Assignment 7

Assignment Objective

The objective of this assignment is to stack the tallest tower from the objects presented to the robot. We are looking for your individual creativity and your ability to reason about sequential manipulation planning.

Instructions

In this assignment, we will ask you to fill out a missing piece of code in a Python script. You will submit the completed code to Canvas. Your code is passed to an auto-grader that will verify the correctness of your work and assign you a score out of 100.

- Download the assignment 7 files from the Files menu of Canvas.
- For each "Part" of the HW assignment, fill in the missing code as described in the problem statement.
- Submit the completed `assignment_7.py` to Canvas.
- The assignment is due on December 15^{th}, 2020.
- You will need the `pybullet` library to simulate the robot. Please download/install this library.

Part 1 – Stack Tower (100 points)

Consider the scene depicted in Fig. 1. The robot is presented with a pile of objects. The objective of the robot is to build the tallest tower it can with these objects.

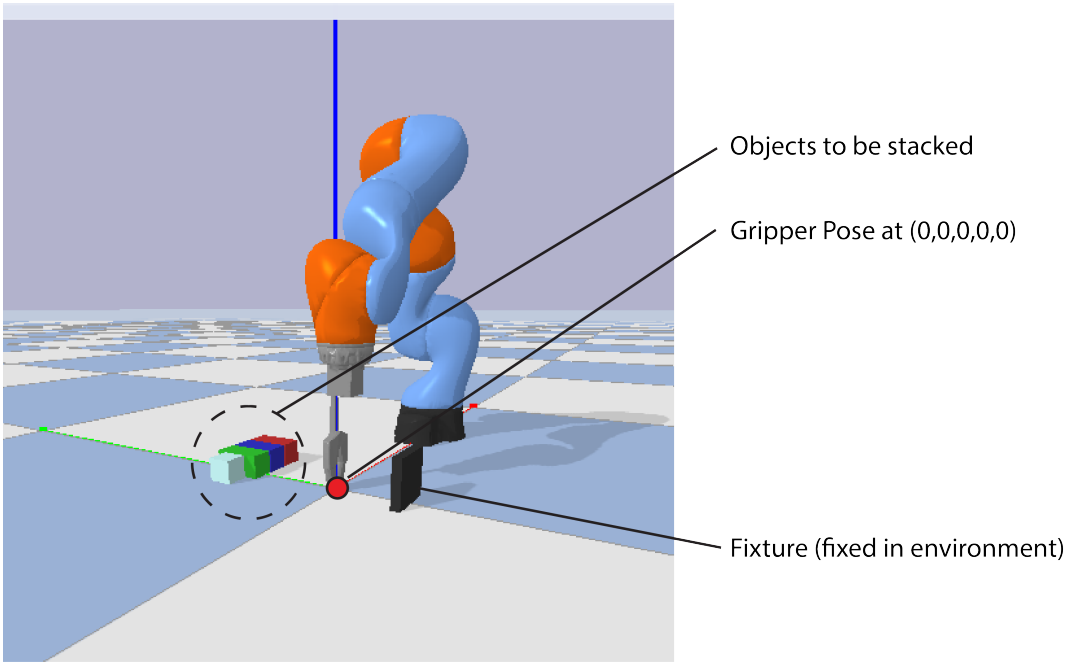

Figure 1. The robot and pile of objects to be stacked.

You may make the following assumptions regarding the pile of objects:

- The set of objects to be stacked is fixed and is composed of the red, green, and blue rectangular polygonal shapes and the cyan box shape. The black rectangular polygon is a fixture and can be used freely but cannot be moved.

- The initial configuration of the objects is fixed.
- Every object is both graspable and pushable.
- You will have access to the pose of all objects at every time step.

Complete the `stack` function in `assignment_7.py`. This function does not have any inputs and should output `True` once you're done. To help you, we have provided the following functionalities:

1. `get_obj_states` returns the configurations of the objects as a `np.array((4, 7))` where the rows represent each object, the first 3 columns represent the object linear positions (x, y, z), and the next 4 columns represent the quaternion orientation of the object (q_x, q_y, q_z, q_w) – all in the world frame. The order of the objects is (red, green, blue, box).
2. `get_robot_state` returns the Cartesian pose of the `np.array((4, ))` where the first 3 numbers are the cartesian pose of the gripper in the world frame and the last number denotes the gripper angle in radians.
3. `robot_command` commands the robot through a sequence of desired configurations defined by waypoints as `[np.array((5, )), ... , np.array((5, ))]` – a list of arrays. The first three numbers specify the gripper pose in the world frame, the fourth number specifies the orientation of the gripper with respect to the positive x axis (counter-clockwise rotations are positive), and the final number specifies how wide the gripper opens. We have written the internal controller that will move the robot through the desired sequence of waypoints – you do not need to worry about impedance control here. The gripper number range is between $(0, 0.2)$ corresponding to the angle of opening. At max opening, the gripper can grasp the rectangular blocks along their width but no along their length. Values outside this range are clipped to the boundaries.

We suggest that you try to decompose the task into a sequence of pushes and grasps. You can write your own "push" and "grasp" primitive functions that produce trajectories for these actions. The assignment is graded based on the average height of the objects once the `stack` function is finished executing.