Design Lab 10: Hallway World

- **Athena machines:** Do `add -f 6.01` and `athrun 6.01 update`.
- **Lab laptop:** Do `athrun 6.01 update`.
- **Personal laptop:** Download software `swlab10.zip` file from course web page.

Be sure to start idle with the **-n** flag.

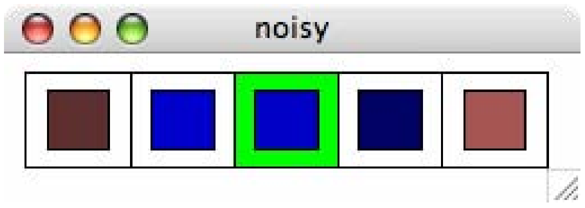
In the next two labs, we'll use basic probabilistic modeling to build a system that estimates the robot's pose, based on noisy sonar and odometry readings. We'll start by building up your intuition for these ideas in a simple simulated world, then we'll move on to modeling and using the real robots, next week. This week we'll start by working with a non-deterministic grid-world simulator.

1 Hallway World

We are going to work with an abstract simulation of a robot moving up and down a hallway made up of a discrete number of rooms, each of which has a color (think of it as what color the walls are painted in that room). The robot can make a possibly noisy observation of the color of the room it is in. The robot can try to move up to 4 squares in either direction. If it attempts to move off the end of the hallway, it stays where it is, and it may not always move to the room it intends to. We will consider state estimation when:

- The robot **knows** the colors of the rooms in the world.
- The robot **knows** its own sensing and transition abilities.
- The robot **does not know** which room it is in.

Here is an instance of a colored hallway world, showing both the colors of the rooms (in this case, white, white, green, white, white) and the robot's belief state. The colors of the rooms are the colors of the outer part of each square.



The robot's *belief state* is a probability distribution over which room it is in. In the window, the current belief state is displayed in the colors of the inner squares in each block, with brighter

Athena is MIT's UNIX-based computing environment. OCW does not provide access to it red values closer to zero and brighter blue values closer to one. The color black is assigned to the probability associated with the uniform distribution (in this case 0.2). The belief state is also printed out whenever you move; in fact it is printed twice: once after the sensing update and again after the transition update.

Start idle with `-n`, and load the file `dl10Work.py` and run it. Now, type

```
p = makePerfect()
p.run(10)
```

At this point, `p` is an instance of a little application that both simulates the robot moving in the world and shows the estimated belief state. When you run it, it prompts you for inputs. You can type an integer from `-4` to `4` to move it. It will attempt to move that many spaces to the right (so it moves left if the command is negative). If you want to quit, type `quit`. If you type `0`, the robot will attempt to stay in its current location. If you anything else, it will print an error message and stay where it is. It will stop after 10 steps unless you call `run` with a larger numeric argument.

The world above has perfect motion and perfect sensing. You can create and run one with noisy motion and sensing as follows:

```
n = makeNoisy()
n.run(20)
```

On each step, the robot makes an observation of the room it is in, and then it executes the specified action.

Check Yourself 1. A. Make an instance of a perfect simulator and move the robot around. Be sure you understand what the colors representing the belief state mean and that the numbers being printed out in the Python shell make sense.

B. Make an instance of a noisy simulator and move the robot around. Be sure you understand what the colors mean, and have a basic idea of what might be going on. Feel free to ask a staff member for clarification.

1.1 The observation model

The observation model specifies a probability distribution over what the robot sees given what state it is in: $P(O_t = o_t | S_t = s_t)$. In our case o_t ranges over

```
('black', 'white', 'red', 'green', 'blue', 'purple', 'orange', 'darkGreen',
'gold', 'chocolate', 'PapayaWhip', 'MidnightBlue', 'HotPink', 'chartreuse')
```

and s_t ranges over all the possible grid squares the robot could be in.

If there are m possible observations and n possible locations, then it will, in general, require $m \cdot n$ numbers to specify the observation model. In this problem, we will make an assumption that makes the model much more compact: the robot's observation only depends on the *color* of the square it's in. That is, all white squares have the same distribution over possible observations and all green squares have the same (though different from the white ones) distribution over possible observations. Given this assumption, we only need to specify $P(\mathit{observedColor} \mid \mathit{actualColor})$, and then we can find the probability of observing each color in any state, as long as we know the actual color of that state:

$$P(O_t = \mathit{observedColor} \mid S_t = s_t) = P(O_t = \mathit{observedColor} \mid \mathit{ActualColor} = \mathit{actualColor}(s_t)) \enspace .$$

We will call this conditional probability distribution

$$P(O_t = \mathit{observedColor} \mid \mathit{ActualColor} = \mathit{actualColor}(s_t)) \enspace .$$

which specifies a distribution on the observed color given the actual color of the robot's room, the *observation noise distribution*. We can specify an observation noise distribution in Python as a procedure that takes an actual color as input and returns a distribution on observed colors. Here is a very simple example that always observes the true color.

```
def perfectObsNoiseModel(actualColor):
    return dist.DDist({actualColor: 1.0})
```

Now, if we have an observation noise distribution, such as `perfectObsNoiseModel`, defined, we can use it to construct the entire observation model as shown below:

```
def makeObservationModel(hallwayColors, obsNoise):
    return lambda loc: obsNoise(hallwayColors[loc])
```

Here, `hallwayColors` is a list specifying the true color of each location in the hallway and `obsNoise` is a conditional distribution of observed color given actual color. This procedure returns a conditional probability distribution, which is a procedure that takes a location as input and returns a distribution over observed colors.

