensor model models too little noise, or the wrong type of noise. For example, a simple Gaussian will not work here.
- The motion model assumes too little or too much noise, or the wrong type of noise. Here a Gaussian will work fine though.
- The implementation may introduce unnecessarily high variance in the resulting sampling set, by sampling too often, or by sampling in the wrong way. This problem manifests itself by diversity disappearing prematurely, often with the wrong samples surviving. While the basic MCL algorithm, as stated in the book, suggests that sampling should occur after each motion update, implementations that sample less frequently tend to yield superior results. Further, drawing samples independently of each other is inferior to so-called low variance samplers. Here is a version of low variance sampling, in which $\mathcal{X}$ denotes the particles and W their importance weights. The resulting resampled particles reside in the set S'.

```
function LOW-VARIANCE-WEIGHTED-SAMPLE-WITH-REPLACEMENT($S, W$):
    $S' = \{\,\}$
    $b = \sum_{i=1}^{N} W[i]$
    $r = \text{rand}(0; b)$
    for $n = 1$ to $N$ do
        $i = \text{argmin}_j \sum_{m=1}^{j} W[m] \geq r$
        add $S[i]$ to $S'$
        $r = (r + \text{rand}(0; c))$ modulo $b$
    return $S'$
```

The parameter c determines the speed at which we cycle through the sample set. While each sample's probability remains the same as if it were sampled independently, the resulting samples are dependent, and the variance of the sample set S' is lower (assuming $c < b$). As a pleasant side effect, the low-variance samples is also easily implemented in $O(N)$ time, which is more difficult for the independent sampler.

- Samples are started in the occupied or unknown parts of the map, or are allowed into those parts during the forward sampling (motion prediction) step of the MCL algorithm.
- Too few samples are used. A few thousand should do the job, a few hundred will probably not.

The algorithm can be sped up by pre-caching all noise-free measurements, for all x-y-θ poses that the robot might assume. For that, it is convenient to define a grid over the space of all poses, with 10 centimeters spatial and 2 degrees angular resolution. One might then compute the noise-free measurements for the centers of those grid cells. The sensor model is clearly just a function of those correct measurements; and computing those takes the bulk of time in MCL.

26.5 Planning and Control

Exercise 26.ABMA

Consider a robot with two simple manipulators, as shown in figure **??**. Manipulator A is a square block of side 2 which can slide back and on a rod that runs along the x-axis from x=−10 to x=10. Manipulator B is a square block of side 2 which can slide back and on a rod that runs along the y-axis from y=−10 to y=10. The rods lie outside the plane of manipulation, so the rods do not interfere with the movement of the blocks. A configuration is then a pair $\langle x, y \rangle$ where x is the x-coordinate of the center of manipulator A and where y is the y-coordinate of the center of manipulator B. Draw the configuration space for this robot, indicating the permitted and excluded zones.

See Figure S26.2.

Exercise 26.ABMB

Suppose that you are working with the robot in Exercise 26.ABMA above and you are given the problem of finding a path from the starting configuration of figure **??** to the ending configuration. Consider a potential function

$$D(A, Goal)^2 + D(B, Goal)^2 + \frac{1}{D(A, B)^2}$$

where $D(A, B)$ is the distance between the closest points of A and B.

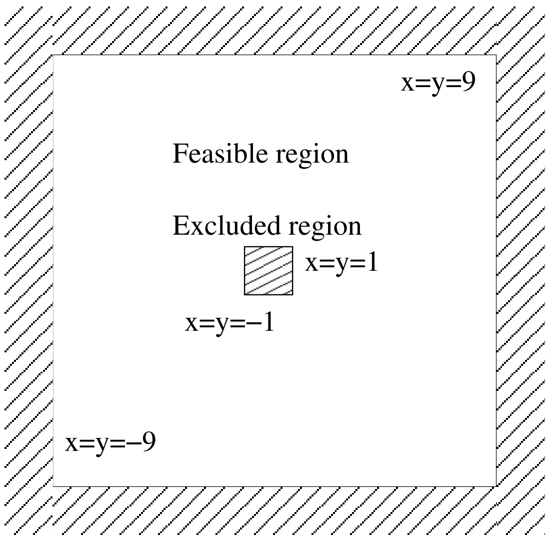


Figure S26.2 Robot configuration.

a. Show that hill climbing in this potential field will get stuck in a local minimum.

b. Describe a potential field where hill climbing will solve this particular problem. You need not work out the exact numerical coefficients needed, just the general form of the solution. (Hint: Add a term that "rewards" the hill climber for moving A out of B's way, even in a case like this where this does not reduce the distance from A to B in the above sense.)

