

Week 9 Lab – Manipulation Challenge

This lab sheet will guide you through creating your submission for the third challenge, which is about grasping. **Your goal will be to get an OpenManipulator robot to successfully pick objects and sort them into boxes based on their size.**

Exercise 1: Manipulating objects

Create a new package in your catkin workspace called lab9. Download the models and script files from SurreyLearn and extract them into the models and scripts subfolders of lab9.

Re-launch the simulator and controller you were using last week:

```
roslaunch open_manipulator_controller open_manipulator_controller.launch use_platform:=false
roslaunch open_manipulator_controllers joint_trajectory_controller.launch sim:=true
```

Remember that you need to click the “play” button in the bottom left of gazebo in order for RVIZ to appear. Assuming gazebo is still running, in the terminal run the following command to spawn a block in front of the robot:

```
roslaunch gazebo_ros spawn_model -urdf -file `rospack find lab9`/models/block.urdf -model block -x 0.2
```

Next spawn a pair of trays next to the arm using

```
roslaunch gazebo_ros spawn_model -sdf -file `rospack find lab9`/models/tray.sdf -model tray -y 0.3
roslaunch gazebo_ros spawn_model -sdf -file `rospack find lab9`/models/tray.sdf -model tray2 -y -0.3
```

Note that if you add any of these models incorrectly, you can delete them using

```
rosservice call gazebo/delete_model '{model_name: [block/tray/tray2] }'
```

Now, try to operate the arm by dragging it in MoveIt! Starting from the initial “rest” state, create a sequence of poses to pick up the block. At the start and end, echo the **/joint_states** topic and make a note of the position values. You don’t need to copy the first 2 numbers, which relate to the gripper, just the final 4 numbers.

You should record these poses in the “**pick_poses**” list at the top of lab9/scripts/run_challenge.py, with one pose stored in each list entry as shown in the example. If you wish to add some additional poses between the start and end, feel free to do so.

Next with the block picked up, do the same to create a sequence of poses that place the block into the left tray. Store these in the “**place_left_poses**” list in the same script. Note that if you find it difficult to move the arm side to side, you can switch to the “joints” tab in rviz and adjust the joints directly.

Week 9 Lab – Manipulation Challenge

Finally create a sequence of poses that place a block into the right tray (assuming it is already gripped). Store these in the “**place_right_poses**” list. If you like, you can try doing this by modifying the pose sequence you already for the left tray placement, rather than using teleoperation.

Exercise 2: Automated pick and place

In your lab9 package, create a launch file called “**run_challenge.launch**”. This launch file should in turn run the **open_manipulator_controller.launch** file and the **joint_trajectory_controller.launch** files using the same arguments as used previously.

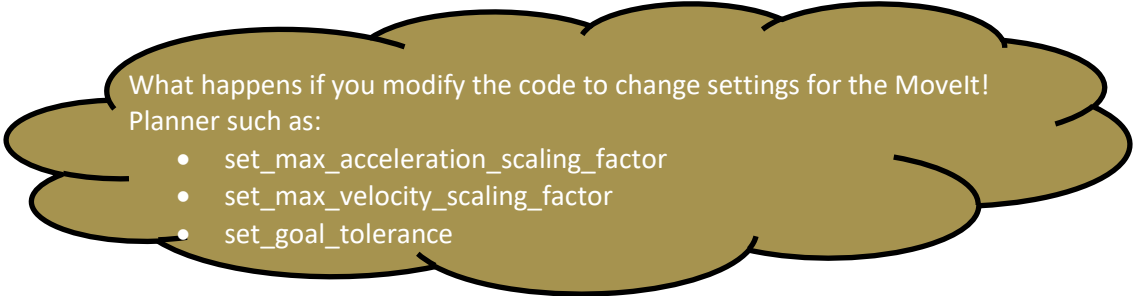
This launch file will now set up the environment needed to run your “**run_challenge.py**”, but first we need to add some missing code. If you launch it now the arm won’t do anything because the moveit planning code is missing.

In order for the code to work, you will need to add 3 missing lines of code in the **execute_move_sequence** function in **run_challenge.py**. The first line of code should call the “**set_joint_value_target**” function of the “**moveit_arm**” object, passing the “**pose**” object as an input argument.

The second and third lines of code should call the “**plan**” and “**execute**” functions of the move group respectively. Refer to the “Python Interface: MoveIt! Commander” slides if needed.

With this completed, you should be able to execute your launch file, then run the “**run_challenge.py**” node in another terminal. Every time you press “enter” on the keyboard, you should see a cube spawn in RVIZ (not gazebo). The arm should then complete a pick and place action following the route that you specified, and picking a random tray for placement. If the placement doesn’t work reliably you may wish to tweak the poses you chose in exercise 4.

NOTE: don’t worry if you see warnings about “no path found”. This is expected as the gripper closes on the block.



What happens if you modify the code to change settings for the MoveIt! Planner such as:


- `set_max_acceleration_scaling_factor`
- `set_max_velocity_scaling_factor`
- `set_goal_tolerance`

Exercise 3: Block sorting challenge

You may have noticed that the blocks have two possible sizes. In this final exercise we will try to change the code so that small blocks are placed into the left tray and large blocks are placed in the right tray.

To do this, we will modify the **inspect_object()** function in **run_challenge.py**. This function runs after a block has been picked, and before it is placed. Currently it returns 0 or 1 randomly. Try to modify it so that it returns 0 for small blocks and 1 for large blocks. **HINT: Try querying the grippers joint value after the pick. It will close more for smaller blocks and will stop early for larger blocks.**

Week 9 Lab – Manipulation Challenge



The `run_challenge.py` script comprises your entry for next week's challenge.

Consult with your group members and bring one such script per group on a USB stick to next week's session.

If you want to improve things further and increase reliability with the real arm, you might explore:

- Tweaking your additions to `run_challenge.py` (the pose lists, object inspection and `execute_move_sequence`)
- Play with the pathplanner being used by moveit, along with the other moveit settings.
- Have multiple placement points so that repeated placements don't collide.
- Adding the trays as collision objects to the moveit planning scene. This is very tricky as moveit cannot directly import SDF files. You can potentially build the trays out of primitive shapes based on the contents of the SDF file provided.