Robot Autonomy - Homework 1

Professor: Oliver Kroemer

Due: March 12 2021, 11:59pm

1 Introduction

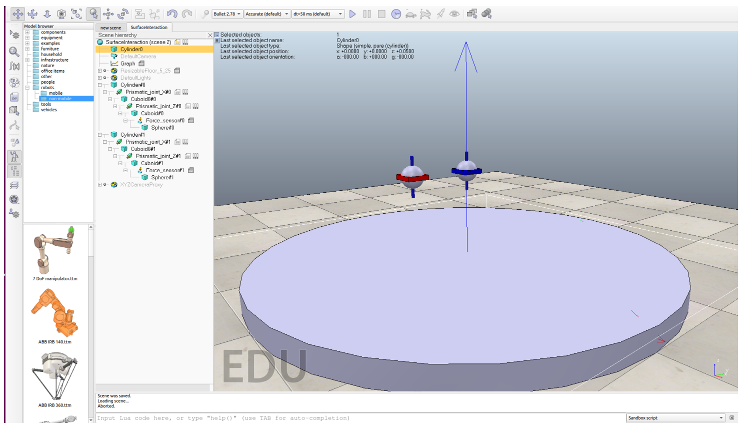
This homework will focus on the topics of kinematics and control from the lectures. You will explore force and impedance control on a simple scene in the VREP simulator for the first part, and then implement forward and inverse kinematics for the Locobot.

To use VREP (aka Coppelia Sim) you will need to download it from the website www.coppeliarobotics.com and install it. Use the educational version. The simulator can be run on Ubuntu, Windows, and MaxOs. If you require a virtual machine or help with the installation, please let me know.

You are encouraged to work together and discuss questions with your peers, but please write your own code for the implementations.

2 PID Control

In this section, you will implement PID controllers with a simple scene in VREP (aka Coppelia Sim) to help familiarize yourself with the simulation environment and understand the basics of PID control. First, launch VREP and then click File, Open Scene, then open the .ttt file in the Control folder called SurfaceInteraction.ttt. The scene should look like this:



Selected objects: 1
Last selected object name: Cylinder0
Last selected object type: Shape (simple, pure (cylinder))
Last selected object position: x: +0.0000 y: +0.0000 z: +0.0500
Last selected object orientation: a: -000.00 b: +000.00 g: -000.00

```
Scene was saved.
Loading scene...
Aborted.
```

If you look in the Scene Hierarchy tree on the left of the window, you will see the two elements below:

Prismatic_joint_Z#0 : Red sphere

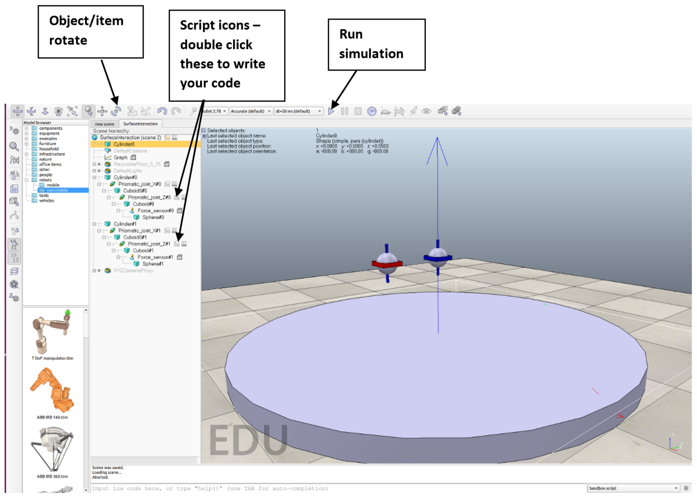
Prismatic_joint_Z#1 : Blue sphere

Impedance Control: Implement an Impedance controller on the Red sphere (*Prismatic_joint_Z#0*) to maintain a constant force of -10 N. Since this is impedance control, you will need to determine what target position to set to achieve the desired force.

Force Control: Implement a Force controller on the Blue sphere (*Prismatic_joint_Z#1*) to maintain a constant force of 10 N.

Implementation guidance:

Double click on the script icon next to each of the above objects in the tree - this script is where you'll be implementing your impedance and force controllers (in Lua). We've provided you with some starter code to work with. The comments in the code will guide you through what parts of the code you should modify and write yourself. You only need to edit the function called sysCall_jointCallback() to implement your controllers.



