Agent/Testbed Simulator Project

CMPS 523
Dr. Maida
Draft

March 4, 2009

1 Introduction

This assignment requires you to implement an agent/testbed simulator to study algorithms in the *Probabilistic Robotics* text. The simulator allows a simulated agent to interact with a simulated world, known as the testbed. The tool provides a graphical display of the world (testbed) state so that the user can observe the agent's behavior in the world. The simulator also provides a graphical display of the agent's belief state superimposed on the testbed graphics. The simulator has three primary functions:

1. The user can *setup* the testbed state and the agent state. This includes placing objects in the testbed, positioning the agent in the testbed, choosing simulation time-step size, and setting the noise parameters within the testbed. In regard to the agent state, the user can set the agent's initial belief and specify the particular Bayesian filter algorithm that the agent uses to update its beliefs.

2. The user can *run* the simulation to see how the agent updates its beliefs as it explores the world. As the beliefs are updated, the graphical display is updated so that the user can see that agent's belief state superimposed on the actual world state.

3. The user can *export* simulation data so that it can be analyzed in Matlab.

1.1 Deliverables

There are two parts or deliverables for this project. The first part is worth 50% of the project grade and the second part is worth 50% of the project grade. The entire project is worth 25% of the course grade.

1. The first deliverable is a simulation that is able to model the scenaries given in Figures 7.8, 7.9, and 7.10 of the text using the extended Kalman filter given in Table 7.2. It should also

be able to model the scenario in Figure 8.12 of the text using the MCL filter given in Table 8.2

The easiest language to use for this project is Java using the Java 2D graphics package. Alternatively, you can use C++ with OpenGL and GLUT.

2. The second deliverable is a ten-to-twelve page paper written in a conference paper format. The paper is to describe the results, comparisons, and tests that you have performed using your simulator. The paper should have the following format.

Title By this, I mean the title, author, and affiliation.

Abstract This is a self-contained summary of the paper. The purpose of the abstract is to allow someone to determine whether the paper is relevent to his or her needs.

Introduction The introduction of the paper introduces the topic, explains the purpose of the paper, and foreshadows the results.

Literature survey This part of the paper is optional. I mention it here so that you are aware that a real conference paper needs a literature survey. The purpose of the literature survey is to describe the current state of the art so that you can explain why the research in your paper is a new contribution.

Methods This part of the paper explains how you did your simulation experiments, what the algorithms are, and what parameter values you used in your simulations. The purpose of this section is to give enough information to allow the reader to reproduce, or replicate, your work if desired.

Results In this part of the paper, you present the results of your simulation experiments. For instance, does algorithm A perform better than algorithm B? You might present these results using a table or graph of data points that you obtained by performing simulation experiments.

Conclusion This is were you interpret the experimental results presented in the previous section and draw some general conclusions. It is also a place for you to explain the limitations of your work and future work that you think should be carried out.

Bibliography This is were you include references to the literature. In your case, you should cite the class textbook because it is the source of the algorithms that you used in the simulations.

2 The Agent/Testbed Architecture

Interaction between the agent and the testbed is very restricted. Interaction occurs in discrete time steps whose size is Δt. In a particular Δt, the agent sends a $\mathbf{u}_t$ directive to the testbed and the

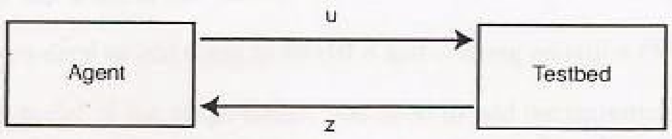

Figure 1: Agent/testbed interaction. At each time step, the agent issues a control command, u_t to the testbed. The agent may or may not receive a measurement, z_t.

---

testbed responds with $\mathbf{z}_t$ data. This interaction is depicted in Figure 1. The system should be able to incorporate Δt values ranging from .01 second to 1 seconds.[1]

2.1 Contents of the Testbed

The testbed implements a RoboCup soccer field depicted on pages 210 and 215 of the text. The horizontal dimension of the field, x, is $[-100, 800]$ and its vertical dimension, y, is $[-100, 600]$. Units are centimeters. There are six landmarks located at: $(0, 0)$, $(310, 0)$, $(620, 0)$, $(620, 450)$, $(310, 450)$, and $(0, 450)$.

The testbed maintains the pose $\mathbf{x}_t$ of the agent. The agent lives in a 2D world and thus has location and heading. Accordingly, the pose $\mathbf{x}_t = (x, y, \theta)^T$, with x and y being the location of the robot, and $\theta \in [0, 2\pi)$ being the heading of the robot. East has value 0 and θ increases with counter clockwise rotation.

