Robot Autonomy - Homework 1

Professor: Oliver Kroemer

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 MuJoCo (**Mu**lti-**Jo**int Dynamics with **Co**ntact) simulator for the first part, and then implement forward and inverse kinematics for the Franka arm. To set up MuJoCo for the homeworks instructions have been provided in the next section. An additional introductory pdf has been provided to get you started with MuJoCo once you have set up the simulator. The homeworks have been tested on Ubuntu. 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. MuJoCo Set Up

To set up MuJoCo,

1. Navigate to the desired working directory
2. Ideally create a separate environment to run your homework using conda. To do this, run "**`conda create –name <env_name>`**". Replace `env_name` with whatever you'd like.
3. Activate this environment by running "**`conda activate <env_name>`**".
4. Within this environment, run "**`conda install pip`**".
5. Find the location where pip was installed for your conda environment. It should look something like this, "**`/anaconda/envs/<env_name>/"`.**
6. Use this executable to install MuJoCo, "**`/anaconda/envs/<env_name>/bin/pip install mujoco"`.** This will install the python bindings as well as a copy of the library so that you don't need to install MuJoCo separately.

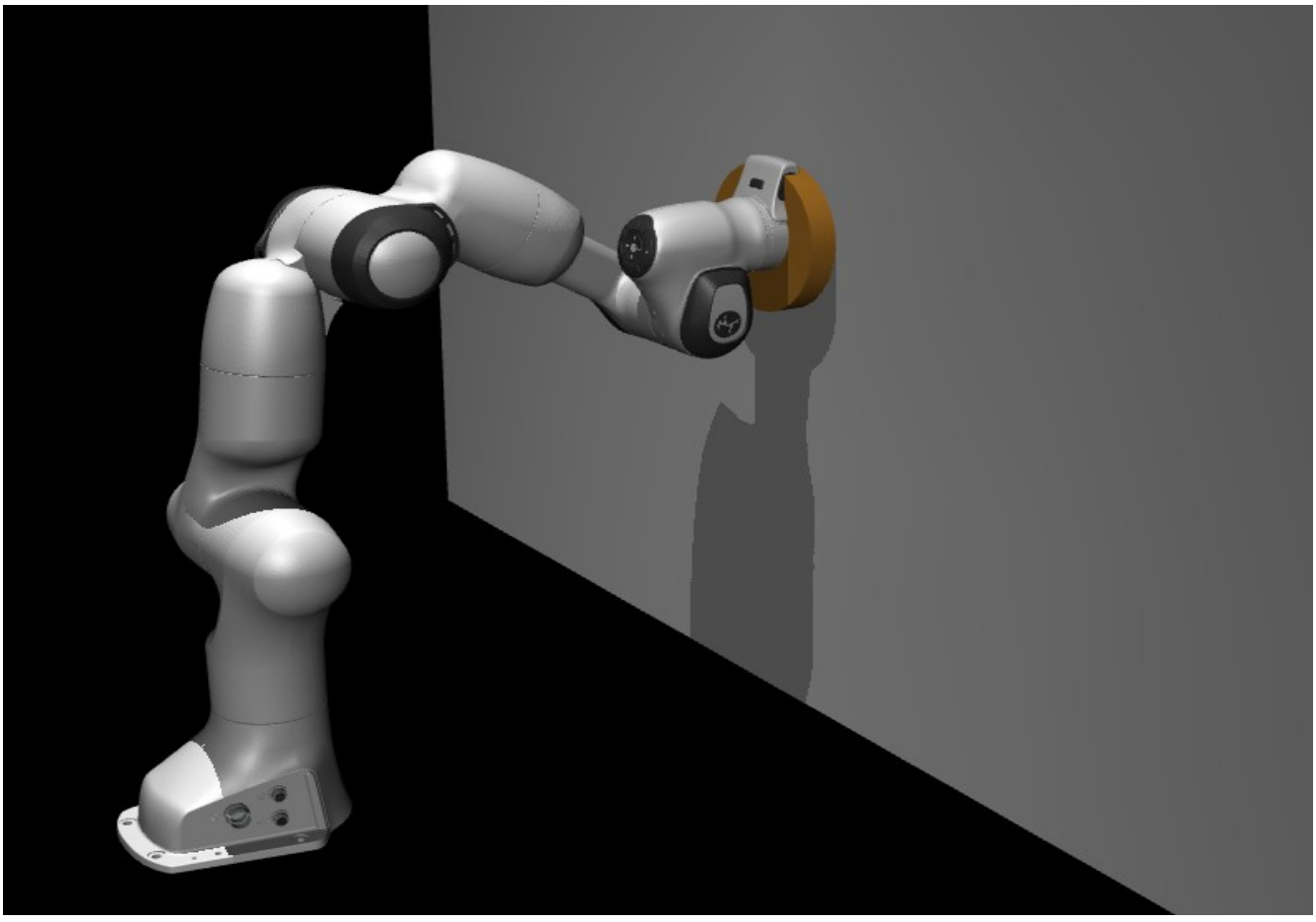
3. PID Control

In this section, you will implement PID controllers with a simple scene in MuJoCo to help familiarize yourself with the simulation environment and understand the basics of PID control.