A. Hill climbing down the potential moves manipulator B down the rod to the point where the derivative of the term "square of distance from current position of B to goal position" is exactly the negative of the derivative of the term "1/square of distance from A to B". This is a local minimum of the potential function, because it is a minimum of the sum of those two terms, with A held fixed, and small movements of A do not change the value of the term "1/square of distance from A to B", and only increase the value of the term "square of distance from current position of A to goal position"

B. Add a term of the form "1/square of distance between the center of A and the center of B." Now the stopping configuration of part A is no longer a local minimum because moving A to the left decreases this term. (Moving A to the left does also increase the value of the term "square of distance from current position of A to goal position", but that term is at a local minimum, so its derivative is zero, so the gain outweighs the loss, at least for a while.) For the right combination of linear coefficient, hill climbing will find its way to a correct solution.

Exercise 26.WSCS

This exercise explores the relationship between workspace and configuration space using the examples shown in Figure **??**.

a. Consider the robot configurations shown in Figure **??**(a) through (c), ignoring the obstacle shown in each of the diagrams. Draw the corresponding arm configurations in configuration space. (*Hint:* Each arm configuration maps to a single point in configuration space, as illustrated in Figure 26.12(b).)

b. Draw the configuration space for each of the workspace diagrams in Figure **??**(a)–(c). (*Hint:* The configuration spaces share with the one shown in Figure **??**(a) the region that corresponds to self-collision, but differences arise from the lack of enclosing obstacles and the different locations of the obstacles in these individual figures.)

c. For each of the black dots in Figure **??**(e)–(f), draw the corresponding configurations of the robot arm in workspace. Please ignore the shaded regions in this exercise.

d. The configuration spaces shown in Figure **??**(e)–(f) have all been generated by a single workspace obstacle (dark shading), plus the constraints arising from the self-collision constraint (light shading). Draw, for each diagram, the workspace obstacle that corresponds to the darkly shaded area.

e. Figure **??**(d) illustrates that a single planar obstacle can decompose the workspace into two disconnected regions. What is the maximum number of disconnected regions that can be created by inserting a planar obstacle into an obstacle-free, connected workspace, for a 2DOF robot? Give an example, and argue why no larger number of disconnected regions can be created. How about a non-planar obstacle?

a. The configurations of the robots are shown by the black dots in Figure S26.3.

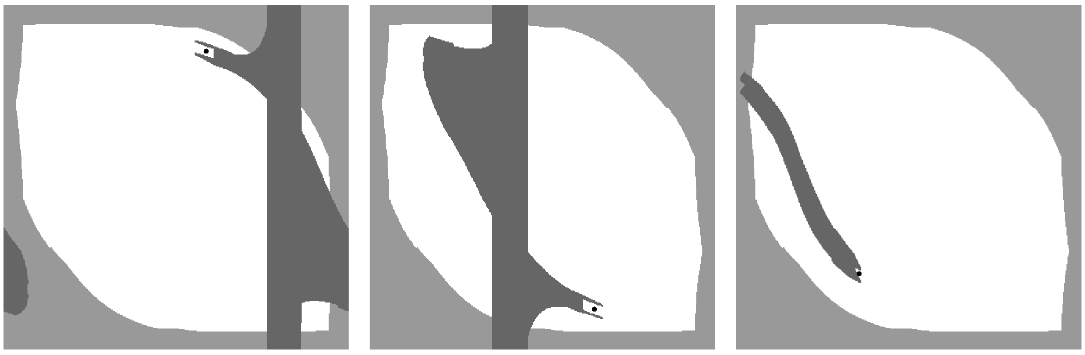

(a) (b) (c)

Figure S26.3 Configuration of the robots.

b. Figure S26.3 also answers the second part of this exercise: it shows the configuration space of the robot arm constrained by the self-collision constraint and the constraint imposed by the obstacle.

c. The three workspace obstacles are shown in Figure S26.4.

