Issued: September 26, 2005
Due: Wednesday, October 5, 2005

2.12 Introduction to Robotics
Problem Set 2: Robot Programming

The goal of this problem set and the associated lab sessions on Thursday, September 28th and Friday, September 29th, is for you to develop a mobile robot motion planning and control algorithm for a simulated de-mining robot.

1. First, using matlab, write an algorithm to generate a series of waypoints that will cover a 5 by 5 meter area, (the "box.txt" environment in simple_sim, our de-mining robot simulator) using each of three different strategies: (a) back-and-forth ("mowing the lawn") motions, (b) spiraling, and (c) random motions. Your program should write the waypoints out to a file, in the form of (x, y) coordinates in two columns. Read in the data from the file and generate a plot of the waypoints. (Don't specify waypoints too close to one another; a separation for example of at least 0.5 meters is desirable.)

2. Next, write a C program to generate the waypoint files that you generated using matlab in part 1.

3. Next, run simple_sim on an athena linux workstation (we'll show you how in Lab). First, teach yourself to manually steer the robot around a simple environment with four mines. Perform two or three runs where you create a data log file for manual control to activate all the mines, and load and plot the data to reconstruct the trajectory that you manually executed. Details for the data logging format will be provided in lab. Plot both the "true" (x, y) trajectory based on the simulated robot state (columns 2 and 3 of the data file), and the dead-reckoned robot trajectory computed for you by the simulator by integrating the encoders on the robot wheels. Compare the two trajectories. Try to do this for two different motion control strategies (e.g., conservative vs. agressive velocity control). How quickly can you reach all the mines?

4. As discussed in class, simple_sim in with a pre-built trajectory controller, that can read in a set of waypoints and then systematically perform trajectory control to try to reach each of the waypoints in turn (assuming no obstacles!). Run simple_sim in this mode providing the waypoint files that you automatically generated above in part 1 to see how the waypoint controller of the simulator performs on your waypoint lists. Does it do a good job? Can you think of ways to do better?

5. Now, add your own C code to the file `user_code.cpp` to implement your own trajectory control algorithm in simple_sim. To start, try to simply to integrate your waypoint generator with a simple waypoint controller, and see how it performs. Log data and plot the results in matlab. How does your controller compare to the built-in simple_sim controller tested in part 4, and to the results that you obtained under manual control?

6. (Optional): Develop a more complex motion controller that would be capable of running in a more complex environment with numerous obstacles (such as "maze.txt") and/or could achieve good coverage despite large amounts of dead-reckoning error. For example, can you implement a finite state machine that switches between the three modes of long transits, bouncing off walls, and spiraling motions, that seems to be the mode of operation of the roomba robot vacuum cleaner? Does your code out-perform a simple controller in a more cluttered environment? How fast can you find all the mines? (If you do not feel that you have the prior C programming experience to attempt this, then feel free to sketch out a potential solution strategy on paper.)