To help you with selecting PID gains for your force and impedance controllers, we've provided you with a general range of what you might want to set your gain values to:

Force control gains:

P = 0.05 - 0.15, I = 5 - 15, D = 0 (don't use a D term here)

Impedance control gains:

P = 150 - 250, I = 0 (don't use an I term here), D = 2 - 7

Run Controllers: Run the simulation (press the start simulation/play button in the top toolbar) with your controllers and plot the forces. Submit a screenshot of the force plots.

Disturbance: Now change the orientation of the plane - to do this click the object/item rotate icon in top toolbar, select the orientation tab, and change the beta angle to a nonzero angle (e.g. 45 degrees) and run the simulation.

Submission: For the control section, submit a pdf page with the following items:

- A screenshot of the force plots for the case where the surface is level
- A screenshot of the force plots for the case where the surface is tilted
- A screenshot of the spheres when they are above the lower half of the tilted surface.
- A 2-3 sentence explanation of the differences in the behavior of the force and impedance controllers.

3 Forward Kinematics for Locobot

In this section you will implement the forward kinematics for the Locobot arm and compute the Jacobian. Please use Python for your implementation. We have given you starter code in LocoBot.py and RobotUtil.py in the Kinematics folder. You may use the functions in RobotUtil.py to aid your implementation or you can implement your own functions. Please write your code to compute the FK and Jacobian in the ForwardKin() method in the Locobot class in Locobot.py. Your FK method should output the computed transforms for each of the joints based on the input angles and the Jacobian matrix.

Test your FK implementation: Run your FK method for the following joint configurations to compute the end effector pose. Submit your computed end effector poses for each set of joint positions below (note: these joint positions are also defined as a list called "joint_targets" in the MainTest.py file).

$$q_1 = \begin{bmatrix} 0^\circ \\ 0^\circ \\ 0^\circ \\ 0^\circ \\ 0^\circ \end{bmatrix}, \quad q_2 = \begin{bmatrix} -45^\circ \\ -15^\circ \\ 20^\circ \\ 15^\circ \\ -75^\circ \end{bmatrix}, \quad q_3 = \begin{bmatrix} 30^\circ \\ 60.^\circ \\ -65^\circ \\ 45^\circ \\ 0^\circ \end{bmatrix}$$

Self-check: You can check that your results with the following answer (Try to get within a few millimeters for translation and 2 or 3 decimal places for the rotation values):

$$t_2 = \begin{bmatrix} 0.32 \\ -0.25 \\ 0.33 \end{bmatrix} \; R_2 = \begin{bmatrix} 0.66 & 0.42 & -0.62 \\ -0.66 & -0.05 & -0.75 \\ -0.34 & 0.91 & 0.24 \end{bmatrix}$$

4 Inverse Kinematics for Locobot

In this section you will implement the inverse kinematics for the Locobot arm using the damped least squares method (set $W = 1$ and $C = 10^6$). Please use Python for your implementation. Write your code in the IK method in the IterInvKin() method in the Locobot class in Locobot.py.

Test your IK implementation: Run your IK method for starting joint configuration and goal end effector pose defined in the MainTest.py file (called qInit and HGoal in the code, respectively) and compute the joint configurations to achieve the desired end effector pose.

$$q_{init} = \begin{bmatrix} 0.0 \\ -0.5 \\ 1.0 \\ 0.5 \\ 0.0 \end{bmatrix}, t_g = \begin{bmatrix} 0.367 \\ 0 \\ 0.111 \end{bmatrix} R_g = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

Submission: For the kinematics section, submit a pdf page with the following items:

- The 3 end effector poses (in matrix form) corresponding to q_1, q_2, and q_3.
- The final joint angles for moving to the end effector goal pose of R_g,t_g.

5 Dynamic Movement Primitives (DMP)

Coming soon

For this portion of the homework, you will be implementing DMPs to generate smooth trajectories using a 1D trajectory that we have provided.

DMP/DMPMain.py contains a partial script for creating and executing a DMP. The first step will be to complete the learning script using linear regression. The second task is to complete the execution script. See the TODOs in the script to see where additional code needs to be added.

Submission: For the DMP section, submit a pdf page with the following items:

- The 15 weights learned from the demo data.
- A picture of the DMP executed from an initial state of 1.5, a goal state of -2.75 and a duration 2.1 seconds.
- A picture of the DMP like in the previous point, but using only 5 basis functions.