ME 446 Laboratory #3
Inverse Dynamics Joint Control

Report is due at the beginning of your lab time the week of April 3rd. One report per group. Lab sessions will be held the weeks of March 20th and the week of March 27th.

Objectives

- Add Friction Compensation for all three joints and investigate how well it works to cancel friction effects in the arm.
- Design and implement an inverse dynamics control algorithm and compare its performance to the PD plus feedforward control designed in lab 2.
- Investigate how well the parameters of the CRS robot have been identified.

Part 1: Implement Friction Compensation.

1.1 Identified Friction Curves.

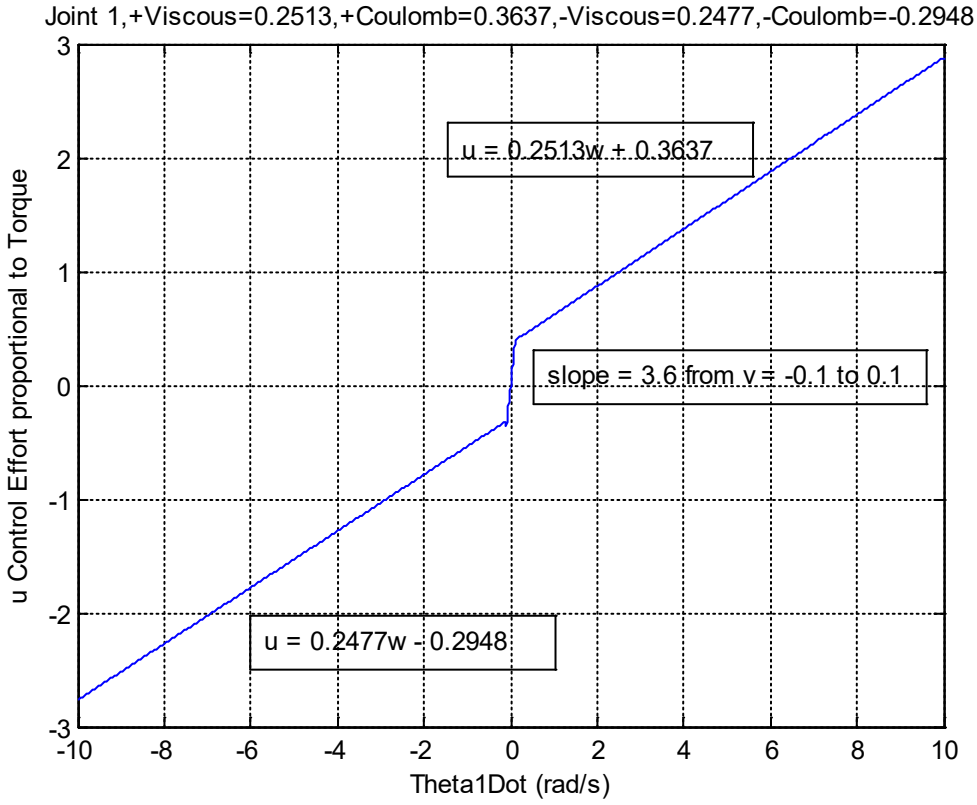
Below are experimental friction plots that were identified on one CRS robot arm. Constant velocity and control effort (proportional to torque) was recorded and these plots show the best fit lines that were fit to the collected data. Use these straight line equations to implement friction compensation for each joint of your robot arm. Make sure to not hard code the friction coefficients because these values will need to be tuned for your robot arm.