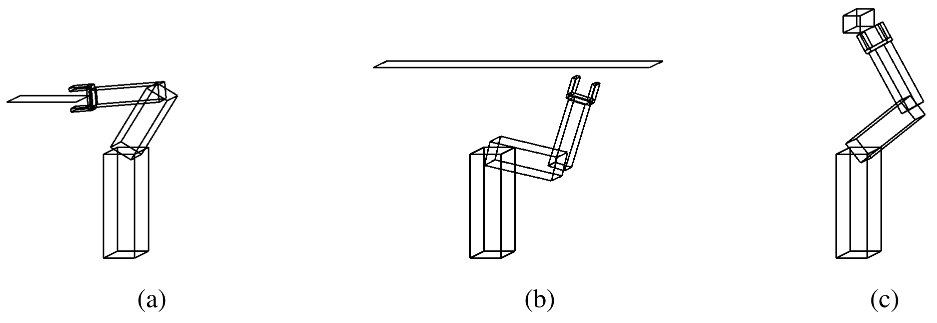


Figure S26.4 Workspace obstacles.

d. This question is a great mind teaser that illustrates the difficulty of robot motion planning! Unfortunately, for an arbitrary robot, a planar obstacle can decompose the workspace into *any* number of disconnected subspaces. To see, imagine a 1-DOF rigid robot that moves on a horizontal rod, and possesses N upward-pointing fingers, like a giant fork. A single planar obstacle protruding vertically into one of the free-spaces between the fingers could effectively separate the configuration space into $N+1$ disjoint subspaces. A second DOF will not change this.

More interesting is the robot arm used as an example throughout this book. By slightly extending the vertical obstacles protruding into the robot's workspace we can decompose the configuration space into five disjoint regions. The following figures show the configuration space along with representative configurations for each of the five regions.

Is five the maximum for any planar object that protrudes into the workspace of this particular robot arm? We honestly do not know; but we offer a $1 reward for the first person who presents to us a solution that decomposes the configuration space into six, seven, eight, nine, or ten disjoint regions. For the reward to be claimed, all these regions must be clearly disjoint, and they must be a two-dimensional manifold in the robot's configuration space.

For non-planar objects, the configuration space is easily decomposed into any number of regions. A circular object may force the elbow to be just about maximally bent; the resulting workspace would then be a very narrow pipe that leave the shoulder largely unconstrained, but confines the elbow to a narrow range. This pipe is then easily chopped into pieces by small dents in the circular object; the number of such dents can be increased without bounds.

Exercise 26.INVK

Consider the robot arm shown in Figure 26.12. Assume that the robot's base element is 70cm long and that its upper arm and forearm are each 50cm long. As argued on page **??**, the inverse kinematics of a robot is often not unique. State an explicit closed-form solution of the inverse kinematics for this arm. Under what exact conditions is the solution unique?

Let α be the shoulder and β be the elbow angle. The coordinates of the end effector are then given by the following expression. Here z is the height and x the horizontal displacement between the end effector and the robot's base (origin of the coordinate system):

$$\begin{pmatrix} x \\ z \end{pmatrix} = \begin{pmatrix} 0cm \\ 70cm \end{pmatrix} + \begin{pmatrix} \sin\alpha \\ \cos\alpha \end{pmatrix} \cdot 50cm + \begin{pmatrix} \sin(\alpha+\beta) \\ \cos(\alpha+\beta) \end{pmatrix} \cdot 50cm$$

Notice that this is only one way to define the kinematics. The zero-positions of the angles α and β can be anywhere, and the motors may turn clockwise or counterclockwise. Here we chose define these angles in a way that the arm points straight up at $\alpha = \beta = 0$; furthermore,

