HW1: Working with Robot Simulator

CS4910

Spring 2022

The goal of this assignment will be to create a simulator for top-down grasping of a randomly placed object. Please refer to the Pybullet Quickstart Guide for help interacting with the simulator.

This assignment is **due January 28 by 3:25pm**. To submit your assignment, send an email with your completed 'topdowngrasping.py' file to klee.d@northeastern.edu.

1 Implementing Inverse Kinematics [10 pts]

A top down grasp means that the gripper is oriented with its z-axis pointing down, with some rotation, θ, about the z-axis. While it simple to express the grasp as a function of position and orientation of the gripper, we must eventually convert to joint positions to send commands to motors. We can do this conversion using inverse kinematics. Implement the method `RobotArm.solve_ik` to calculate the arm joint positions needed to realize a given position and orientation of the gripper. The method should also return the residuals (i.e. the position and orientation error of the solution), so that we can ensure that the gripper placement is accurate. See the docstring for formulas on how to calculate the residuals. To test your implementation, you can use the function `check_workspace_reachability`, which checks what x,y positions the robot can accurately perform a top-down grasp.

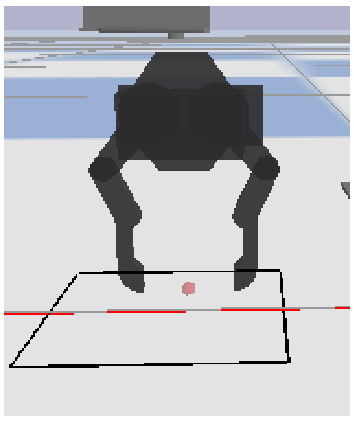
Figure 1: Top-down Grasp

2 Performing Top-down Grasp [15 pts]

Now that you have a reliable inverse kinematics solver, it is time to implement top-down grasping. Implement the method `RobotArm.move_gripper_to` which issues motor commands to reach a desired gripper pose for top down grasping. [Hint: it may be easier to think about gripper orientation with Euler angles, then convert to quaternion with `pybullet.getQuaternionFromEuler`]. Verify that your implementation is working properly using the function `test_move_gripper_to`.

Now, you are ready to implement the method `TopDownGraspingEnv.perform_grasp`, which moves the gripper to a grasp location, closes the gripper, and then returns to home position. It should return a boolean indicating whether the object was grasped. There should be a simple way to determine grasp success using either the gripper state or the position of the object.

3 Sampling Object Pose and Appearance [10 pts]

It is now time to place the object so that it can be grasped. A proper workspace is already defined for you, where top-down grasps are feasible. Implement the method `TopDownGraspingEnv.reset_object_position`, which places the object randomly in workspace. (Only rotate around z-axis, such that the long side is not sticking up).

Next, implement the method `TopDownGraspingEnv.reset_object_texture` to randomly change the texture of the object. [Hint: look up documentation for `pybullet.changeVisualShape`]

4 Expert Grasp Actions [10 pts]

If we were to collect a dataset, it is best to have an even amount of successful and unsuccessful grasps. If we randomly sample grasps in the workspace, then success rate is too low. However, we may use the state of the object to predict a grasp location that will have much higher success.

Implement the two methods `TopDownGraspingEnv.sample_random_grasp` (which should look very similar to your object position sampling code) and `TopDownGraspingEnv.sample_expert_grasp`. You should achieve over 90% success rate with the expert prediction.

5 Adding Camera [5 pts]

Lastly, we need a camera to render images of the scene. For top-down grasping, it is common to place the camera directly over the workspace looking down. Complete the implementation of `Camera.__init__` such that the workspace is fully contained within the image. You can test your camera placement with the function `test_camera_placement` (the black debug lines are not rendered by the camera).

6 Tying It All Together

Nice work! You now have everything you need to collect a dataset. You can run `mock_data_collection` to watch the process. In practice, the simulator can run headless to perform thousands of grasps per minute.