Lab1: Behavior-Based Control

Due: Janurary 15th, 2009

The goal of this lab is to familiarize yourself with the development environment we will be using for the course, and to implement a behavior-based robot program.

1 Behavior-Based Programming

In this section, you will write a behavior-based program to move the robot around the environment in response to its sensors.

1.1 Setup

Before you start, you must read the "Swarm Robot Care and Handling" Handout, found on the course website under the "Handouts" section. After you have familiarized yourself with the proper way to handle and care for the swarm robots, you may get a robot from James to begin your lab work.

Next, you will want to become familiar with the skeleton starter code and the API available to you for Lab 1. Both the code and the Lab 1 API documentation is available in the subversion "distribution" repository. To check out the repository, we suggest that one member in each group take the following steps:

1. Create a folder on their Z drive called "robotics"
2. Create a subfolder on robotics called "SwarmBotAPI" and another one called "Labs"
3. Navigate to the "SwarmBotAPI" directory. Use TortoiseSVN to check out the API distribution code:
   https://cubist.cs.washington.edu/repos/cse481c_09wi_dist
4. Navigate to the "Labs" directory. Use TortoiseSVN to check out your group code:
   https://cubist.cs.washington.edu/repos/cse481c_09wi_{red, green, blue}
5. Select one member of your group to copy "Lab1.zip" from "Robotics\SwarmBotAPI" to "Robotics\Labs". Unzip this file to a "Lab1" directory in the "Labs" directory. Add this directory to the SVN repository, and commit.
6. All other group members preform a SVN update.

Once you have the repository checked out, you can open the documentation's main file, index.html, located in the lab1\doc folder. Once it is loaded in a web browser, click the "Files" tab, then click on "Lab1API.h" (do not click on "Code" unless you want to see the header source itself).

Once you are familiar with the available API, you can start working on the code by opening the "lab1.gpj" file, which will load the MULTI IDE that allows you to edit, compile, and debug code as described in the "Care and Handling" handout. You may use any editor to work on your code (we use Eclipse or Vim), but compilation and debugging only works in the MULTI IDE.

1.2 Move Forward

Once you are setup, let's drive the wheels. The goal for this part is to get the robot to move forward while still on the anesthetizer. Use the `TASK_MOVEFORWARD` case for this code. You can open a

terminal GUI window to see the motion of the wheels and the encoder values by typing `window motor` at the robot console.

1.3 Seek Light

Next, write a behavior that uses the light sensors and moves the robot towards light. The units of rotational velocity are milliradians per second, so you will need a large number to get responsive rotation. Use the `TASK_SEEKLIGHT` case for this code. It will be useful to use `cfprintf()` to display debugging information, such as the light sensor values. You can open a terminal GUI window to see the light sensor values by typing `window lightsensor` at the robot console.

Test this behavior by building a flash image and programming the robot. You can leave the serial port connected while the robot is running around, but not the JTAG connector.

1.4 Avoid Obstacles

Write a behavior that uses the bump sensors to move the robot away from obstacles. Use the `TASK_AVOIDBUMPS` case for this code. You can open a terminal GUI window to see the motion of the wheels and the encoder values by typing `window bumpsensor` at the robot console.

Test this behavior on the anesthetizer using the debug build, then build a flash image and test in the playpen. You can leave the serial port connected while the robot is running around, but not the JTAG connector.

1.5 Movin' Seekin' Bumpin'

Finally, put all three behaviors together to make a moving, seeking, bumping robot! Use the `TASK_MOVINSEEKINBUMPIN` case for this code. You should be able to get the robot through the maze from the start location to the goal location, the light.

2 Odometry Analysis

In this section, you will measure the accuracy of the robot's odometers and its ability to keep track of its pose in an external coordinate system.

2.1 Write the code

We will use the `TASK_IDLE` and `TASK_AVOIDBUMPS` cases for this code. In order to make collecting data easier, we will need to renumber the behavior modes at the top of the file to:

```
#define TASK_IDLE 0
#define TASK_MOVEFORWARD 3
#define TASK_SEEKLIGHT 2
#define TASK_AVOIDBUMPS 1
#define TASK_MOVINSEEKINBUMPIN 4
```

Use the `odometryPoseGet()` function to get the robot's current pose estimate, and use `cprintf()` to spew the values of the pose structure to the robot console.

2.2 Collect the Data

For each trial:

1. Switch to the `TASK_IDLE` behavior mode. This mode reset the pose to $0, 0, 0$, so start the robot at the same position in the playpen, and have it facing the positive x-axis.
2. Measure the starting position of the robot with the super-advanced two-dimensional measuring system. (Position is pose without the heading) You only need to do this once, and can mark this starting pose with blue tape.
3. With the robot plugged into the serial port, switch to the `TASK_AVOIDBUMPS` behavior mode.
4. Run the robot for the specified number of bumps.
5. Switch to the `TASK_IDLE` behavior mode. It might be easier to do this from the console by typing `beh 0`.
6. Measure the final position of the robot.

Conduct experiments with 0 - 4 bumps, with at least three trials each. The dimensions of the playpen are $8' \times 8'$.

2.3 Write-up (Done individually)

Use Excel or MATLAB of Java or Python or whatever you want to make a graph of the position error vs. number of bumps. Make a **one page** document with a plot of the data and a brief explanation of the measurements you made and the sources of error. Pontificate as to the usefulness of odometry measurement in robots in general, like in the video of the map-making robot team. What about these robots in particular affects their performance the most?