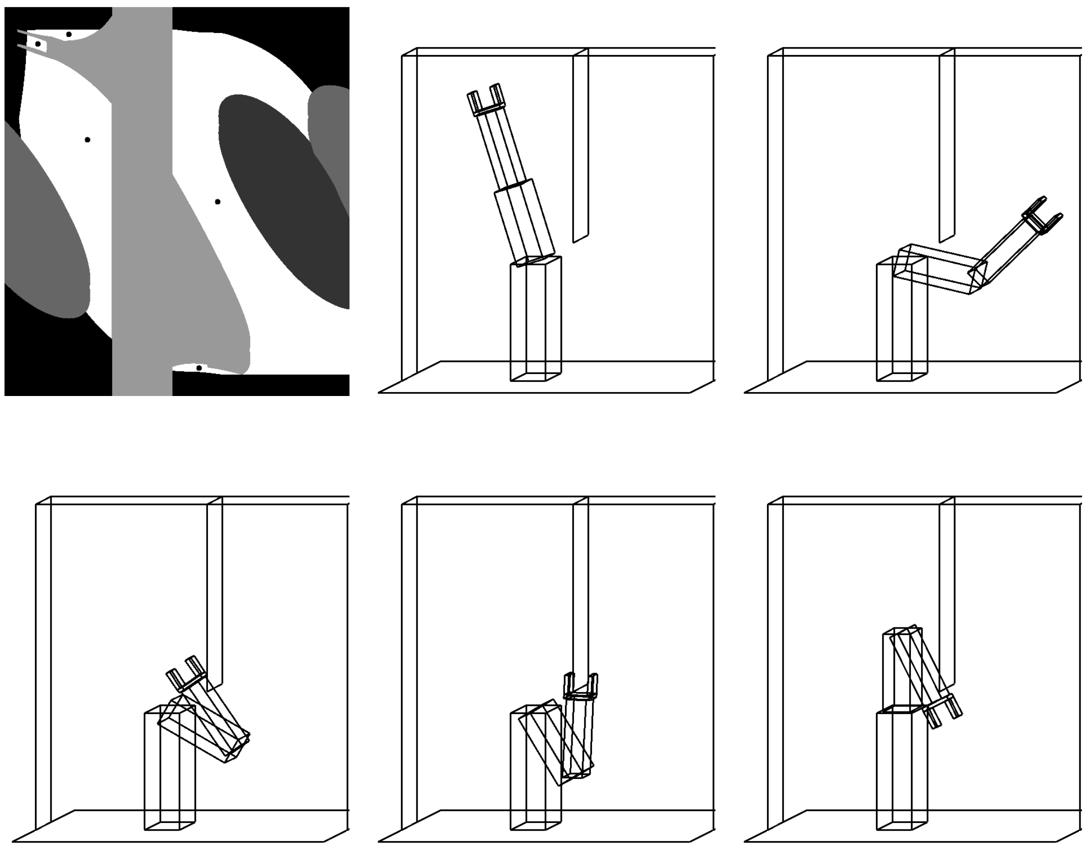


Figure S26.5 Configuration space for each of the five regions.

increasing α and β makes the corresponding joint rotate counterclockwise.

Inverse kinematics is the problem of computing α and β from the end effector coordinates x and z. For that, we observe that the elbow angle β is uniquely determined by the Euclidean distance between the shoulder joint and the end effector. Let us call this distance d. The shoulder joint is located $70cm$ above the origin of the coordinate system; hence, the distance d is given by $d = \sqrt{x^2 + (z - 70cm)^2}$. An alternative way to calculate d is by recovering it from the elbow angle β and the two connected joints (each of which is $50cm$ long): $d = 2 \cdot 50cm \cdot \cos \frac{\beta}{2}$. The reader can easily derive this from basic trigonometry, exploiting the fact that both the elbow and the shoulder are of equal length. Equating these two different derivations of d with each other gives us

$$\sqrt{x^2 + (z - 70cm)^2} \;=\; 100cm \cdot \cos \frac{\beta}{2} \tag{26.1}$$

or

$$\beta \;=\; \pm 2 \cdot \arccos \frac{\sqrt{x^2 + (z - 70cm)^2}}{100cm} \tag{26.2}$$

In most cases, β can assume two symmetric configurations, one pointing down and one pointing up. We will discuss exceptions below.

To recover the angle α, we note that the angle between the shoulder (the base) and the end effector is given by $\arctan 2(x, z - 70cm)$. Here $\arctan 2$ is the common generalization of the arcus tangens to all four quadrants (check it out—it is a function in C). The angle α is now obtained by adding $\frac{\beta}{2}$, again exploiting that the shoulder and the elbow are of equal length:

$$\alpha \;=\; \arctan 2(x, z - 70cm) - \frac{\beta}{2} \tag{26.3}$$

Of course, the actual value of α depends on the actual choice of the value of β. With the exception of singularities, β can take on exactly two values.

The inverse kinematics is ***unique*** if β assumes a single value; as a consequence, so does alpha. For this to be the case, we need that

$$\arccos \frac{\sqrt{x^2 + (z - 70cm)^2}}{100cm} \;=\; 0 \tag{26.4}$$

This is the case exactly when the argument of the $\arccos$ is 1, that is, when the distance $d = 100cm$ and the arm is fully stretched. The end points x, z then lie on a circle defined by $\sqrt{x^2 + (z - 70cm)^2} = 100cm$. If the distance $d > 100cm$, there is no solution to the inverse kinematic problem: the point is simply too far away to be reachable by the robot arm.

