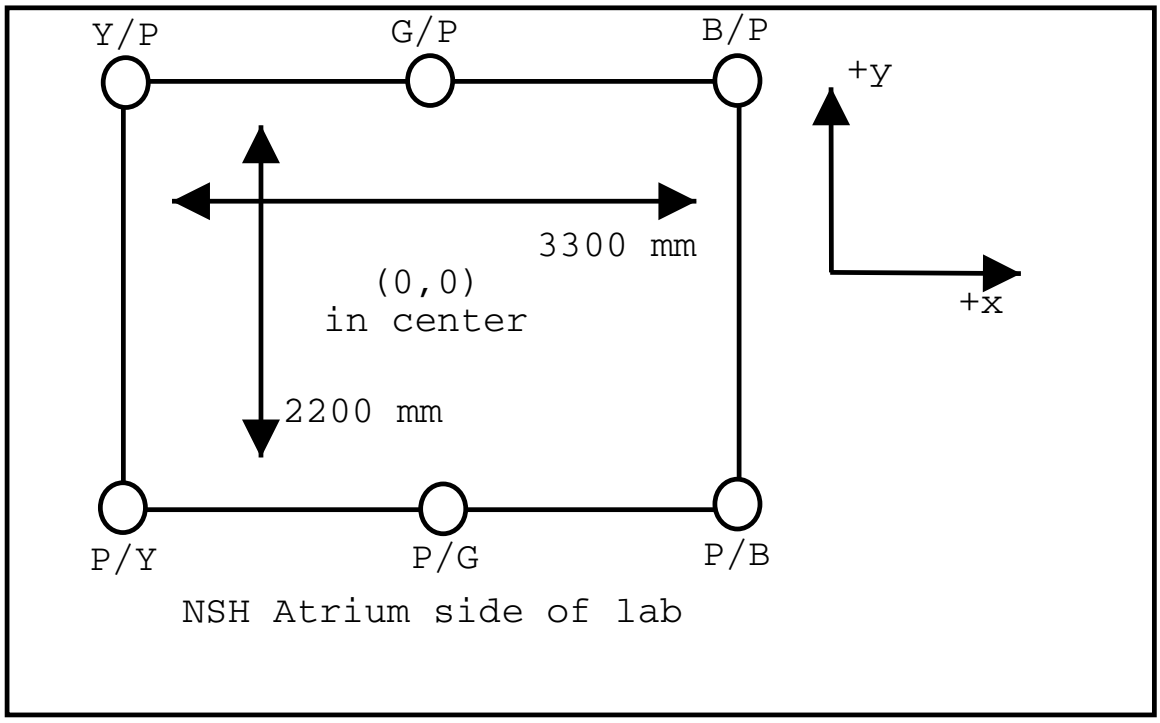
Homework 9 - due 11/12/2003

15-491: CMRoboBits

1 Introduction

In this lab you will experiment with a sample based localization method. You will write a behavior that uses the localization module, calculate the probability that you are in a region of the lab carpet, and experiment with different numbers of samples to represent the robot's belief.

2 Background



The one letter color abbreviations are as follows: P = Pink, Y = Yellow, G = Green, and B = Blue. Y/P should be read as "Yellow over Pink".

2.1 Retrieving Localization Information

You retrieve localization information from the FeatureSet. Subscribe to the FeatureSet as usual (see SpinDog for an example). FeatureSet contains a RobotPositionInfo structure called my_position. This structure is defined in agent/headers/MessageStructures.hh (note the odd extension). This structure in turn contains two Gaussian2 structures (two dimensional gaussians). Gaussian2's are defined in agent/headers/Gaussian2.h. The interesting bits are mean, sMaj, sMin, and thetaMaj. The mean field describes the mean of the two dimensional gaussian. sMaj and sMin give the standard deviations along the major (longer) and minor axes. thetaMaj gives the angle of the major axis (the 2d gaussian is not axis aligned - i.e. it is not at right angles with the XY coordinate grid).

The short version of the above is that you access your position with:

- FS->my_position.position.mean.x
- FS->my_position.position.mean.y

To get an idea of your uncertainty, look at:

- FS->my_position.position.sMaj;

This gives you the standard deviation of your samples along the direction with the most uncertainty. It's a good estimate of how lost the robot is in the XY plane.

