16-662 Assignment 2:
Motion Planning & Collision Avoidance

Instructor: Oliver Kroemer
TAs: Timothy Lee and Mohit Sharma

Due Date: Wednesday, February 27, 2018 at 11:59pm

Version 2. Content edits are marked in red underline.

Instructions

1. **Integrity and collaboration:** If you work as a group, include the names of your collaborators in your write up. Code should **NOT** be shared or copied. Please **DO NOT** use external code unless permitted. Plagiarism is strongly prohibited and may lead to failure of this course.
2. **Start early!**
3. **Piazza:** If you have any questions, please look at Piazza first. Please go through the previous questions before submitting a new question and adding its tag in the sticky.
4. **Write-up:** Please note that we **DO NOT** accept handwritten scans for your write-up in this assignment. Please type your answers to theory questions and discussions for experiments electronically.
5. **Code:** Please stick to the function prototypes mentioned in the handout. This makes verifying code easier for the TAs.
6. **Submission:** Please include all results in your writeup pdf submitted on Canvas. For code submission, create a zip file, `<andrew-id>.zip`, composed your Python or C++ files. Please make sure to remove any temporary files you've generated.

1 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.[1] We use the Tait-Bryan ZYX parameterization of orientation (i.e., the standard roll-pitch-yaw convention, 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).

To complete this part of the homework, please implement the Separating Axis Theorem. 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.

Tip: You will use your implementation later in the assignment for motion planning with V-REP. For this reason, you may find it easier to integrate with V-REP if you implement your collision checking in Python. You may implement collision checking in C++, but you would need to create your own Python bindings to integrate your code with the provided `vrep_utils.py` library.

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)
1	(0, 1, 0)	(0, 0, 0)	(0.8, 0.8, 0.8)
2	(1.5, -1.5, 0)	(1, 0, 1.5)	(1, 3, 3)
3	(0, 0, -1)	(0, 0, 0)	(2, 3, 1)
4	(3, 0, 0)	(0, 0, 0)	(3, 1, 1)
5	(-1, 0, -2)	(.5, 0, 0.4)	(2, 0.7, 2)
6	(1.8, 0.5, 1.5)	(-0.2, 0.5, 0)	(1, 3, 1)
7	(0, -1.2, 0.4)	(0, 0.785, 0.785)	(1, 1, 1)
8	(-0.8, 0, -0.5)	(0, 0, 0.2)	(1, 0.5, 0.5)

[1] To be abundantly clear, a cuboid with dimensions in meters of (1, 1, 1) will have a volume of 1 m^3, not 8 m^3.

Deliverables:

1. A list of yes/no responses for whether the cuboids in Table 1 are colliding with the reference cuboid.
2. 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.

2 LocoBot Arm Bounding Boxes

Before motion planning with the LocoBot arm, we must first define collision geometry for the arm. Similar to the first problem we will now define cuboids that approximate the collision geometry for the the robot arm. We have already defined these bounding boxes for you in the scene file. These are named using the convention **<link name>_collision_cuboid**, e.g., *elbow_link_collision_cuboid*. Please locate these bounding boxes in the scene file. You will use these bounding boxes for Question 3 to ensure your motion planning solution is feasible.

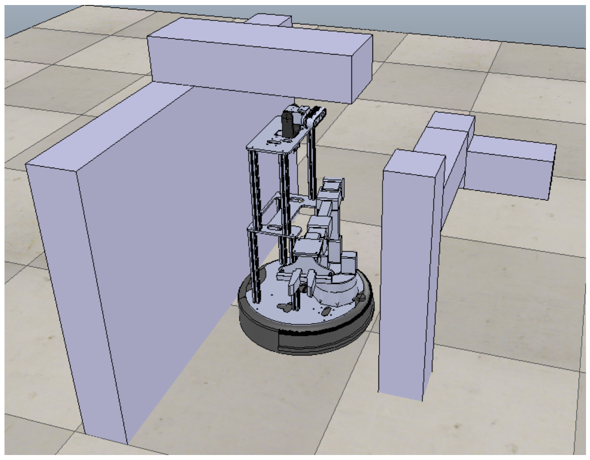
We also have provided additional functions in vrep_utils.py that allow you to query the location, orientation and bounding boxes for objects in the scene. For this part of the homework you need to implement a simple function that allows you to query the above properties for each of the collision bounding boxes defined for the robot arm. This can be done by first obtaining the cuboid handles by their name — see `get_arm_joint_handles` as an example — then using the cuboid handle with the new functions. Look at Table 2 and make sure you're able to get the location, orientation and dimensions for all the collision bounding boxes for each link.