Unfortunately, configurations like these are numerically unstable, as the quotient may be slightly larger than one (due to truncation errors). Such points are commonly called *singularities*, and they can cause major problems for robot motion planning algorithms. A second singularity occurs when the robot is "folded up," that is, $\beta = 180°$. Here the end effector's position is identical with that of the robot elbow, regardless of the angle α: $x = 0cm$ and $z = 70cm$. This is an important singularity, as there are *infinitely* many solutions to the inverse kinematics. As long as $\beta = 180°$, the value of α can be arbitrary. Thus, this simple robot arm gives us an example where the inverse kinematics can yield zero, one, two, or infinitely many solutions.

Exercise 26.VORO

Implement an algorithm for calculating the Voronoi diagram of an arbitrary 2D environment, described by an $n \times n$ Boolean array. Illustrate your algorithm by plotting the Voronoi diagram for 10 interesting maps. What is the complexity of your algorithm?

Code not shown.

26.6 Planning Uncertain Movements

Exercise 26.MOBH

Consider a mobile robot moving on a horizontal surface. Suppose that the robot can execute two kinds of motions:

- Rolling forward a specified distance.
- Rotating in place through a specified angle.

The state of such a robot can be characterized in terms of three parameters $\langle x, y, \phi$, the x-coordinate and y-coordinate of the robot (more precisely, of its center of rotation) and the robot's orientation expressed as the angle from the positive x direction. The action "$Roll(D)$" has the effect of changing state $\langle x, y, \phi$ to $\langle x + D\cos(\phi), y + D\sin(\phi), \phi \rangle$, and the action $Rotate(\theta)$ has the effect of changing state $\langle x, y, \phi \rangle$ to $\langle x, y, \phi + \theta \rangle$.

a. Suppose that the robot is initially at $\langle 0, 0, 0 \rangle$ and then executes the actions $Rotate(60°)$, $Roll(1)$, $Rotate(25°)$, $Roll(2)$. What is the final state of the robot?

b. Now suppose that the robot has imperfect control of its own rotation, and that, if it attempts to rotate by θ, it may actually rotate by any angle between $\theta - 10°$ and $\theta + 10°$. In that case, if the robot attempts to carry out the sequence of actions in (A), there is a range of possible ending states. What are the minimal and maximal values of the x-coordinate, the y-coordinate and the orientation in the final state?

c. Let us modify the model in (B) to a probabilistic model in which, when the robot attempts to rotate by θ, its actual angle of rotation follows a Gaussian distribution with mean θ and standard deviation $10°$. Suppose that the robot executes the actions $Rotate(90°)$, $Roll(1)$. Give a simple argument that (a) the expected value of the location at the end is not equal to the result of rotating exactly $90°$ and then rolling forward 1 unit, and (b) that the distribution of locations at the end does not follow a Gaussian. (Do not attempt to calculate the true mean or the true distribution.)

The point of this exercise is that rotational uncertainty quickly gives rise to a lot of positional uncertainty and that dealing with rotational uncertainty is painful, whether uncertainty is treated in terms of hard intervals or probabilistically, due to the fact that the relation between orientation and position is both non-linear and non-monotonic.

A. $x = 1 \cdot cos(60°) + 2 \cdot cos(85°) =.$
$y = 1 \cdot sin(60°) + 2 \cdot sin(85°) =.$

$\phi = 90°$

B. The minimal value of x is $1 \cdot \cos(70°) + 2 \cdot \cos(105°) = -0.176$
achieved when the first rotation is actually $70°$ and the second is actually $35°$.
The maximal value of x is $1 \cdot \cos(50°) + 2 \cdot \cos(65°) = 1.488$
achieved when the first rotation is actually $50°$ and the second is actually $15°$.
The minimal value of y is $1 \cdot \sin(50°) + 2 \cdot \sin(65°) = 2.579$
achieved when the first rotation is actually $50°$ and the second is actually $15°$.
The maximal value of y is $1 \cdot \sin(70°) + 2 \cdot \sin(90°) = 2.94$
achieved when the first rotation is actually $70°$ and the second is actually $20°$.
The minimal value of ϕ is $65°$ achieved when the first rotation is actually $50°$ and the second is actually $15°$.
The maximal value of ϕ is $105°$ achieved when the first rotation is actually $70°$ and the second is actually $35°$.