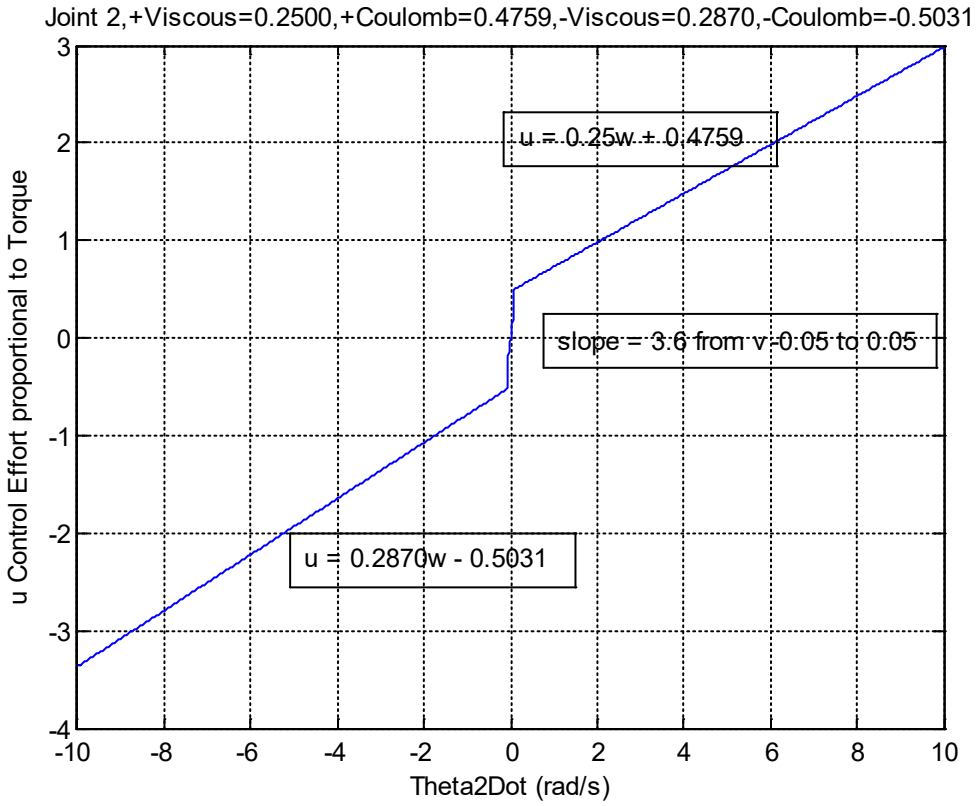
Once you have friction compensation implemented, adjust each joint's friction coefficients until you feel you have the correct coefficients for your robot arm.

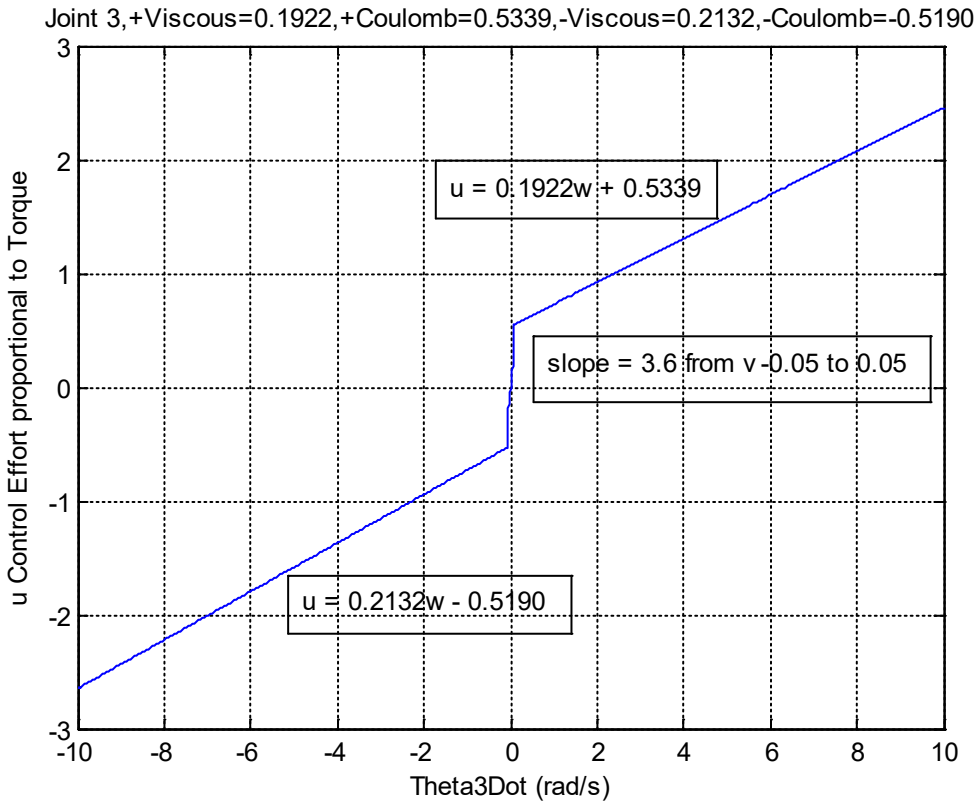
Pseudocode:

```
if (joint_velocity > minimum_velocity) {
        u_fric = Viscous_positive*joint_velocity + Coulomb_positive ;
} else if (joint_velocity < -minimum_velocity) {
        u_fric = Viscous_negative*joint_velocity + Coulomb_negative;
} else {
        u_fric = slope_between_minimums*joint_velocity;
}
```

Then after full control effort has been calculated, add u_fric to your calculated control effort. When testing only friction compensation simply set control effort only to u_fric.


