MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.01—Introduction to EECS I
Spring Semester, 2008

Week 13 Course Notes

Abstraction and Modeling for a Complex Estimation Problem

The goal of this lecture is to see how to apply the ideas of state estimation and control to the large continuous spaces of a real robot. Imagine that one of our robots is turned loose in the Stata Center, with the goal of getting to Hal's office. It has an accurate map of the floor, but has no idea where it is, to start with. As it rolls around, it makes sonar readings, which give it an indication of the current distance to walls in the various directions (but, of course, especially if we don't coat the Stata Center in bubble wrap, the readings will be noisy). In addition, as it rolls around, it will count wheel revolutions to give it an indication of how far it has moved, but wheel-slip and different surfaces will make that information noisy, as well.

What is the poor robot to do? It has to do some combination of figuring out where it is, and trying to get where it needs to go. We'll start by considering the problem of figuring out where it is, which is also called *localization*.

Robot Localization

We have studied a model of state estimation that assumes discrete time, state, action, and observation. Our robot operates in continuous space, over continuous time; its actions are trajectories through continuous space; its observations are sonar readings, which although technically digitized to 16 or 32 bits, are appropriately treated as being real-valued.

There are models for state-estimation in continuous domains, but they typically make very strong assumptions about the transition and observation distributions. We'll take a mixed approach, discretizing the state space, but treating actions and observations as if they were continuous. This particular strategy is only one point in a huge space of possible choices. We'll talk about how it works, and why we made the decisions we did, but there is no right answer. We have to balance trade-offs between accuracy in modeling the world and computational efficiency of the resulting system.

State space

We start by discretizing the state space, similar to the way we did for planning. However, in order to appropriately interpret the sonar readings, we will need to have a much more fine-grained discretization of the robot's heading. So, our state space will be a three dimensional grid, with discrete coordinates representing the x, y, and θ (orientation) of the robot, in the global coordinate frame in which the map is described.

There is already an interesting trade-off here: if we discretize each of these dimensions into 20 bins, we will have a space of 8,000 states. The finer the discretization, we might expect, the more precise the results; but also the more computation necessary to do the state updates.

Transitions

The simplest way to model actions would be to discretize them, as well (as we did for the planner). So, we might imagine having an action space of moving forward or turning a fixed distance. But we would ideally like our state estimator to be able to run in parallel with any system for choosing the robot's actions, including random wandering.

So, instead, we'll allow the robot driver to do whatever it wants to do, and just assume that periodically our state estimator is called and asked to update the state. Here's how it works. When the state estimator is initialized, and whenever it is called subsequently, we will pass in the pose as computed by the odometry. This pose is reported in a coordinate frame that is *completely* disconnected from the coordinate frame in which the map is specified (after all, if we knew where we were in the map frame, we'd be done!).

We compute the *delta pose* (change in pose) between the current pose and the pose that we had the last time the update was done. This change in pose is represented *relative* to the robot's pose on the previous step. So, it is independent of the odometry reference frame. We treat this relative delta pose as the action the robot just took.

Now, we have to figure out transition probabilities. Generally, we would model the error in the odometry: if the robot reports that it has made a particular change in pose, what change has it, in fact, made? This will depend on how much it turned (turning is much less reliably measured than straight motion), what kind of surface it was on, etc. If we were modeling states as continuous, we might model the robot's error as a *Gaussian distribution*, with its center at the reported delta pose, and some error around it.

Because we are using a discretized state space, it's more complicated. Imagine that there were no error at all in the reported odometry. We would still end up with probabilities in our transition distribution because of the size of the grid squares. If, for instance, I know the robot is in a one meter square, and it moved half a meter forward, where do I think it is now? We have uncertainty in our actions introduced by uncertainty in the state. We'll model it by saying there's probability 0.5 that the robot stays in the square it's in and 0.5 that it moves forward one.

Observations

Finally, we have to model the robot's sensory abilities. We'll just model the sonar sensors, which can give us a lot of information about where the robot is. But we have to remember that there is noise in the sensor readings and also that many different (x, y, θ) poses of the robot would yield the same sensor readings. So, for us, an observation will be a vector of 8 sonar readings, $\langle o_0 \dots o_7 \rangle$. The sensor model is supposed to give us a probability distribution on the sonar readings, given the robot's pose: $\Pr(\langle o_0 \dots o_7 \rangle | x, y, \theta)$. This seems very difficult, because the sonar readings are continuous values and because there are eight of them.

