

EECS 598: Motion Planning

Winter 2019

Homework Assignment #1

Due 1/28/2019 at 1:29pm

Rules:

1. All homework must be done individually, but you are encouraged to post questions on Piazza.
2. No late homework will be accepted.
3. You must show all your work to receive credit for your solutions.
4. The goal of this homework is to familiarize you with openrave and some motion planning concepts.
5. Submit your python code along with a pdf of your answers in a zip file to Canvas.

Questions

1. (10 points) Chapter 4 exercise 5
2. (15 points) Chapter 4 exercise 8
3. (10 points) Chapter 4 exercise 14
4. (10 points) Chapter 4 exercise 16

Software

1. (15 points) Install Ubuntu 16.04 and Openrave on your computer. Follow the directions [here](#). Include a screenshot showing that the fastgrasping example runs successfully in your pdf. The screenshot should be similar to the one in the installation directions.
2. Complete the Numpy tutorial [here](#). You may need to use Numpy throughout the course when implementing code.
3. Download and unzip [HW1files.zip](#). Run drawing.py. You should see the PR2 robot in an environment with a doorway and some tables in the openrave viewer. If you do not see this, openrave is not installed correctly (go back and install it). You may also see warnings in the terminal that look like:

```
[WARN] [kinbody.cpp:1504 KinBody::SetDOFValues] dof 30 value is not  
in limits -2.321000e+00<-2.121308e+00
```

These are numerical precision issues and you can ignore them. You may also see errors/warnings related to Coin and SoQt (these are graphics packages). You can ignore these, too.
4. You will need to consult the [openrave.py documentation](#) frequently to complete this assignment.

Important Concepts in Openrave

Below are important concepts in openrave, some of which may be useful for completing the homework.

- **Locking the environment:** You can grab the lock on the environment data structure (what is displayed in the viewer) using the `with env:` statement (where `env` is an instance of the `Environment` class). See the `simplmanipulation.py` example to see how to use this. It is important to use this lock so that other openrave threads don't interrupt what you're doing and change variables you may be working on. Keep in mind that as long as you are inside a `with env:` block, changes you make to the robot (e.g. setting the configuration) may not be visible on the display. It is only after you exit this block and wait some amount of time that the display has the chance to refresh and show what you modified. However, the changes you make inside `with env:` still take place in the data structure. For instance, you can check a collision inside a `with env:` block by setting the configuration and calling `check collision`. Although you may not see the display update, the robot is actually at the new configuration and the collision check is taking this into account. After you exit the `with env:` block and wait, you will see the robot rendered at the new configuration.
- **Active DOFs:** A robot can have many degrees of freedom (e.g. a humanoid), but you may not want to plan for all of those DOF at the same time. For instance, you may want to drive the mobile base of a robot but not move the robot's arms. In openrave, you can select which DOF you want to plan for by using `robot.SetActiveDOFs(...)` (you can look up this function to see how it works). From then on, you can `robot.SetActiveDOFValues(...)` to set only the DOFs of the robot you care about. For example, `robot.SetActiveDOFs([0,5,6])` will set the 0th, 5th, and 6th DOFs to be active. Then you can do `robot.SetActiveDOFValues([pi/2,pi,0])` which will set the 0th joint to $\pi/2$ radians, the 5th joint to π radians, and 6th joint to 0 radians.
- **Controllers:** In openrave you often set a configuration for the robot to test something about it. For instance, you set a configuration and then test if it is in collision. However, you don't always want the robot to go there (remember that openrave can directly control a real robot). So we can use commands such as `robot.SetActiveDOFValues(...)` to set a configuration for the robot for testing something out. If we really want the robot to go there, we have to tell that to the robot's controller. For instance, `robot.GetController().SetDesired(robot.GetDOFValues())` will tell the robot to go to whatever configuration is set on it (in simulation the robot moves to this configuration instantly, on the real robot this of course takes time). It is important to note that the controller runs in a different thread from your main code (this is so you can do things while the robot is moving). Thus, if you want to move the robot somewhere, after you tell the controller where to go, you need to wait until the controller is done. This can be done by polling this function: `robot.GetController().IsDone()`. Note that when you execute a planned path, you also need to use the robot's controller and so you need to wait for that to complete by polling the same function.
- **Viewer:** The openrave viewer comes with some handy functionality to allow you to zoom in and out and focus on different parts of the scene as well as move things around. The text below is specific to the Coin viewer. To toggle between panning mode and selecting mode, push `Esc`.
 - **Panning Mode:** To zoom in/out use the scroll wheel on your mouse. To focus on a part of the scene, click on the viewer window and then push `s`. Then click on the part of the scene you

