ECSE/CSDS 376/476
Lab 6: Vision-Based Manipulation
Assigned: 4/26/21
Due: 5/3/21

This assignment is a **group** assignment.

You should use your results from PCL processing to command the Baxter robot to pick up and put down a rectangular block. In **simulation**, you should be able to:

roslaunch baxter_gazebo baxter_world.launch (not necessary when using the real robot)

roslaunch exmpl_models add_table.launch (adds a table to the environment)

roslaunch exmpl_models add_toy_block.launch (puts a block on the table at known coordinates)

In simulation **or** physically:

roslaunch baxter_launch_files baxter_object_grabber_nodes.launch

In simulation, invoke manipulation with respect to the a priori known block coordinates with:

rosrun object_grabber example_object_grabber_action_client

This action client should cause the simulated robot to pick up the block, rotate it to align with the table, and place it back down.

For the physical robot you will want to incorporate an object-grapper action client, but invoke it with coordinates based on your point-cloud processing of Kinect images.

For the table used in this lab, the existing robot-arm planning code is not working well. You should modify it to produce better arm trajectories.

Key poses are:

- a waiting pose
- a pre-approach pose (newly defined here)
- an approach/depart pose
- a grasp pose

The waiting pose should have the hand above the table, gripper pointing down, and out of the view of the camera.

The grasp pose and approach/depart poses are defined with respect to the gripper frame. The gripper frame can be visualized in rviz (frame named “right_gripper”). This frame has its origin between the fingertips, with the z-axis pointing outwards, normal to the flange frame. Correspondingly, grasping objects from above requires requesting a gripper frame that has its z-axis pointing “down” ([0;0;-1]) with respect to the robot’s torso frame. The gripper-frame y-axis points from one fingertip to the other, and thus the gripper x-axis should be parallel (or anti-parallel) to the major axis of a long object in order to pick it up.

A grasp pose for grabbing the example rectangular block corresponds to placing the gripper fingers oriented pointing down, straddling the block at the block's centroid but not touching the table, and with the gripper-frame x-axis parallel to the block's major axis. The approach/depart poses are the same, but 0.1m higher (more positive z value, with respect to the robot's torso frame).

The (newly defined) "pre-approach" pose is merely the "waiting pose", but with the wrist joints (joints 5, 6 and 7) commanded to the same q5, q6 and q7 as the approach pose. This helps avoid hitting the table or block as the arm re-orients in joint space to move to the approach pose.

The arm's forward and inverse kinematic functions assume use of the torso frame and the right-arm flange frame. The flange frame (named "right_hand", but really is the right flange) is the same as the gripper frame, but offset along the gripper z axis by negative 0.158m along the gripper-frame z axis. (see line 158 of test_baxter_ik.cpp).

The test IK package "test_baxter_ik" (uploaded on Canvas) provides some code to help evaluate arm poses for manipulation. An objective is to find solutions for the grasp pose and the approach/depart pose that are close in joint-space, so the arm does not attempt to do a wrist flip while approaching the object nor after it is grasped. You should experiment with this code for candidate poses, such as those in arm_tf.txt (also on canvas).

To improve on the existing manipulation routines, you will want new functions in the library "baxter_cartesian_planner" in the package "cartesian_planner", which will be used by the node "baxter_rt_arm_cart_move_as.cpp". New functions may include:
compute_jspace_keyposes(Eigen::Affine3d gripper_pose)
plan_jspace_path_current_to_waiting_pose()
plan_jspace_path_current_to_pre_approach_pose()
plan_jspace_path_current_to_approach_pose()
plan_jspace_path_current_to_grasp_pose()

Correspondingly, you would want to define additional member variables in the class baxter_cartesian_planner to hold the results for computation of joint-space solutions for key poses.

The above functions could use code from test_baxter_ik (integrated into baxter_cartesian_planner) to compute better key poses, and to compute simple joint-space trajectories among these key poses.

The action-server case "EXECUTE_PLANNED_PATH" should be re-usable without change, but you would need to create one or more codes in cartesian_planner/action/cart_move.action to use the new functions (and the counterparts in object_grabber.cpp).

The intent of separating "object_grabber" from "baxter_rt_arm_cart_move_as" was to keep object-grabber generic and robot agnostic. There should be no reference to joint values in this code. However, this node can request that a robot-specific action server compute and save key joint-space poses and plan joint-space trajectories that can be invoked by the object_grabber.

The baxter_rt_arm_move_as action-server is necessarily targeted to Baxter kinematics. Your new functions can implement Baxter-specific key jointspace poses and jointspace trajectories.

Note: you may want to initially develop the manipulation code without use of PCL-based perception. During live labs, you could ask the TA to place the block on the table, position the robot's gripper straddling the block, and get the grasp pose by running: rosrun tf tf_echo torso right_gripper. This is how the data in "arm_tf.txt" was obtained. You should test with this data first, before allocating live lab time. You can do this in simulation, even without a table or block, to test the validity of arm motion plans.

Deliverables:

Submit a (group) report. Include data documenting the robot's performance (ideally, a video of successful object manipulation).

Describe what changes you had to make to the code to get this to work. Include a link to your code on github.

Describe observations—what worked, what didn't, what surprised you.