ME 449 Assignment 6
Robot Motion Control
Due 1:30 PM, Wednesday December 10, 2025

Important notes:

- As with all assignments, you are welcome to consult with other students on concepts, but you are not allowed to copy answers or share code.
- This document may undergo minor changes. If that happens, you will be notified in class.

In this assignment, you will plan and control the motion of a Universal Robots UR5, a popular 6-dof industrial robot arm. The robot has geared motors at each joint, but in this project, we ignore the effects of gearing, such as friction and the increased apparent inertia of the rotor. We will, however, assume damping at the joints, equal to 0.5 Nm/(rad/s), i.e., $\tau_i = -(0.5 \text{ Nms})\dot{\theta}_i$. (This value is not realistic, but it is probably more realistic than zero damping!)

The relevant kinematic and inertial parameters of the UR5 are:

$$M_{01} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.089159 \\ 0 & 0 & 0 & 1 \end{bmatrix}, M_{12} = \begin{bmatrix} 0 & 0 & 1 & 0.28 \\ 0 & 1 & 0 & 0.13585 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, M_{23} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -0.1197 \\ 0 & 0 & 1 & 0.395 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$M_{34} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0.14225 \\ 0 & 0 & 0 & 1 \end{bmatrix}, M_{45} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.093 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, M_{56} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.09465 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$M_{67} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.0823 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, G_1 = \texttt{diag}([0.010267495893, 0.010267495893, 0.00666, 3.7, 3.7, 3.7]),$$

$G_2 = \texttt{diag}([0.22689067591, 0.22689067591, 0.0151074, 8.393, 8.393, 8.393]),$
$G_3 = \texttt{diag}([0.049443313556, 0.049443313556, 0.004095, 2.275, 2.275, 2.275]),$
$G_4 = \texttt{diag}([0.111172755531, 0.111172755531, 0.21942, 1.219, 1.219, 1.219]),$
$G_5 = \texttt{diag}([0.111172755531, 0.111172755531, 0.21942, 1.219, 1.219, 1.219]),$
$G_6 = \texttt{diag}([0.0171364731454, 0.0171364731454, 0.033822, 0.1879, 0.1879, 0.1879]),$

$$\texttt{Slist} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & -0.089159 & -0.089159 & -0.089159 & -0.10915 & 0.005491 \\ 0 & 0 & 0 & 0 & 0.81725 & 0 \\ 0 & 0 & 0.425 & 0.81725 & 0 & 0.81725 \end{bmatrix}.$$

For your convenience, these parameters are given in Python, Mathematica, and MATLAB at http://hades.mech.northwestern.edu/index.php/Modern_Robotics#Supplemental_Information.

You will write code, using the Modern Robotics library, which takes a user input file and produces (1) a file that allows the user to visualize the resulting robot motion in CoppeliaSim and (2) files suitable for plotting. Specifically:

Input. The input to your software is a text file specifying a motion planning and control problem for the UR5, where the robot moves from one rest configuration to another. This text file can be typed directly in a text editor by a user who knows the required format, or, more ambitiously, it could be generated by an easy-to-use user interface. At a minimum, your input should include:

1. a specification of the start configuration $T_{s,\text{start}}$ of the planned trajectory of the robot's end-effector frame {b} (an $SE(3)$ matrix, or a simple representation of one, such as the (x, y, z) coordinates of the origin of the {b} frame in {s} and three angular exponential coordinates representing the orientation of {b} in {s})

2. a specification of the end configuration $T_{s,\text{end}}$ of the planned trajectory

3. a specification of the type of trajectory the robot should follow from start to end, allowing the user to choose from at least the following four options: screw trajectory with cubic time scaling, screw trajectory with quintic time scaling, Cartesian trajectory with cubic time scaling, or Cartesian trajectory with quintic time scaling

4. the duration T of the planned trajectory in seconds

5. a specification of the robot's actual initial configuration (e.g., a six-vector of joint angles) with the end-effector away from the trajectory start configuration, to emphasize the action of feedback control

You may also give the user the option to provide other inputs, such as the control law, the control gains, or joint torque limits, but these are not required. They can be hard-coded in your software.

This input file (or the GUI) should make it simple for a user to test new instances of motion planning and control. If the user specifies an impossible task (e.g., the start or end configurations are outside the workspace, or the planned trajectory moves outside the workspace), it is OK for your software to simply fail. Ideally your code would recognize the error and report to the user, but this is not necessary for this project.

Note: As described above, this assignment requires reading data from a text file, and optionally the use of a GUI to generate that text file. You are welcome to share code snippets for these aspects of the assignment, but you are not permitted to share code for any other aspect of the assignment (e.g., simulation, motion planning, and control).

Output. Your program will generate a reference trajectory for the end-effector, from rest at the start configuration to rest at the end configuration, and then simulate the UR5's motion under the action of your controller for time T. With a good controller and a modest initial error, the UR5 should converge to the desired trajectory within the time T. Your code should use MR code, but it is up to you how to use it.

Your code should output:

1. a csv file that records the simulated motion of the controlled robot and that can be animated in CoppeliaSim by Scene 2

2. a file that records the joint angles as a function of time

3. a file that records the commanded joint torques as a function of time

4. a file that records the angular error $||\omega_b||$ and linear error $||v_b||$ between the desired and actual end-effector configurations as a function of time

Post-processing. After completing a run of your code, you should be able to create a movie using Scene 2, plot the joint angles as a function of time, plot the joint torques as a function of time, plot the linear and angular errors as a function of time, and put these all in a single folder.

Debugging. You may find it helpful to test your work in pieces. For example, you could check whether the dynamic simulation is reasonable when the control torques are zero, and you could check whether a simple intuitive controller performs in an understandable way before moving on to a more complex controller.

What to Turn In

A single zip file with

- **A text file** with a brief description of your project, including a description of how to run the code (and where the MR code must be to work with your code). If your code has any special features, please mention them here. There should also be a brief description of the three example runs you are submitting, e.g., the controller type, trajectory type, etc., and any observations about the performance of the controllers in the three example runs.
- A folder with your code.
- Three folders showing simulation results. Each folder should include **the input file**, **the video from Scene 2**, a plot of **the joint angles as a function of time**, a plot of **the joint torques as a function of time**, and a plot of **the linear and angular errors as a function of time**. It is up to you to decide what to demonstrate with the three different examples; for example, they could demonstrate different trajectory types, different start/end configurations, different control gains, or anything else that is interesting or illuminating. Suggested: Demonstrate poor control behavior (e.g., bad control gains) for one example, or put limits on joint torques—if the controller requests a torque beyond a limit, the torque is capped at the limit. (This could prevent using unrealistic large gains that demand unrealistic large torques.) You should demonstrate more than one trajectory type; optionally, you could demonstrate more than one controller type, but it is also fine to use a single controller for all examples.