Joint 1,+Viscous=0.2513,+Coulomb=0.3637,-Viscous=0.2477,-Coulomb=-0.2948
u = 0.2513w + 0.3637
slope = 3.6 from v = -0.1 to 0.1
u = 0.2477w - 0.2948
u Control Effort proportional to Torque
Theta1Dot (rad/s)
3
2
1
0
-1
-2
-3
-10
-8
-6
-4
-2
0
2
4
6
8
10


Joint 2,+Viscous=0.2500,+Coulomb=0.4759,-Viscous=0.2870,-Coulomb=-0.5031
u = 0.25w + 0.4759
slope = 3.6 from v -0.05 to 0.05
u = 0.2870w - 0.5031
u Control Effort proportional to Torque
Theta2Dot (rad/s)
3
2
1
0
-1
-2
-3
-4
-10
-8
-6
-4
-2
0
2
4
6
8
10



Part 2: Implement the Inverse Dynamics Control Law on Joints Two and Three.

2.1 Inverse Dynamics

This document does not go into detail on Inverse Dynamics control. Please see Section 8.2 of Robot Modeling and Control or section 11.3 of "Robot Dynamics and Control" second edition.

Modify your feed forward control (but make sure to keep a backup) from lab 2 to implement Inverse Dynamics control for joint 2 and joint 3. Use the given parameters from lab 2, **p** = [0.0300 0.0128 0.0076 0.0753 0.0298]. Recall from lab 2 that **p** is equal to:

$$p_1 = m_1 l_{c1}^2 + m_2 l_1^2 + I_1$$
$$p_2 = m_2 l_{c2}^2 + I_2$$
$$p_3 = m_2 l_1 l_{c2}$$
$$p_4 = m_1 l_{c1} + m_2 l_1$$
$$p_5 = m_2 l_{c2}$$

For joint one keep the same feed forward control from lab 2 tracking the desired trajectory.

So for joints two and three implement the inverse dynamics control

$$\tau = D(\theta)a_\theta + C(\theta, \dot{\theta})\dot{\theta} + g(\theta)$$

to track the cubic polynomial reference trajectory from part 4 of lab 2. This control equation is called the inner loop because it cancels the nonlinearities leaving the equation $a_\theta = \ddot{\theta}$. (This of course assumes we know the systems perfectly). Then an outer loop control is needed to control this linear set of equations. For the outer loop control use PD plus feed forward control and the same cubic trajectory from lab 2

$$a_{\theta_2} = \ddot{\theta}_2^d + K_{P2} * \left(\theta_2^d - \theta_2\right) + K_{D2} * \left(\dot{\theta}_2^d - \dot{\theta}_2\right)$$

$$a_{\theta_3} = \ddot{\theta}_3^d + K_{P3} * \left(\theta_3^d - \theta_3\right) + K_{D3} * \left(\dot{\theta}_3^d - \dot{\theta}_3\right).$$

Note that your Kp and Kd gains will be different from the PD plus feed forward values from lab 2.

So given these equations, the flow of your code each sample period should be:

1) Calculate the desired trajectory $\theta^d, \dot{\theta}^d, \ddot{\theta}^d$ (Same as cubic trajectory from Lab 2.)
2) Given measured thetas, calculate actual states, theta_dot, error, error_dot.
3) Calculate the outer loop control to come up values for a_{θ_2} and a_{θ_3}.
4) Calculate the inner loop control to find control effort to apply to joint 2 and 3.
5) Calculate Lab 2 feed forward control for joint 1 to find control effort to apply.
6) Calculate friction compensation control effort given the velocities of joint 1, 2 and 3.
7) Add the friction compensation to the control efforts (*tau1, *tau2, *tau3) calculated in 3, 4 and 5 above.

NOTE: When performing the inner loop calculations there are a number of sin() and cos() calls. Try to minimize the number of calls to sin() and cos() by creating a float variable say "float sintheta2". Then each time into your lab() function set sintheta2 = sin(theta2). Then use this variable every place in your calculations where sin(theta2) is needed. This way the sin(theta2) is only calculated once per sample period.

Tune your Kp and Kd gains to minimize error as the joints follow the two second cubic trajectory. (Make sure to ask your instructor how to create these error plots easily in the Simulink real-time plotting application.)

