ME EN 5230/6230, CS 6330

Intro to Robot Control – Spring 2023

Problem Set #5: Coordinated Motion Control

The goal of this assignment is to design and simulate coordinated motion controllers for the Quanser 2-DOF serial robot, whose dynamics are given by:

$$\begin{bmatrix} N_1 k_{t_1} i_1 \\ N_2 k_{t_2} i_2 \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \begin{bmatrix} \ddot{\phi}_1 \\ \ddot{\phi}_2 \end{bmatrix} + \begin{bmatrix} 0 & -h \\ h & 0 \end{bmatrix} \begin{bmatrix} \dot{\phi}_1^2 \\ \dot{\phi}_2^2 \end{bmatrix} + \begin{bmatrix} G_1 \\ G_2 \end{bmatrix} + \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}$$

where ϕ_1 and ϕ_2 are the absolute joint angles (measured by the encoders), i_1 and i_2 are the motor currents, and:

$$\begin{aligned} H_{11} &= N_1^2 J_{m1} + I_1 + m_2 a_1^2 \\ H_{12} &= H_{21} = a_1 r_{12} m_2 \cos(\phi_2 - \phi_1) \\ H_{22} &= N_2^2 J_{m2} + I_2 \\ h &= a_1 r_{12} m_2 \sin(\phi_2 - \phi_1) \\ G_1 &= (r_{01} m_1 + a_1 m_2) g \cos(\phi_1) \\ G_2 &= r_{12} m_2 g \cos(\phi_2) \\ F_1 &= N_1^2 b_1 \dot{\phi}_1 + N_1 c_1 sgn(\dot{\phi}_1) \\ F_2 &= N_2^2 b_2 \dot{\phi}_2 + N_2 c_2 sgn(\dot{\phi}_2) \end{aligned}$$

Assume that the parameters are known to be:

Joint/Link: *i*	**1**	**2**
Link length: a_i	0.150 m	0.150 m
Link mass: m_i	0.092 kg	0.077 kg
Link center of mass: $r_{i,i+1}$	0.062 m	0.036 m
Link moment of inertia: I_i	0.64x10^{-3} kg·m^2	0.30x10^{-3} kg·m^2
Motor inertia: J_{mi}	0.65x10^{-6} kg·m^2	0.65x10^{-6} kg·m^2
Viscous damping constant: b_i	3.1x10^{-6} N·m/(rad/s)	3.1x10^{-6} N·m/(rad/s)
Coulomb friction constant: c_i	0.1 x10^{-3} N·m	0.1 x10^{-3} N·m
Motor Torque Constant: k_{t_i}	0.0077 N·m/A	0.0077 N·m/A
Gear Ratio: N_i	70	70

On the course website, you are provided with a Simulink template <PS5_template2023.slx>, which has the forward dynamics already programmed into the *Manipulator* subsystem block. The desired trajectory for the robot is programmed into the *Trajectory Generator* block using Cartesian coordinates:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.125 + 0.075\cos(2\pi f t) \\ 0.1 + 0.075\sin(2\pi f t) \end{bmatrix}$$

This creates a circular trajectory centered at [x,y]=[0.125m,0.1m] with a radius of 0.075m. Using the *Inverse Kinematics* block, this is converted to a desired trajectory in joint space. Your task is to design and simulate a series of joint space controllers to track this trajectory. Start by using PD control only, and find a set of gains that limits the joint tracking errors to ~0.05 radians (~3 degrees) at relatively low speed (f=0.2 circles/s). Hint: your PD gains from PS#4 should work fairly well. Then try adding various forms of nonlinear compensation as instructed below.

Problem Set #5: Coordinated Motion Control

1. Decentralized Control

For each of the following control schemes, simulate at low speed (f=0.2 circles/s) and high speed (f=1 circles/s) and compare the tracking performance with the other control schemes. For a fair comparison, make sure you use the same PD gains in all parts and speeds. When implementing gains and compensators, be mindful of your units. The angles are in radians and the motor command is in units of Amps. If you feedback a torque, you will need to multiply by the inverse motor gain to convert it to a current in Amps.

1.1 PD Control only

1.2 PD Control with gravity, friction, and centripetal/coriolis feedback compensation

1.3 PD Control with Computed Torque Feedforward Control

2. Centralized Control: PD Control with Inverse Dynamics Control.

Now design an Inverse Dynamics Controller (IDC) for the robot. Note that you will now need a different set of PD gains for a fair comparison with previous controllers, since the PD gains will now be multiplied by the inertia matrix and inverse motor gain in your control loop.

2.1 Use root locus techniques to find a comparable set of PD gains for your IDC controller, You can use the same overshoot and settling time criteria as in PS#4, but recall that with IDC control, your open-loop transfer function for the root locus is now effectively $1/s^2$.

2.2 Simulate your IDC controller at low speed (f=0.2 circles/s) and high speed (f=1 circles/s) and compare the tracking performance with the decentralized control schemes.

Notes: To implement your nonlinear compensators, use Embedded Matlab Function Blocks. Use the *Forward Dynamics* block inside the *Manipulator* subsystem in the template as an example of how to program an Embedded Matlab Function. You can even copy code from the *Forward Dynamics* block for use in your compensators. If you wish, you can use the *RobotApp* that is posted on Canvas to control the simulations. If the GUI is too sluggish, you can adjust the refresh rate of the *RobotApp* (this corresponds to adjusting the sample time in the Phi scope block), or you can always run the model directly in Simulink without the GUI.

Requirements for your analysis/writeup:
For each control scheme, provide an image of your Simulink model and the code from your Embedded MATLAB Functions. For each simulation, make a plot of the x-y trajectory along with the desired trajectory and a plot of the joint angle errors. Be sure to properly title your plots and label your axes. (To do this, you'll want to send the x-y coordinates and joint data to the workspace.) Include the code from any MATLAB scripts you use to analyze/plot the data. When comparing the tracking performance of different controllers, use both peak joint errors and the RMS (Root-Mean-Square) of the joint errors as metrics. For a fair comparison, be sure you use the same Model Settings (solver, step time, time duration). It is recommended that you use an application like MS Word to compile your writeup, in which case you should copy your Simulink models and MATLAB plots as Metafiles in order to make high-resolution figures.