C. The maximal possible y-coordinate (1.0) is achieved when the rotation is executed at exactly $90°$. Since it is the maximal possible value, it cannot be the mean value. Since there is a maximal possible value, the distribution cannot be a Gaussian, which has non-zero (though small) probabilities for all values.

[[need exercises]]

26.7 Reinforcement Learning in Robotics

[[need exercises]]

26.8 Humans and Robots

[[need exercises]]

26.9 Alternative Robotic Frameworks

Exercise 26.ROBE

Consider the simplified robot shown in Figure **??**. Suppose the robot's Cartesian coordinates are known at all times, as are those of its goal location. However, the locations of the obstacles are unknown. The robot can sense obstacles in its immediate proximity, as illustrated in this figure. For simplicity, let us assume the robot's motion is noise-free, and the state space is discrete. Figure **??** is only one example; in this exercise you are required to address all possible grid worlds with a valid path from the start to the goal location.

a. Design a deliberate controller that guarantees that the robot always reaches its goal location if at all possible. The deliberate controller can memorize measurements in the form of a map that is being acquired as the robot moves. Between individual moves, it may spend arbitrary time deliberating.

b. Now design a *reactive* controller for the same task. This controller may not memorize past sensor measurements. (It may not build a map!) Instead, it has to make all decisions based on the current measurement, which includes knowledge of its own location and that of the goal. The time to make a decision must be independent of the environment size or the number of past time steps. What is the maximum number of steps that it may take for your robot to arrive at the goal?

c. How will your controllers from (a) and (b) perform if any of the following six conditions apply: continuous state space, noise in perception, noise in motion, noise in both perception and motion, unknown location of the goal (the goal can be detected only when within sensor range), or moving obstacles. For each condition and each controller, give an example of a situation where the robot fails (or explain why it cannot fail).

A simple deliberate controller might work as follows: Initialize the robot's map with an empty map, in which all states are assumed to be navigable, or free. Then iterate the following loop: Find the shortest path from the current position to the goal position in the map using A*; execute the first step of this path; sense; and modify the map in accordance with the sensed obstacles. If the robot reaches the goal, declare success. The robot declares failure when A* fails to find a path to the goal. It is easy to see that this approach is both complete and correct. The robot always find a path to a goal if one exists. If no such path exists, the approach detects this through failure of the path planner. When it declares failure, it is indeed correct in that no path exists.

A common reactive algorithm, which has the same correctness and completeness property as the deliberate approach, is known as the BUG algorithm. The BUG algorithm distinguishes two modes, the boundary-following and the go-to-goal mode. The robot starts in go-to-goal mode. In this mode, the robot always advances to the adjacent grid cell closest to the goal. If this is impossible because the cell is blocked by an obstacle, the robot switches to the boundary-following mode. In this mode, the robot follows the boundary of the obstacle until it reaches a point on the boundary that is a local minimum to the straight-line distance to the goal. If such a point is reached, the robot returns to the go-to-goal mode. If the robot reaches the goal, it declares success. It declares failure when the same point is reached twice, which can only occur in the boundary-following mode. It is easy to see that the BUG algorithm is correct and complete. If a path to the goal exists, the robot will find it. When the robot declares failure, no path to the goal may exist. If no such path exists, the robot will ultimately reach the same location twice and detect its failure.

