CS 6301 Introduction to Robot Manipulation and Navigation Homework 4

Professor Yu Xiang

October 26, 2022

In this homework, write down your solutions for problems 1 and finish the coding problem 2. Upload your solutions and code to eLearning. Our TA will check your solutions and run your scripts to verify them.

Problem 1

(5 points)

Robot Control. Exercise 11.6 in Lynch and Park, Modern Robotics.

Develop a controller for a one-dof mass-spring-damper system of the form $m\ddot{x} + b\dot{x} + kx = f$, where f is the control force and $m = 4$ kg, $b = 2$ Ns/m, and $k = 0.1$ N/m.

(a) Choose a P controller $f = K_p x_e$, where $x_e = x_d - x$ is the position error and $x_d = 0$. What value of K_p yields critical damping?

(Hint) Critically damped: damping ratio $\zeta = 1$.

(b) Choose a D controller $f = K_d \dot{x}_e$, where $x_d = 0$. What value of K_d yields critical damping?

(c) Choose a PD controller that yields critical damping. What is the relationship between K_p and K_d?

(d) For the PD controller above, if $x_d = 1$ and $\dot{x}_d = \ddot{x}_d = 0$, what is the steady-state error $x_e(t)$ as t goes to infinity? What is the steady-state control force?

(e) Now insert a PID controller for f. Assume $x_d \neq 0$ and $\dot{x}_d = \ddot{x}_d = 0$. Write the error dynamics in terms of $\ddot{x}_e$, $\dot{x}_e$, x_e, and $\int x_e(t)dt$ on the left-hand side and a constant forcing term on the right-hand side.

(Hint) You can write kx as $-k(x_d - x) + kx_d$.

Problem 2

(5 points)

ROS programming, robot control.

In this problem, you will learn using Moveit to control a Fetch robot in ROS. **You can directly use Ubuntu, or Docker or virtual machine to install ROS according to your own set up**. Refer to the ROS wiki page if needed `http://wiki.ros.org/`.

(4.1) Mounting a host folder into Docker if you use Docker. For example, the following command will mount a folder in Windows "C:\data" as a folder "/data" in Docker:

- docker run -it -v C:\data:/data ubuntu:ros

In this way, you can save all your code in the host machine and use them in the Docker environment.

(4.2) Creating a ROS workspace. Reuse your workspace from the previous homework. A ROS workspace is a place to store your own ROS packages. Following the link here to create a ROS workspace `http://wiki.ros.org/catkin/Tutorials/create_a_workspace`. You should create the ROS workspace in the mounted folder from the host machine.

(4.3) Install and launch Fetch Gazebo Simulator by following the steps in Homework 2. Use terminator to start multiple windows.

- roslaunch fetch_gazebo simple_grasp.launch.

You shall see the Gazebo environment as in Figure 1.

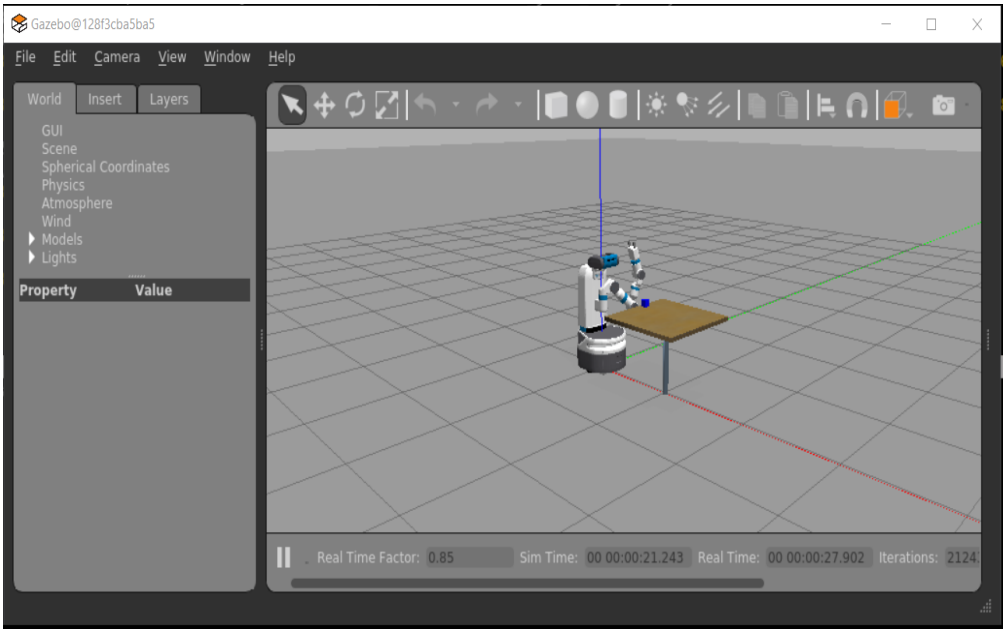

Figure 1: A Fetch Gazebo Interface.

(4.4) Install MoveIt and the fetch_ros package by following the steps in Homework 3 and then launch MoveIt planning by

- roslaunch fetch_moveit_config move_group.launch

You need to see the output as in Figure 2 that indicates the planning is ready.