want to focus on. You can rotate around that point by holding down the left mouse button and dragging the mouse. You can translate the camera by holding down the scroll wheel and moving the mouse.

- Selecting mode: Left click on an object to select it. There will be a green box around the object if it is collision-free and an orange box if it is in collision. Once you select an object drag the mouse to move it around. You can also `Ctrl + Click` on a link of a robot to move that link's joint. When you `Ctrl + Click` a wheel will appear. Drag the wheel with the left mouse button to move the joint.

Implementation

The following implementation problems should be done in python starting from the provided templates. Only edit what is inside the `### YOUR CODE HERE ###` block. Feel free to look around the internet and the openrave installation for example code for reference, especially in the openrave examples, but you should implement your own code from scratch unless explicitly stated otherwise.

1. (5 points) Run the `drawing.py`, it should load the PR2 robot in an environment with several tables and a doorway. Using your mouse to control the viewer, move all the tables to be on the PR2's side of the room. Record their position and rotation (using the display in the GUI). Now edit `drawing.py` to place the tables automatically at the poses you recorded. Include a screenshot of the scene in your pdf. Save the code as `tables.py` and include it in your zip.
2. (20 points) Edit a clean copy of `drawing.py` to load the Puma robot arm into the environment and place the Puma near the PR2 so that the two robots both have their bases on the floor but are not touching. Set joint values on the PR2 robot so that the PR2 touches the Puma with any part of its left gripper. You may determine the joint values to do this through experimentation. Confirm that the two robots are touching by printing out the result of a `CheckCollision` command between the two robots. Include a screenshot in your pdf. Save the code as `pr2puma.py` and include it in `HW1.zip`.
3. (10 points) Edit a clean copy of `drawing.py` to draw a set of red rectangles around the boundaries of every table in the environment. You may use a series of lines to draw the rectangles. Next edit the template to draw 35 blue points in a circle that encompasses the environment shown in the scene. Make sure the points are large enough to be easily visible in a screenshot. (see `.../openravepy/_openravepy_0.9/examples/tutorial_plotting.py` for drawing examples). Include a screenshot showing the points and the rectangles in your pdf. Save the code as `linespoints.py` and include it in your zip.
4. (15 points) Edit a clean copy of `drawing.py` to place the PR2 facing the wall in the center of the environment so that the wall is close to the robot but not touching it. Confirm that the robot is not colliding with the wall by printing out the result of a call to `env.CheckCollision(robot)`. Set the 7 joints of the PR2's right arm as active (all others should be inactive). Inside a `with env:` block, use the `SetActiveDOFValues` function to set a configuration for the arm so that it is pointing straight forward. Call `env.CheckCollision(robot)` to confirm the robot is colliding with the wall in this new configuration and print out the result. Now use the `SetDesired()` function on the robot's

controller to command the robot to go to the new configuration. Use `waitrobot()` (which is defined inside `drawing.py`) after `SetDesired()` to ensure the robot reaches the target configuration. Be careful in deciding if `waitrobot()` should be inside or outside the `with env:` block. Explain why you placed `waitrobot()` inside/outside the `with env:`. What would happen if it were placed in the other location and why? Include your answers in your pdf. Take a screenshot of the robot in the new configuration in the viewer and include it in your pdf. Save the code as `collision.py` and include it in your zip.