HW2 - Robot Autonomy Plus

Professor: Oliver Kroemer, TA's: Steven Lee, Ami Sawhney

Due: 12/4/12 at 11:59pm Eastern Daylight Time (GMT-4))

1 Introduction

This homework will focus on collision avoidance and motion planning discussed in lecture. You will first implement a collision detection algorithm, and then use it to determine if two cuboids are intersecting. You will then implement a Probabilistic Roadmap (PRM) as discussed in lecture and use it to plan a collision-free path from a starting to target joint configuration. Please review the lecture slides from Lecture 6 prior to starting this assignment, as you will find helpful information in the slides for your implementations.

2 Cuboid-Cuboid Collision Detection

In the first part of the homework, we will implement a collision detection algorithm for later use in the second part of the homework, which involves motion planning. Collision avoidance is a critical aspect of robot motion planning, as we want our robots to operate safely around objects, other robots, and humans. A first-order approach for collision avoidance is to define cuboids around objects (also known as rectangular prisms) and detect if these cuboids intersect. In this context, these cuboids are also referred to as bounding boxes.

For our collision detection algorithm, we recommend implementing the Separating Axis Theorem to determine when two cuboids intersect. We will only use cuboids to parameterize collision geometry in this assignment.

For this assignment, we will parameterize a cuboid through specification of 1) the pose of its centroid, which serves as the cuboid local frame; and 2) the specification of the cuboid's lengths in the local x-, y-, and z-dimensions. We use the standard roll-pitch-yaw convention for orientation, where the (r, p, y) tuple represents the angle about the body's x-axis (r, roll), y-axis (p, pitch), and z-axis (y, yaw), respectively).

The function BlockDesc2Points(H, Dim) we provided to you in RobotUtil.py takes in an object pose (specified by a homogeneous transformation matrix, H) and object dimensions (length(x)/width(y)/height(z)) and returns the bounding box of the object and the axes given by the rotation matrix.

To complete this part of the homework, please implement the Separating Axis Theorem. More specifically, please implement the function **CheckPointOverlap()** and **CheckBoxBoxCollision()** in RobotUtil.py.

Then, for the following cuboids in Table 1, provide a yes/no response for whether the test case cuboid is colliding with the reference cuboid. The reference cuboid is the cuboid defined as follows (i.e., with centroid pose coincident with the world frame):

Origin (m) (x, y, z): (0, 0, 0)
Orientation (rad) (r, p, y): (0, 0, 0)
Dimensions (m) (d_x, d_y, d_z): (3, 1, 2)

Consider cases where cuboid faces are touching in the same plane to be a collision, as well as cases where one cuboid completely encloses another cuboid.

Table 1: Test cases for cuboid-cuboid collision checking.

'''	Test Case	Origin (m): (x, y, z)	Orientation (rad): (r, p, y)	Dimensions (m): (d_x, d_y, d_z)
False	1	(0,1,0)	(0,0,0)	(0.8,0.8,0.8)
True	2	(1.5,-1.5,0)	(1,0,1.5)	(1,3,3)
True	3	(0,0,-1)	(0,0,0)	(2,3,1)
True	4	(3,0,0)	(0,0,0)	(3,1,1)
True	5	(-1,0,-2)	(.5,0,0.4)	(2,0.7,2)
False	6	(1.8,0.5,1.5)	(-0.2,0.5,0)	(1,3,1)
True	7	(0,-1.2,0.4)	(0,0.785,0.785)	(1,1,1)
True	8	(-0.8,0,-0.5)	(0,0,0.2)	(1,0.5,0.5)
'''				

Deliverables:

- A list of yes/no responses for whether the cuboids in Table 1 are colliding with the reference cuboid.
- Your implemented cuboid-cuboid collision detection algorithm code, along with any scripts you may have used to generate your output for the cuboid collision test cases

3 LocoBot PRM Motion Planning

In this section of the homework, you will implement a **Probabilistic Roadmap (PRM)** motion planner in joint space. This planner should provide a feasible, collision-free path from an initial joint configuration to a given target joint configuration.

Table 2: Initial and final joint targets for motion planning.

Joint name	Initial Value	Final Value
joint_1	$-80°$	$0°$
joint_2	$0°$	$60°$
joint_3	$0°$	$-75°$
joint_4	$0°$	$-75°$
joint_5	$0°$	$0°$

The motion planner should plan in joint space for joints 1 through 5. We are not concerned with moving the gripper fingers in this assignment, so PRM planning for joint 6 and joint 7 is not necessary. Your plan should also have a smooth trajectory when you visualize it (i.e. you will probably need to interpolate between points in the path).