Let's tackle the second problem first. Given the robot's pose, we can easily compute a *nominal* set of sonar readings; let's call them $\langle g_0 \dots g_7 \rangle$ (where g stands for good). These are the readings we would get if the sonars were perfect, and returned the distance along the ray centered at each sensor to the nearest obstacle. Just as we did in the simple grid world, we're going to make the simplifying assumption that the noisy sonar readings only depend on the nominal readings; that any two poses of the robot with the same nominal readings will generate the same distribution over actual readings. So, now our problem is to specify $\Pr(\langle o_0 \dots o_7 \rangle | \langle g_0 \dots g_7 \rangle)$.

It's time for yet another reasonable assumption (you'll find that the art of making mathematical models of the real world is figuring out what kinds of assumptions are both mathematically helpful and realistically plausible). We'll assume that the amount of noise in one sensor reading is independent of the noise in the other sensor readings, given the nominal readings. This would be a good assumption if the noise is a property of the individual sonar sensors for example; it would be a bad assumption if it were a more global property of the environment, like having an ultrasonic rodent-repeller in the room. If we can make this independence assumption, we get a huge simplification in our problem:

$$\begin{aligned} \Pr(\langle o_0 \dots o_7 \rangle | x, y, \theta) &= \Pr(\langle o_0 \dots o_7 \rangle | \langle g_0 \dots g_7 \rangle) \\ &= \prod_{i=0}^{7} \Pr(o_i | g_i) \end{aligned}$$

For those of you have run across the notion of independence or conditional independence of random variables, it will make sense that we can convert the joint distribution into a product.

And assuming that all of the sensors have basically the same noise behavior, we only need to specify a single model: $\Pr(o|g)$. This is a probability distribution over a single actual sonar reading o, given its nominal reading g. We could do this by discretizing the values of o, and providing a probability for each of those values. But we'd have to discretize the g values, too, and the table starts to get big. Instead, we'll keep things continuous and use a Gaussian distribution, which is a way to say which elements of a continuous interval are more likely than which other ones.

Just to review, the Gaussian (also called normal) distribution has two parameters: the mean μ, which describes the most likely value, and the variance σ^2, which describes how spread out the distribution is. If we fix σ^2 to some value (we could determine this by collecting sonar readings and doing some statistical analysis) and set μ to be the nominal value g, then this distribution provides a plausible description of the likelihood of various o values.

So, we'll assume that the conditional distribution on an observation random-variable O, which has nominal value g, is $N(g, \sigma^2)$; that is, that it is normally distributed with mean g and variance σ^2. In our model, which you'll use below, we set σ to 0.5, which makes the variance 0.25. (As you must know by now, this is a pretty terrible model of the error in the sonars: we should really be modeling the fact that the sonars often bounce off of walls when the angle is too steep, but we're being lazy for now).

Planning

Now, what if we want the robot to go to some particular location? We can use our old state-space search approach to planning, but with some modifications.

First, the basic search model assumes we know the initial state of the system. We don't know the robot's pose exactly—we spent much of this lab just trying to figure out where the robot is, and ending up with a distribution over its pose. We will make a big assumption for the purposes of planning, that the robot is actually in the pose that is currently the most likely, according to the belief state.

Second, the basic search model assumes that the "dynamics" of the world are deterministic. That is, among the set of possible successors of a state, we can reliably choose which one will occur. This

is like assuming there is no noise in the transition model, and each action has exactly one outcome state, with probability one. We will make this assumption for the purposes of planning.

This is strategy might seem pretty misguided. We don't know where the robot is, and we don't know what the outcomes of its actions are going to be, but we're going to close our eyes and generate plans anyway. This would be completely ridiculous if we were then going to execute the entire plan without looking at the world again. But we will gain a large measure of robustness by pursuing an approach called *continuous replanning*. This means that after the robot takes the first step of a plan, we will re-estimate the pose, take the most likely one to be our initial state, and re-plan. Thus, errors in localization and execution can be corrected after a single step.