ME 132 a: Lab 2

Shir Aharon
Tiffany Huang
Steven Okai

March 11, 2013

1 Part I: Point-to-point Control

Notes on how to run code:

The mainloop for this section is in the `lab2_part1.cc` file, with helper functions in the `lab2_shared.h` file. For interfacing with the Player/Stage system, we reuse the code from the previous lab in `common_functions` and `cmdline_parsing` files.

The code is currently hardcoded to expect a file named `goal_points.txt` which consists of the (x, y) points to which the robot should attempt to drive to.

1.1 go_to_point() function

For this function, we start by computing the relative angle that the robot needs to turn, and the distance it needs to drive. The `theta_dot` returned is half of the remaining distance to turn, which is done to prevent overshooting the necessary rotation amount before the next time step. The `r_dot` calculation is a little more complicated. First, if the current direction is more than 22.5° off, then the robot is told not to move and just turn. This prevents motion in the incorrect direction. Now again to prevent overshooting the destination or moving too far off from the direct line connecting the current robot location and the goal point, the speed returned is at most the distance to travel. This is divided by 10 for overshoot reasons, but can be scaled back up by the log of the angle offset. Thus the closer to being pointed directly at the goal point, the faster we can safely move.

These parameters were determined somewhat by trial and error. The log scaling for the speed was done in an attempt to move as fast as was deemed “safe”. The constraints on the turning speed and the 22.5° restriction come from experimentally attempting to drive around the map without colliding into walls. In particular, this configuration seemed to allow driving in the center of a 1m wide path on the occupancy map.

1.2 Simulation Component

The code for this is in the `lab2_part1.cc` file. It simply connects to the robot and then iterates through the goal points given in a loop that checks if close enough to the current point, and then getting updating speed/direction values.

1.3 Lab Component

Demoed in lab.

2 Part II: Occupancy-Grid Maps

Notes on how to run code:

The mainloop for this section is in the `lab2_part2.cc` file, with helper functions in the `lab2_shared.h` file. Again, for interfacing with the Player/Stage system, we reuse the code from the previous lab in the `common_functions` and `cmdline_parsing` files.

As before, the code is currently hardcoded to expect a file named `goal_points.txt` which consists of the (x, y) points to which the robot should attempt to drive to.

Graphics Method

We used the built in graphics abilities of the Player/Stage system. In order to do this, we modified the `occupancy.cfg` file to add `"graphics2d:0"` to the `provides` list of the driver for the robot model `r0`. For some reason however, this caused problems with the robot connection function `check_robot_connection` from the lab 1 code, so this was taken out. In simulation, the system still connected to the robot and worked correctly.

2.1 occupancy_grid_mapping() function

2.2 Simulation Component

2.3 Lab Component

Demoed in lab.

2.4 Group of 3