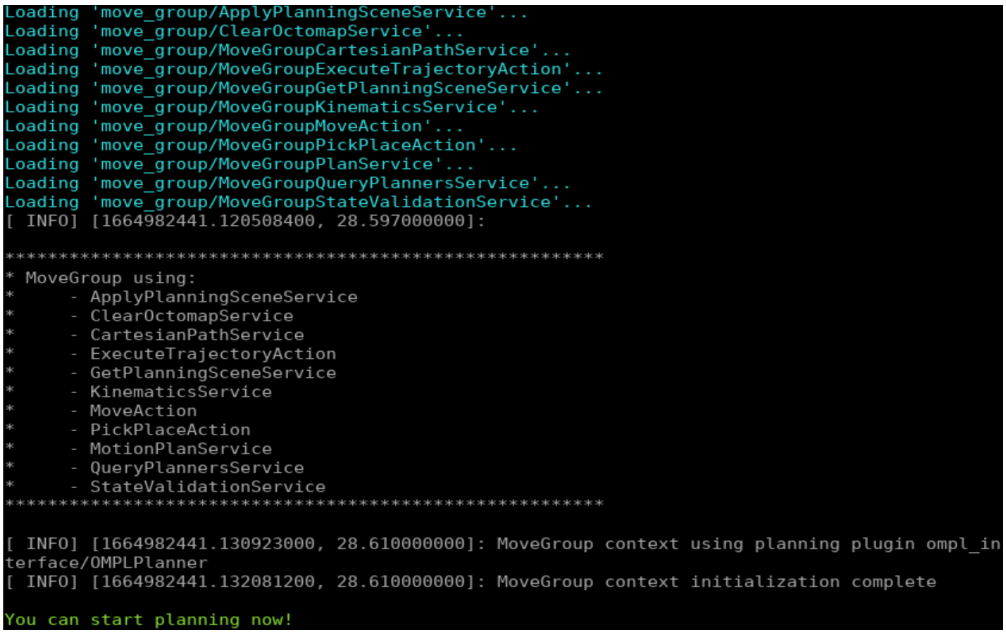

Figure 2: Output by launching *roslaunch fetch_moveit_config move_group.launch.*

(4.5) Up to now, you shall have the Gazebo and Moveit running. In this coding assignment, you need to use the pose of the demo cube computed from Homework 2 and trac_ik `https://bitbucket.org/traclabs/trac_ik/src/master/` for inverse kinematics from Homework 3 to grasp the demo cube.

You will use *move_group.go(joint_goal, wait=True)* to control the robot for grasping. This function will move the robot to the *joint_goal*. So we need to figure out *joint_goal* for grasping the demo cube. Figure 3 shows 3 tfs of the base link, the wrist roll link and the demo cube. The wrist roll link is the end-effector frame for inverse kinematics. Therefore, you need to figure out how to place the wrist roll link to grasp the cube. The distance between the wrist roll link and the finger tip is about 15cm. Once you decide the transformation for the wrist roll link in the robot base, you can use trac_ik to compute the *joint_goal*. Then the python script will move the robot to the *joint_goal*, close the gripper and move the arm back to the starting pose.

Download the grasp_block.py and the gripper.py from eLearning. Finish the implementation of the TODOs in the python script. Then you can run the python script. Figure 4 shows a Gazebo scene of running the script.

Submission guideline: Upload your implemented grasp_block.py file and the screen capture of Gazebo after running the script to eLearning.

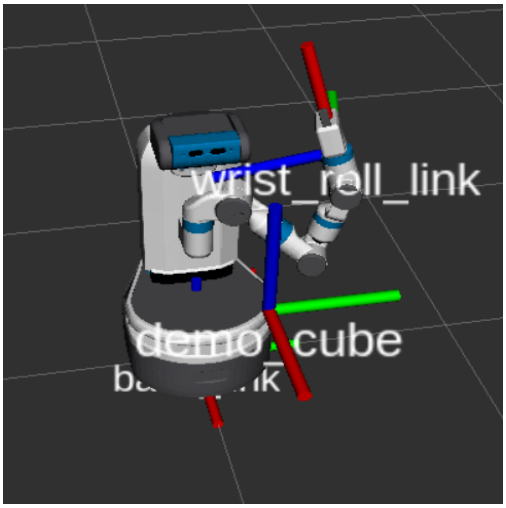

Figure 3: base_link, wrist_roll_link and demo_cube.

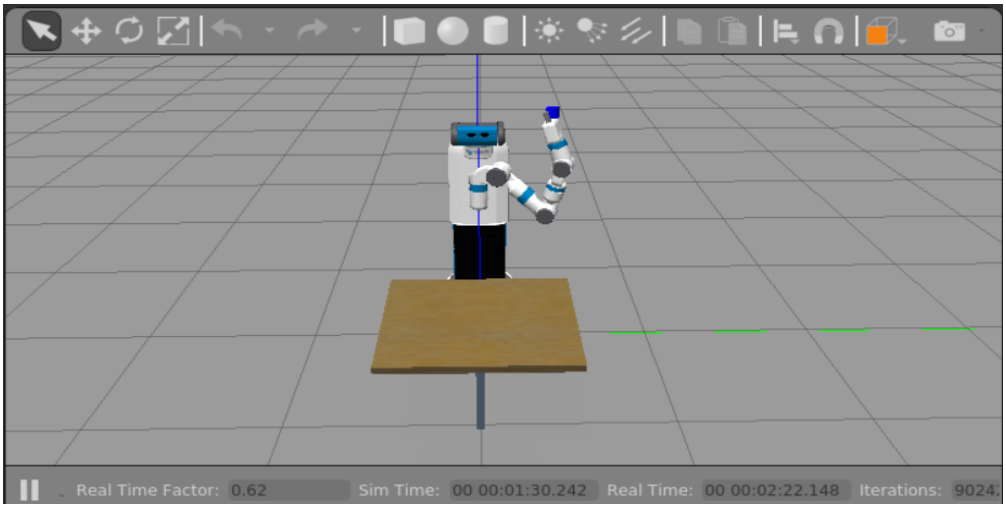

Figure 4: A Gazebo scene of picking up the cube.