CS 6301 Introduction to Robot Manipulation and Navigation Homework 5

Professor Yu Xiang

November 11, 2024

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, grasping.

In this problem, you will learn using Moveit to control a Fetch robot for grasping 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) **Setup ROS workspace.** 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) **Setup Gazebo.** Install and launch Fetch Gazebo Simulator by following the steps in Homework 2. Then follow the following steps to set up a Gazebo envoronment with a cracker box.

- Download and unzip cs6301_hw5.zip from eLearning. In the src folder of your ROS workspace, create a symbol link of cs6301_hw5, and then build your ROS workspace. Once built, source your workspace.
- Copy the models folder under cs6301_hw5 to the ~/.gazebo folder. In this way, we can use these 3D models in Gazebo.
- "cd cs6301_hw5/launch", then run "roslaunch table_ycb.launch".

You shall see the Gazebo environment as in Figure 1 with a table and a cracker box.

(4.4) **Setup Moveit.** 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.

(4.5) **Coding assignment.** Up to now, you shall have the Gazebo and Moveit running. In this coding assignment, you need to use moveit to plan a trajectory to enable Fetch to grasp a cracker box on a table.

Read the code in the grasp_cracker.py under cs6301_hw5/scripts. The code loads 50 planned grasps of the cracker box and then use Moveit to try which grasp can be used to grasp the object. Finally, it executes the planned trajectory for grasping.

Finish the implementation of the TODOs in the python script. Then you can run the python script. Figure 3 shows a Gazebo scene of running the script.

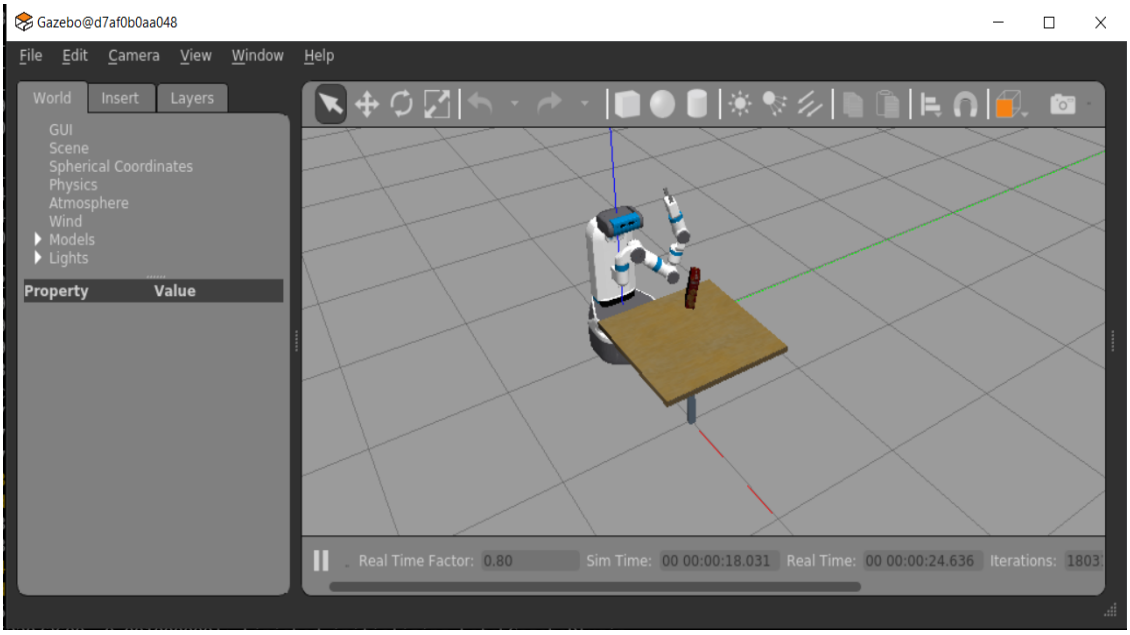

Figure 1: A Fetch Gazebo Interface.

To visualize the planned trajectory, you can start Rviz using my saved configuration file fetch_gazebo.rviz under folder cs6301_hw5:

- rosrun rviz rviz -d fetch_gazebo.rviz

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

```
Loading 'move_group/ApplyPlanningSceneService'...
Loading 'move_group/ClearOctomapService'...
Loading 'move_group/MoveGroupCartesianPathService'...
Loading 'move_group/MoveGroupExecuteTrajectoryAction'...
Loading 'move_group/MoveGroupGetPlanningSceneService'...
Loading 'move_group/MoveGroupKinematicsService'...
Loading 'move_group/MoveGroupMoveAction'...
Loading 'move_group/MoveGroupPickPlaceAction'...
Loading 'move_group/MoveGroupPlanService'...
Loading 'move_group/MoveGroupQueryPlannersService'...
Loading 'move_group/MoveGroupStateValidationService'...
[ INFO] [1664982441.120508400, 28.597000000]:

*********************************************************
* MoveGroup using:
*     - ApplyPlanningSceneService
*     - ClearOctomapService
*     - CartesianPathService
*     - ExecuteTrajectoryAction
*     - GetPlanningSceneService
*     - KinematicsService
*     - MoveAction
*     - PickPlaceAction
*     - MotionPlanService
*     - QueryPlannersService
*     - StateValidationService
*********************************************************

[ INFO] [1664982441.130923000, 28.610000000]: MoveGroup context using planning plugin ompl_in
terface/OMPLPlanner
[ INFO] [1664982441.132081200, 28.610000000]: MoveGroup context initialization complete

You can start planning now!
```

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

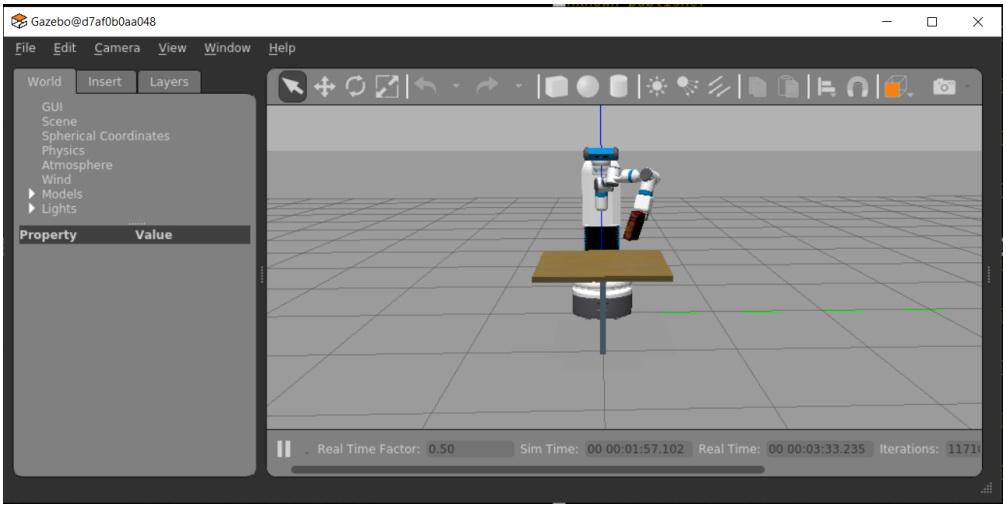

Figure 3: A Gazebo scene of picking up the cracker box.