Table 2: Definition of the LocoBot arm bounding boxes.

Link name	**Base joint name**	**Origin (m):** (x, y, z)	**Orientation (rad):** (r, p, y)	**Dimensions (m):** (d_x, d_y, d_z)
`arm_base_link`	`arm_base_link_joint`	TBD	TBD	TBD
`shoulder_link`	`joint_1`	TBD	TBD	TBD
`elbow_link`	`joint_2`	TBD	TBD	TBD
`forearm_link`	`joint_3`	TBD	TBD	TBD
`wrist_link`	`joint_4`	TBD	TBD	TBD
`gripper_link`	`joint_5`	TBD	TBD	TBD
`finger_r`	`joint_6`	TBD	TBD	TBD
`finger_l`	`joint_7`	TBD	TBD	TBD

Deliverables:

1. **None**. There are no deliverables for this part of the assignment. However, make sure you're able to query V-REP to progammatically obtain the collision geometry for the robot links for use in Question 3.


(a)

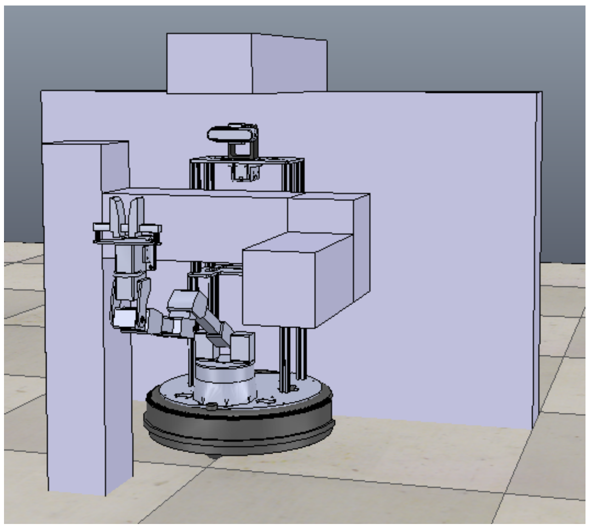

(b)

Figure 1: Visualizations for (a) initial and (b) final arm configurations.

3 LocoBot PRM Motion Planning

In this part of the homework, you will implement a ***Probabilistic Roadmap (PRM)*** motion planner in joint space. This planner will provide a feasible path to a given target joint configuration, and your `ArmController` class that you implemented in Assignment 1 will be used to execute this path.

The motion planner should plan in joint space for arm joints `joint_1` through `joint_5`. We are not concerned with moving the gripper fingers in this assignment, so PRM planning in `joint_6` or `joint_7` is not necessary as long as the fingers remained closed using joint feedback control. However, please do incorporate collision checking for the gripper fingers.

Table 3: Initial and final joint targets for motion planning.

Joint name	Initial Value	Final Value
`joint_1`	-80 deg	0 deg
`joint_2`	0 deg	60 deg
`joint_3`	0 deg	-75 deg
`joint_4`	0 deg	-75 deg
`joint_5`	0 deg	0 deg
`joint_6`	-0.03 m	-0.03 m
`joint_7`	0.03 m	0.03 m

The V-REP scene file `locobot_motion.ttt` contains the LocoBot model as well as the obstacles. Use the V-REP interface to determine the cuboid specifications.

Deliverables:

1. A small, concise paragraph (2-4 sentences) describing your PRM motion planner.
2. A time history plot for `joint_1` through `joint_7` (same joints as in the last assignment). Include the target joint position on each plot as determined by the motion planner. **Your plots must include labels and units on each axis.** It is expected that the joints corresponding to the gripper fingers will remain closed.
3. A video of the LocoBot executing your motion plan. 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.
4. Your code for Questions 1 and 3.