CIS 390 Fall 2015
Robotics: Planning and Perception
Kostas Daniilidis
Project 3

1 Introduction

This is the final project, where you will combine parts of your code from previous projects and homeworks. You will be making your robot move through a 'maze' with a known map, localizing yourself using the particle filter from project 2 (step 1) and planning using Dijkstra's from project 1 phase 1 (step 2). There is a lot of flexibility in this project as long as the robot completes the maze.

2 Step 1: Localization

Since we provide you a map of the environment, you already know the exact configuration of the environment. However, you will need to recover your initial position in this map, thus localizing yourself. To do so, you will use the particle filter code from Project 2. To initialize your particle positions, you may distribute your particles around an estimate of the true position (e.g. a Gaussian distribution centered at the true position in the simulation). For the first N measurements you receive (with N as a tunable parameter), the robot should stay in its initial position, while localizing itself with the particle filter. In the initial position, there will be no ambiguity in the measurements. By the end of this step, your filter should have converged to a single location.

3 Step 2: Navigation

Knowing your initial location, you will have to navigate through this environment. We provide a discretized map of the environment, which means that you can use a navigation algorithm you are already familiar with (for example Dijkstra's) to move around, avoiding any obstacles. The map is represented as a graph (self.graph in the code), with the first and last node being the initial and goal position (within the map). Using the control laws we used in previous project phases, you can move between cells of the discretized space. You will start somewhere outside the occupancy map (position after localization procedure) and the first movement will be to get to the initial position of the map. Be careful that when going between two points you do not go too far off the path or else you could run into obstacles. As a practical advice, for two points a and b that you visit sequentially, instead of waiting to reach a to start moving towards b, you might find it useful to direct towards b when you are within a small radius close to a.

4 Simulation

We provide you with simulation code in `FinalControlSim.py` to allow you to test your implementation. The code handles the generation of the navigation graph given the occupancy grid for you. You just have to run your implementation of Dijkstra's or A-star to get your path. The specific variables you will need are:

- `occupancy_map`: This is a $n \times n$ representing the cells of the space that are occupied with obstacles. Free cells are represented with 0, and occupied cells with 1. One is given for you, if you want to test more, feel free to change it.
- `world_map`: This is as in the Particle Filter simulation of the previous project. It is a list of lists with (x, y, θ) of the April tags.

- `pos_init`: This will be the position and orientation of your starting point (outside the map), represented as (x, y, θ) in a numpy array.
- `self.graph`: This is the graph generated from the `occupancy_map`, which you will use to navigate the world. Each node corresponds to the center of the cell of the occupancy map. After you localize yourself (position `pos_init`) you should move at the location of the first node of the graph and follow the planned path until you reach the last (goal) node.

You are given some hints in the `command_create` function to help out with the initial localization.

5 Grading

This will be graded on completion.