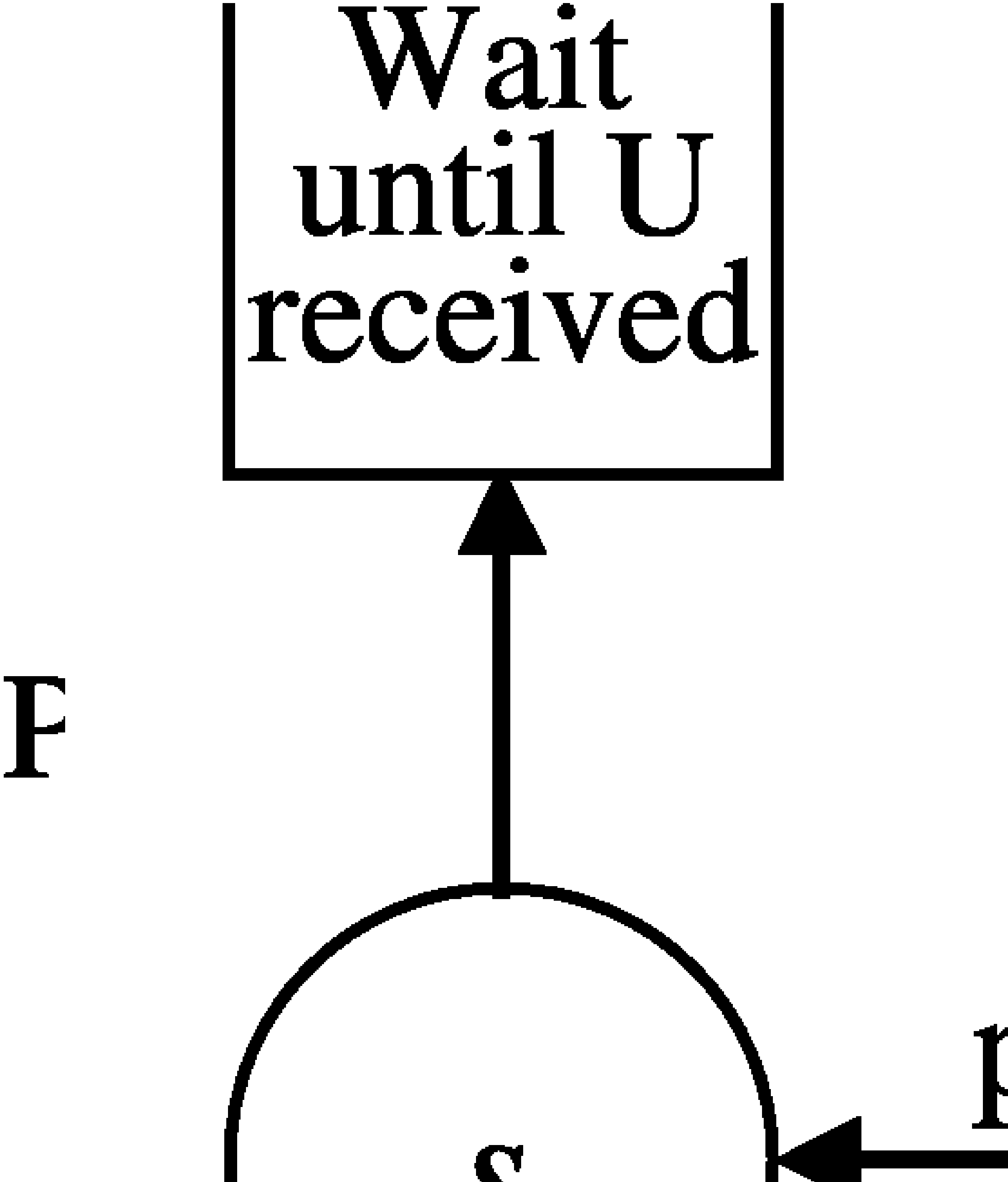
Both algorithms can cope with continuous state spaces provides that they can accurately perceive obstacles, plan paths around them (deliberative algorithm) or follow their boundary (reactive algorithm). Noise in motion can cause failures for both algorithms, especially if the robot has to move through a narrow opening to reach the goal. Similarly, noise in perception destroys both completeness and correctness: In both cases the robot may erroneously conclude a goal cannot be reached, just because its perception was noise. However, a deliberate algorithm might build a probabilistic map, accommodating the uncertainty that arises from the noisy sensors. Neither algorithm as stated can cope with unknown goal locations; however, the deliberate algorithm is easily converted into an *exploration* algorithm by which the robot always moves to the nearest unexplored location. Such an algorithm would be complete and correct (in the noise-free case). In particular, it would be guaranteed to find and reach the goal when reachable. The BUG algorithm, however, would not be applicable. A common reactive technique for finding a goal whose location is unknown is random motion; this algorithm will with probability one find a goal if it is reachable; however, it is unable to determine when to give up, and it may be highly inefficient. Moving obstacles will cause problems for both the deliberate and the reactive approach; in fact, it is easy to design an adversarial case where the obstacle always moves into the robot's way. For slow-moving obstacles, a common deliberate technique is to attach a timer to obstacles in the grid, and erase them after a certain number of time steps. Such an approach often has a good chance of succeeding.

Exercise 26.SUBA

In Figure 26.32(b) on page 976, we encountered an augmented finite state machine for the control of a single leg of a hexapod robot. In this exercise, the aim is to design an AFSM that, when combined with six copies of the individual leg controllers, results in efficient, stable locomotion. For this purpose, you have to augment the individual leg controller to pass messages to your new AFSM and to wait until other messages arrive. Argue why your controller is efficient, in that it does not unnecessarily waste energy (e.g., by sliding legs), and in that it propels the robot at reasonably high speeds. Prove that your controller satisfies the dynamic stability condition given on page **??**.