Control The testbed accepts velocity control commands, $\mathbf{u}_t$, of the form $\mathbf{u}_t = (v_t, \omega)^T$, where v_t is the translational velocity and ω_t is the rotational velocity. The maximum translational velocity of the robot is 15 cm/sec and the maximum rotational velocity is 10 degrees/sec.

Measurement The robot's sensor mechanism is an idealized range finger. The testbed models the physics of this range finder using a method similar to the beam model in Chapter of 6.3 of the text book. The range finder consists of 181 laser-beam lines with equal separation of 1 degree. The laser beam collision detection algorithm uses ray casting. The laser beam has a range between 100 and 300 cm. This is a parameter of the simulation setup. If a collision is not detected, then the scan does not return a measuerment. A measurement $\mathbf{z}_t = (z_t^1, z_t^2, \ldots)^T$ consists of a set of range measurement triples z_t^i. Each z_t^i is of the form $z_t^i = (r_t^i, \phi_t^i, s_t^i)$, where r_t^i is the range, ϕ_t^i is the bearing, and s_t^i is the signature.

Static objects in the testbed have unique sensor signatures and are therefore called landmarks. Landmarks in the testbed are vertical cylinders whose diameter is greater than zero.[2]

---

[1] You will be using a kinematic model rather than a dynamic model. Because of this, do not change the velocity commands at a high rate.

[2] In the robocup example in the text, the landmarks had zero width. If the laser beams also have zero width, then I

Noise You need to add noise at two places:

1. For control, you need to add noise to model $\hat{v}$ and $\hat{\omega}$ using equation (5.10) from the text.
2. For the beam model of the range finder, you need to add measurement noise, noise for unexpected objects, noise from failing to detect obstacles, and random measurements. This is described in Chapter 6.3.1 of the text.

Graphics Use 2D graphics. Display the testbed as if you were looking down at it from above. Display the robot indicating its location and heading, the static obstacles, and the laser beams. Display the obstacles as circles using a different color than the robot. Display the beliefs of the robot as uncertainty ellipses. For the model using the Kalman filter, this is done by obtaining the eigenvectors and eigenvalues of the belief covariance matrix. For the particle filter model, it is necessary to create a sample covariance matrix from the data. The laser beams should not penetrate or go through the obstacles.

Debugging Debug the testbed before you attempt to build the agent. Do this in stages. First, get the testbed running without noise in the testbed physics. In particular, give the testbed $\mathbf{u}_t$ commands and use the graphics to see if the robot does what is was told to do. Check to see if the laser beams project correctly for all directions of travel and that they step when they encounter obstacles.

Prepare the random number generators so that you can add noise to the testbed physics. Either use the Gaussian generator built into Java in class `Random` or use the Gaussian generator defined on p. 124 of the text. Test the random number generator to make sure you are doing what you think it is doing. Collect statistics on the random number generator and plot them in Matlab.

When both the noiseless testbed and the random number generators are working, then add noise to the testbed. You now must test the testbed once again. This time you have to collect statistics and plot them in Matlab. An alternative to test the velocity model with noise is the following:

1. Put the bot at some initial location.
2. Give it a velocity command and plot the resulting location of the bot with a dot.
3. Repeat steps 1 and 2 about 10 times and look at the distribution of points to see if it is reasonable.

Hints on implementing the laser range finder and detecting collisions with cylindrical obstacles.

1. Use the Pythagorean theorem to detect collisions with cylindrical obstacles.

---

don't see how a beam could detect a landmark.

2. For ray casting do the following (assume you are in the first quadrant). Extend r in increments whose size is a few centimeters. Use $r\cos\theta$ and $r\sin\theta$ to get the x and y components. Use the x and y components for the collision test with cylinders.

3 Structure of the Agent

3.1 What does the agent do?

The agent has the following components inside it.

1. The agent has a belief about the current state of the world. The initial belief is denoted $\mathrm{bel}(\mathbf{x}_0)$. Otherwise, it is denoted $\mathrm{bel}(\mathbf{x}_t)$, where t is the current time step.

2. The agent has a *control policy* to tell it what actions to take. For this assignment, the control policy is predetermined.

3. The agent has a complete and accurate map of the locations of the static objects in the world, and it has names for these objects. The map is denoted m.

4. The agent has a correspondence table to solve the data association problem. That is, it has a mapping from object signatures to object names that appear in the map. There are names for all static objects in the environment.

For a sequence of time steps, it issues a sequence of control actions to the testbed according to its control policy. It may, or may not, get a measurement reading, $\mathbf{z}_t$, for a particular time step. For each time step t, the agent updates its belief belief, $\mathrm{bel}(\mathbf{x}_t)$. If it receives a measurement $\mathbf{z}_t$, it updates its belief using m, c, $\mathrm{bel}(\mathbf{x}_{t-1})$, $\mathbf{u}_t$, and $\mathbf{z}_t$. Otherwise, it updates its belief without using $\mathbf{z}_t$.