In our example shown above,

```
standardHallway = ['white', 'white', 'green', 'white', 'white']
```

Given these procedures, we can specify the observation model for perfect observations with:

```
perfectObsModel = makeObservationModel(standardHallway, perfectObsNoiseModel)
```

Wk.10.1.1 Do this tutor problem **Wk.10.1.1** on defining observation models. You may find it most useful also to do **Wk.10.1.5, Part 1, Questions 1 and 2** and **Wk.10.1.5, Part 2, Questions 1 - 3** at this point.

Checkoff 1. We have a test world that has only alternating green and white squares. Try out your two observation noise models that confuse green and white, and see what happens to the belief state. Use the perfect motion models, as shown below. **Be sure that you use something like `whiteEqGreenObsDist` as the third argument below.**

```
w = makeSim(alternating, actions,
            <your observedColor given actualColor model>,
            standardDynamics, perfectTransNoiseModel)
w.run(50)
```

How do those two perceptual models compare to the two described below?

- A perfect sensor model
- A sensor that always reads 'black' no matter what room it is in

Explain your answers to a staff member.

1.2 The state-transition model

The state-transition model specifies a probability distribution over the state at time $t+1$, given the state at time t and the selected action a. That is, $P(S_{t+1} = s_{t+1} | S_t = s_t, A_t = a_t)$. The next state of the system depends both on where it was before and the action that was taken. If you were to write this model out as a matrix, it would be very big: mn^2, where n is the number of states of the world and m is the number of actions.

Often, the transition model can be described more sparsely or systematically. In this particular world, the robot can try to move some number of squares to the right or left, or to stay in its current location. We'll assume that the kinds of errors the robot makes when it tries to move in a given direction don't depend on where the robot actually is (except if it is at the edge of the world), and we'll further assume that the transition probabilities to most next states are zero (there's no chance of the robot teleporting to the other end of the hallway, for example). This will allow us to describe the transition model more compactly.

We will start on defining the transition model by first defining a *dynamics* procedure, such as `standardDynamics` below. The dynamics is a procedure which returns the nominal new location resulting from taking action `act` in location `loc`, in a hallway with `hallwayLength` locations. This will be useful in other problems. The possible actions that the robot can take are: $-4, \ldots, -1, 0, 1, \ldots, 4$. So, we can just add the robot's action to its current location to get its nominal new location; except we have to be sure it doesn't move off of the edges of the world, so we use `util.clip` to keep the value from going below 0 or above `hallwayLength - 1`.

```
def standardDynamics(loc, act, hallwayLength):
    return util.clip(loc + act, 0, hallwayLength-1)
```

Then, we will define a *noise model*, that is independent of location, which returns a distribution over the possible actual locations given the nominal location that results from an action under the given dynamics. The simplest noise model assumes that transitions are perfect, so that the actual location will be the nominal location.

```
def perfectTransNoiseModel(nominalLoc, hallwayLength):
    return dist.DDist({nominalLoc : 1.0})
```

Ultimately, we need to make a full transition model, which is a conditional probability distribution fo the form $\Pr(S_{t+1} \mid S_t, A_t)$. Because it is conditioned on two variables, we will represent it using nested procedures (representing conditional distributions). So, we'll think of it as something like $\Pr(S_{t+1} \mid S_t \mid A_t)$, or, as a procedure that takes an a_t and returns a procedure that takes an s_t and returns a distribution over S_{t+1}.

Here is the basic form of the transition model that takes two procedures, a dynamics procedure and a noise model, and an integer indicating the length of the hallway.

```
def makeTransitionModel(dynamics, noiseDist, hallwayLength):
    return lambda act: lambda loc: noiseDist(dynamics(loc, act, hallwayLength),
                                             hallwayLength)
```

A perfect transition model for our standard hallway under the standard dynamics is constructed as follows:

```
perfectTransModel = makeTransitionModel(standardDynamics, perfectTransNoiseModel, 5)
```

Wk.10.1.3 Do tutor problem **Wk.10.1.3** on defining transition models. You may find it useful also to do **Wk.10.1.5, Part 1, Question 3** and **Wk.10.1.5, Part 2, Questions 4** at this point.

Checkoff 2. We have a test world that has only white squares. We can initialize it so that it knows for sure it is in location 7.

```
w = makeNoisyKnownInitLoc(7, sterile)
w.run(50)
```

It will use the noisy motion (and observation) models. Experiment with typing 0 several times in a row. What happens? What happens when you drive the robot around? Demonstrate and explain to a staff member.

2 State estimation

State estimation is the process of taking in a sequence of actions and observations and computing a probability distribution that represents our state of belief about the internal hidden state of the machine. We will build intuition by doing some numerical examples of state estimation by hand. These are types of problems that we will expect you to be able to do in quizzes and exams.

Wk.10.1.5 Do the rest of this problem on state estimation in the hallway world.

Wk.10.1.6 This problem is optional, but we strongly encourage it, either now, or as review.

3 Remaining work for the week

If you have time here in lab, start on

- Upload problems for week 10, about defining distributions over ranges of integers. **Note that these problems are due before design lab on Nov. 19.**
- Tutor problems in section **Wk.10.2**

All of these exercises will contribute substantially to your ability to complete design lab 11 successfully.