To determine your heading, examine:

- FS->my_position.heading.mean

This is a vector2d that points in the direction the robot believes that it is facing. You can estimate your uncertainty in heading by examining the sMaj field, but this number tends to be unreliable.

Finally, there is an even shorter way of getting at the position and heading information if you do not care about uncertainty - you may examine:

- FS->my_position_body; // vector2d
- FS->my_angle; // double - angle in radians

2.2 Useful behaviors

The BeGotoPointGlobal and BeTrackObjects classes may be useful in this assignment. Both are declared in agent/Behaviors/BeLowLevel.h and defined in agent/Behaviors/BeLowLevel.cc. BeTrackObject will actively look for markers that the robot should be able to see given its current position. Create a pointer to a BeTrackObjects class in your behavior class. Then create an instance via new in your constructor. To use the behavior (assuming you called your pointer track_objects, FS is a FeatureSet*, command is a MotionCommand*) make the following call:

- (*track_objects)(FS, command, true, false);

The behavior will fill in the head command portions of command - be sure that you do not modify them later or track_objects won't be able to do its job.

Create an instance of BeGotoPointGlobal just like you created an instance of BeTrackObjects. Assuming that you call your instance goto_pt you use it by first setting a destination using the setTarget method and then by calling the class' () operator. The behavior never terminates - you need to manually evaluate your localization data and decide when you are "close enough" to your destination to stop calling goto point and do something else. Here's how you set a target:

- goto_pt->setTarget(x, y) or goto_pt->setTarget(x, y, theta)

X and y are coordinates on the field in millimeters. Remember that (0,0) is in the center. Theta is measured in radians. 0 causes the robot to line itself up along the positive x-axis. Pi causes the robot to line itself up along the negative x-axis.

Once you set the target point, you call goto_pt's () operator. This call fills in the portions of the motion command structure that concern walking. You should not overwrite these values if you want goto_pt to be effective. (Obviously you can send other values once your are in position)

- (*goto_pt)(FS, command);

2.3 Localization Internals

The files you will need to change are located in: dogs/agent/Localization/SRL. Specifically, look at: LocalizationEngine.h and LocalizationEngine.cc

The LocaleSampled structure contains a constant called numSamples that you will need to adjust. It also contains a flag that tells you whether or not the weights of the samples are currently normalized.

The LocalizationEngine class contains the instance of the LocaleSampled and it is called locale. You may use this instance to get at the actual samples. Also, the getPosition method gets called every cycle and is a convenient place to place code that needs to periodically examine the samples.

3 Procedure

3.1 Moving to a Target Point

Create a behavior called LocDog that does the following:

- When the front head button is depressed, the robot should stand in place for several seconds localizing and memorize its current position. This position is the Target Position.

- Whenever the robot is moved from the target position (we will not press a button or otherwise notify the robot - it must detect the move on its own based on localization data), the robot must return to its target position while avoiding a "forbidden zone" on the field.
- The forbidden zone consists of an 80 cm square centered on the origin - i.e. it extends 40 cm above the origin, 40 cm below it, and 40 cm out to either side of it. The robot must not enter this space.
- We will set multiple target points while testing the robot, so each time the front head button is pressed, it should be prepared to stop and memorize a new position.
- Your robot should send its confidence (as a percentage) that it is within the forbidden zone once every 25 frames. You may use a pprintf from within the localization code.

3.2 Varying the number of points

Briefly describe how the robot performs while running your behavior when using 10, 25, 50, 100, 200, and 400 points. Describe the tradeoffs when choosing the number of points to use. How do you think ambiguous sensor readings will affect the number of points required (ambiguous in the sense that the same or similar readings are possible from widely separated points).

4 Grading and Hand In

Hand in your LocDog.cc and .h files. Also hand in a memory stick image of your code for testing.

- 80 percent Behavior performance from 5 test points (we will watch the output of the certainty estimates for being in the target square in Chokechain - make sure they are turned on in the handin)
- 20 percent Description of varying point numbers and answers to the questions in that section.