Homework 4

Robot Autonomy
CMU 16-662, Spring 2018
Due Date: 19 April, 11:59 pm

TAs: Anirudh Vemula, Fahad Islam

1 Introduction

In this homework you will combine the tasks from the first three assignments to create a planner that moves HERB to pick up a bottle from a table.

2 Planning for the Mobile Base

First, you will extend the A-star planner you implemented in Homework 3 to plan in position and orientation for Herb. First copy your implementation of **DiscreteEnvironment.py** from Homework 3 into your code directory. Modify the implementation to allow a different resolution to be specified for each dimension. In other words, the *resolution* parameter passed into the `init` function will now be a 3-dimensional vector specifying the x, y and θ resolutions rather than a single scalar.

Unlike the PR2 robot, base movements for Herb must obey non-holonomic constraints. We can model the system using differential drive dynamics:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{r}{2}\cos\theta & \frac{r}{2}\cos\theta \\ \frac{r}{2}\sin\theta & \frac{r}{2}\sin\theta \\ -\frac{r}{L} & \frac{r}{L} \end{bmatrix} \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix}$$

In the above equation, θ is the current orientation of the robot, r is the radius of the wheel, L is the distance between the wheels and ω_r and ω_l are the left and right wheel velocities. For Herb, $r = 0.2$ and $L = 0.5$. The maximum wheel velocity that can be applied to either wheel is $\omega = 1$.

In **SimpleEnvironment.py**, complete the function *ConstructActions* which precomputes a set of actions and associated footprints for the robot. At the top of the function you will see the definition for a *Control* and an *Action*. A *Control* is composed of left and right wheel velocities and a duration. An *Action* is composed of a control and a footprint. The footprint is a list of configurations the robot will visit during execution of the action.

Select a set of controls and generate the associated actions within *ContructActions*. As an example, $\omega_r = \omega_l$ will drive the robot forward, $\omega_r = -\omega_l$ will turn the robot in place. Use the helper function *GenerateFootprintFromControl* to create the footprints. Generate a footprint for every action at each possible starting orientation.

Next, implement the *GetSuccessors* function in **SimpleEnvironment.py** to compute successors using the actions you defined in the previous step. Copy over your *AStarPlanner.py* implementation from Homework 3. Update the *Plan* function to work with the new *GetSuccessors* function. Additionally, modify the function to return the plan as a list of *Action* objects. You may make any changes to the return value of the *GetSuccessors* function you need to achieve this.

We have provided the file **test_base_planner.py** to help you test your updated planner.

3 Bottle Grasping

Next we will try to make Herb grasp the bottle from the table. Copy over any of the RRT-based planners you have implemented so far. In addition, copy over the matching version of **HerbEnvironment.py**. You will use these to plan for the arm.

In the file **GraspPlanner.py**, implement the *GetBasePoseForObjectGrasp*. This function should return a base configuration in $SE(2)$, *base_pose*, and a 7-DOF arm configuration, *grasp_config*, such that if the robot is placed at *base_pose* and the arm is put in configuration *grasp_config* the robot can grasp the bottle without being in collision with the table.

Copy over your grasp ordering code from Homework 1 so that you pick the best grasp possible that is collision free. To select a base pose you can use the inverse reachability functionality built into OpenRAVE or come up with your own method of finding a base pose.

The function *PlanToGrasp* in **GraspPlanner.py** has been implemented for you. It will utilize the results of *GetBasePoseForObjectGrasp* and your arm and base planners to first plan a path for the base to the desired pose, then plan a path for the arm to execute the grasp and lift the bottle off of the table.

The **run.py** can be used to test your code. Edit this file to set the arm planner to the planner you have selected. We have defined three different test locations for the bottle. You can select a test using the *test* argument:

```
python run.py --test 1
```

4 Deliverables and Grading

Please turn in a zip file (named `hw4-<groupid>.zip`) containing your code, a PDF writeup and the videos mentioned below **on Gradescope**. Only one person per group needs to submit but please make sure everyone's name and andrewid is on the pdf (**and add them as collaborators on Gradescope**). The following shows the point breakdown:

1. List the set of controls you used to generate your action set. How did you select this set? Use the function *PlotActionFootprints* to plot footprints for each orientation. Include a copy of the plots in your writeup. (5pts)
2. Use the file **test_base_planner.py** to test your base planner with 3 different start and goal configurations. Submit a video of the execution of these three plans. What heuristic did you use in your planner? (5 pts)
3. Describe your method for selecting a base pose. Run the full grasp planner for each of the three placements of the bottle. Submit a video for each run. (10 pts)