There are a number of ways to extend the single-leg AFSM in Figure 25.22(b) into a set of AFSMs for controlling a hexapod. A straightforward extension—though not necessarily the most efficient one—is shown in the following diagram. Here the set of legs is divided into two, named A and B, and legs are assigned to these sets in alternating sequence. The top level controller, shown on the left, goes through six stages. Each stage lifts a set of legs, pushes the ones still on the ground backwards, and then lowers the legs that have previously been lifted. The same sequence is then repeated for the other set of legs. The corresponding single-leg controller is essentially the same as in Figure 25.22(b), but with added wait-steps for synchronization with the coordinating AFSM. The low-level AFSM is replicated six times, once for each leg.

For showing that this controller is stable, we show that at least one leg group is on the ground at all times. If this condition is fulfilled, the robot's center of gravity will always be above the imaginary triangle defined by the three legs on the ground. The condition is easily proven by analyzing the top level AFSM. When one group of legs in s_4 (or on the way to s_4 from s_3), the other is either in s_2 or s_1, both of which are on the ground. However, this proof only establishes that the robot does not fall over when on flat ground; it makes no assertions about the robot's performance on non-flat terrain. Our result is also restricted to *static stability*, that is, it ignores all dynamic effects such as inertia. For a fast-moving hexapod, asking that its center of gravity be enclosed in the triangle of support may be insufficient.


Wait until U received
F
S

Exercise 26.HUMR

(This exercise was first devised by Michael Genesereth and Nils Nilsson. It works for first graders through graduate students.) Humans are so adept at basic household tasks that they often forget how complex these tasks are. In this exercise you will discover the complexity and recapitulate the last 30 years of developments in robotics. Consider the task of building an arch out of three blocks. Simulate a robot with four humans as follows:

Brain. The Brain direct the hands in the execution of a plan to achieve the goal. The Brain receives input from the Eyes, but *cannot see the scene directly*. The brain is the only one who knows what the goal is.

Eyes. The Eyes report a brief description of the scene to the Brain: "There is a red box standing on top of a green box, which is on its side" Eyes can also answer questions from the Brain such as, "Is there a gap between the Left Hand and the red box?" If you have a video camera, point it at the scene and allow the eyes to look at the viewfinder of the video camera, but not directly at the scene.

Left hand and **right hand.** One person plays each Hand. The two Hands stand next to each other, each wearing an oven mitt on one hand, Hands execute only simple commands from the Brain—for example, "Left Hand, move two inches forward." They cannot execute commands other than motions; for example, they cannot be commanded to "Pick up the box." The Hands must be *blindfolded*. The only sensory capability they have is the ability to tell when their path is blocked by an immovable obstacle such as a table or the other Hand. In such cases, they can beep to inform the Brain of the difficulty.

We have used this exercise in class to great effect. The students get a clearer picture of why it is hard to do robotics. The only drawback is that it is a lot of fun to play, and thus the students want to spend a lot of time on it, and the ones who are just observing feel like they are missing out. If you have laboratory or TA sections, you can do the exercise there.

Bear in mind that being the Brain is a very stressful job. It can take an hour just to stack three boxes. Choose someone who is not likely to panic or be crushed by student derision. Help the Brain out by suggesting useful strategies such as defining a mutually agreed Hand-centric coordinate system so that commands are unambiguous. Almost certainly, the Brain will start by issuing absolute commands such as "Move the Left Hand 12 inches positive y direction" or "Move the Left Hand to (24,36)." Such actions will never work. The most useful "invention" that students will suggest is the guarded motion discussed in Section 25.5—that is, macro-operators such as "Move the Left Hand in the positive y direction until the eyes say the red and green boxes are level." This gets the Brain out of the loop, so to speak, and speeds things up enormously.

We have also used a related exercise to show why robotics in particular and algorithm design in general is difficult. The instructor uses as props a doll, a table, a diaper and some safety pins, and asks the class to come up with an algorithm for putting the diaper on the baby. The instructor then follows the algorithm, but interpreting it in the least cooperative way possible: putting the diaper on the doll's head unless told otherwise, dropping the doll on the floor if possible, and so on.

[[need exercises]]

26.10 Application Domains

[[need exercises]]