You will also use the cuboid collision detection code from part 2 of this homework to perform collision detection. This includes collisions with the obstacles in the scene and self collisions with the robot. We have given you the bounding boxes for each of the links of the Locobot arm in the file Locobot.py. The variable **self.Cdesc** contains the poses (xyz and roll/pitch/yaw) of each of the robot arm bounding boxes that you will use to define the transforms for each link. Please note that you can ignore the first link of the arm for the collision detection since it only rotates, which is why we only gave you 4 bounding boxes instead of 5.

Additionally, we have given you the bounding boxes for the obstacles in the scene, the robot base, and the camera mount. We will consider the robot base and camera mount as obstacles in the environment since we will not be moving the robot base at all for this homework. These bounding boxes are defined for you in the PRMGenerator.py file. **You will need to use your ForwardKin() implementation from Homework 1 to perform the collision detection.**

You should implement your PRM sampling and generate collision-free edges in **PRMGenerator.py**. This code provides you with the bounding boxes for all the obstacles present in the environment (pointObs and envaxes). Once you've sampled poses and generated collision-free edges, you can then save your prmVertices, prmEdges, pointObs, and axesObs to a file (code provided in **PRMGenerator.py**).

Next, query your PRM planner to generate a feasible collision-free path from start to goal joint configurations in **PRMQuery.py**. Once you've generated a joint space path, you can visualize it using the PlotCollisionBlockPoints() method of the Locobot class.

Next, you will visualize your planned path using the Locobot in VREP. The section below describes this further.

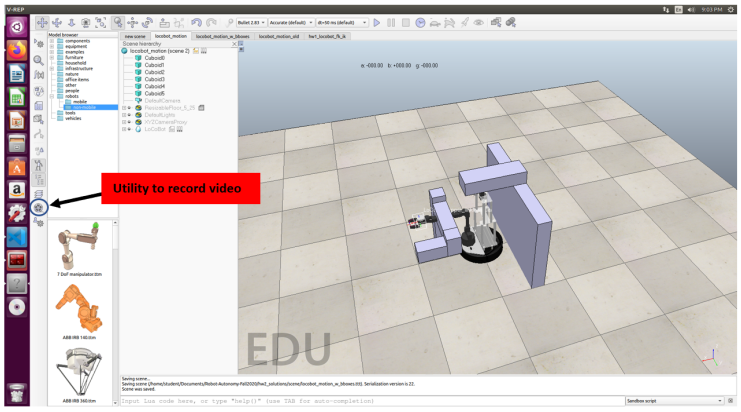
Visualizing the Output of Your Planner in VREP using Pyrobot:
PRMQuery.py has a command line argument of –use_pyrobot, that determines whether the visualization of your path will be in Pyrobot (using VREP) or simply plotting with Python's Matplotlib. The default argument is False.

To test your planner in VREP with the environment obstacles we provided to you, start a roscore and then run **PRMQuery.py** with the command line argument –use_pyrobot = True (don't forget to source the pyrobot virtual environment first). This will start the VREP simulator and load in the Locobot. Note that because of the way Pyrobot initially loads the Locobot into the simulator, the Locobot arm will initially be intersecting one of the blocks in these scene. You can ignore this during the initial loading, because once the robot arm is commanded to go to the starting joint configuration (with the command robot.arm.set_joint_positions()) it will start in the correct location. You should use the path outputted by your PRM planner and use the robot.arm.set_joint_positions() command in the collision_detection.py script to visualize your planned joint space path in VREP.

Deliverables:

- A small, concise paragraph (2-4 sentences) describing your PRM planner.
- A time history plot for joints 1 through 5 that show the path produced by your PRM. Include the target joint position on each plot as determined by the motion planner. Your plots must include labels and units on each axis.
- A video of the LoCoBot executing the plan outputed by the PRM. You can use a screen capturing program or the video recording capabilities within V-REP. If your video is large, please provide a link to it through Google Drive, Dropbox, or some other cloud hosting service so that we may access it later.

Note: Shown below is the icon you can use to record a video in VREP.



300 iterations gave a plan of 4 points. Interpolating between these points seem to result in a plan with collision. Plan seems to not collide without interpolation.

2500 iterations gave a plan of 9 points. Seems to collide without interpolation but does not with interpolation.

Can be seen in videos attached.

Perhaps the robot never does reach the target joint position due to the lack of time, resulting in nondeterministic behaviours