You will find two scripts that are located in the control folder, **PandaArmControl_Part1.py** and **PandaArmControl_Part2.py.** The difference between the files lie in the world that is being loaded. In Part 1, we have a the Franka arm and a static whiteboard. In Part 2, we have the Franka arm along with a moving whiteboard.

3.1 Part 1

In Part 1, you are expected to implement an impedance controller and a force controller such that the arm generates 15 N of force against the whiteboard as shown in the figure below:



To simply things, the robot will be initialized with certain joint positions. This initial state is stored as part of the MJCF model file and will be called every time the simulator is started up. Go through the script to get a feel for how the pipeline works. Breifly,

1. The main function initializes the mjModel and mjData data structures, sets the key-frame and most relevant for the homework, sets the control callback function.

2. The control callbacks are defined above the main function. You are encouraged to set it to gravity compensation and position control to play around with the simulator and see how the robot interacts with the world. For more information on how to interact with the robot, take a look at the Intro to MuJoCo PDF provided with this handout.

3.2 Part 2

In this section, we will be using the same controllers that you implemented but with disturbance in the form of a sinusoidally moving whiteboard. Copy over the same controller implementations from Part 1 into Part 2 and simply run the file. Watch how the robot behaves in this case.

3.3 Plotting for Part 1 and Part 2

Both files will spit out csv files with time and force values that are recoded for 10 seconds during simulation. Make sure to run the simulator for at least 10 seconds in both Part 1 and Part 2 for each controller, Impedance and Force control.

Use any tool you would like to plot the force versus time. Just make sure to include a title, legend and axis labels for each of the plots.

3.4 Submission

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

- A screenshot of the force plots for the case where the whiteboard is static. Plot the outputs of the force controller and impedance controller superimposed on each other on the same figure. Make sure to include a title, axis labels and a legend identifying which curve indicates which corresponding controller.
- A screenshot of the force plots for the case where the whiteboard is oscillating. Plot the outputs of the force controller and impedance controller superimposed on each other on the same figure. Make sure to include a title, axis labels and a legend identifying which curve indicates which corresponding controller.
- A screenshot of the robot when the board is near its maximum amplitude, i.e. closest to the robot.
- A 2-3 sentence explanation of the differences in the behavior of the force and impedance controllers.

4. Kinematics for Franka

4.1 Forward Kinematics

In this section you will implement the forward kinematics for the Franka arm and compute the Jacobian. Please use Python for your implementation. We have given you starter code in Franka.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 FrankArm class in Franka.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).

Targets:

- q1 = [0°, 0°, 0°, 0°, 0°, 0°, 0°]
- q2 = [0°, 0°, -45°,-15°,20°,15°,-75°]
- q3 = [0°, 0°, 30°, -60°,-65°,45°,0°]

Expected output from FK for q1:

$$t = \begin{pmatrix} 0.088 \\ 0 \\ 0.926 \end{pmatrix}$$

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

4.2 Inverse Kinematics

In this section you will implement the inverse kinematics for the Franka arm using the damped least squares method. For the weights, use the following for W and C:

$$C = \begin{pmatrix} 1000000.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1000000.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1000000.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1000.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1000.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1000.0 \end{pmatrix}$$

$$W = \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 100.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 100.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 100.0 \end{pmatrix}$$

Please use Python for your implementation. Write your code in the IK method in the IterInvKin() method in the FrankArm class in Franka.py.

Test your IK implementation: Run your IK method for starting joint configuration and goal end effector pose defined in the Main.py file (called qInit and HGoal in the code, respectively) and compute the joint configurations to achieve the desired end effector pose. (*** The q_{init} given below is in radians**)

$$q_{init} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \\ -2.11 \\ 0.0 \\ 3.65 \\ -0.785 \end{pmatrix} \quad , \quad t_g = \begin{pmatrix} 0.6 \\ 0 \\ 0.5 \end{pmatrix} \quad , \quad R_g = \begin{pmatrix} 0.0 & 0.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \\ -1.0 & 0.0 & 0.0 \end{pmatrix}$$

4.3 Submission

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

- The 3 end effector poses (in matrix form) corresponding to q1 , q2 , and q3 .
- The final joint angles for moving to the end effector goal pose of R_g, t_g.
- Use the position controller callback and move the robot to the joint position that the inverse kinematics returns. Include an image of the robot at this position.