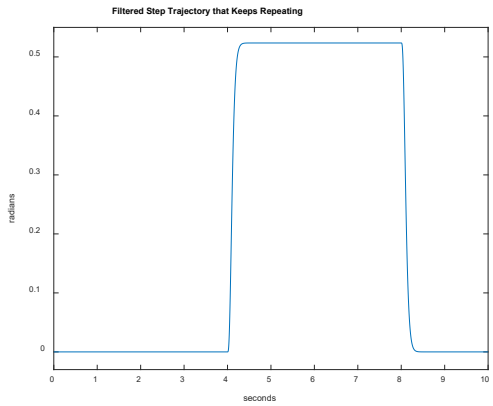
Now that you have tuned your Kp and Kd gains for this two second cubic trajectory, or what we will call the "slower" trajectory, we will introduce a faster trajectory in order to excite more of the dynamics of the robot link and hopefully see that the inverse Dynamics Control rejects some of those dynamic effects. We could design a faster cubic trajectory, but I would like to take this opportunity to introduce another way to design a smooth trajectory using a filter.

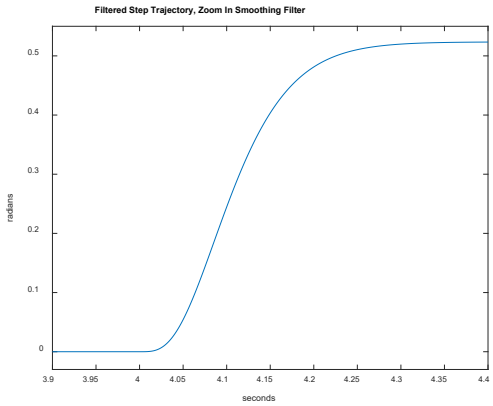
For this trajectory you will go back to creating a step reference that steps between 0.0 radians and π/6 radians. But before the step reference is used in the controller's error we will implement a discrete filter that filters the step reference to produce a smooth trajectory similar to the cubic trajectory. Your instructor will lecture on this way to produce a trajectory. Implementing discrete filters is a bit out of the scope of this class so I have given you an M-file to produce the C code of a discrete filter. This M-file is found here, http://coecsl.ece.illinois.edu/me446/filtersteptoC.m , and is also be in your repository's Matlab folder.

Use this discrete filter to create a smooth trajectory that:

1. Moves each joint from 0 radians to π/6 radians and back from π/6 radians to 0 radians and then repeats.
2. The step signal that will be filtered by the discrete filter will be a repeating step that steps to π/6 radians and holds there for 4 seconds and then steps back to 0 radians and holds there for 4 seconds.
3. Design a discrete filter to filter this step so that the smooth trajectory moves from its start position to its end position in .35 seconds. See figures below for an example.

With this faster trajectory, tune your Kp and Kd gains to minimize error. Produce trajectory response plots along with error plots for your report.





Part 3: Compare Inverse Dynamics Controller to Feedforward PD controller.

Part 3.1 Compare PD plus feedforward to Inverse Dynamics

Below you will run a comparisons between your Feedforward PD control from Lab 2 and the inverse dynamics control you just designed. Hopefully these comparisons will demonstrate that the inverse dynamics control does a better job controlling the linkage. I say "hopefully" here because the given identified parameters are not exact.

1. Implement your Feedforward PD control from lab 2 on joint 2 and joint 3 (joint 1 already finished in Part 2), but make sure to add the friction compensation you implemented with your inverse dynamics controller. For this comparison remove the integral control. Have the Feedforward PD control follow the same "faster" trajectory you just designed in Part 2. Also very important here, use a global integer to decide which controller is currently being run. So each 0.001 seconds, calculate both the Feedforward PD controller and the Inverse Dynamics Controller. Then if the global integer is equal to 0 set *tau2 and *tau3 to the Feedforward PD controller

calculations. If the global integer is equal to 1 set *tau2 and *tau3 to the Inverse Dynamics controller calculations. (*tau1 is always the Feedforward PD controller calculations.) With the Feedforward PD implemented, tune its Kp and Kd gains to achieve similar error responses as your inverse dynamics control.

2. Now that you have both your inverse dynamics controller with friction compensation and the Feedforward PD controller with friction compensation working somewhat similar, we are going to change the angles the joints step to, to see if the inverse dynamics controller performs better with this new trajectory.

   New Trajectories:

   1. For all these trajectories use the same filter that smooths the step to a .35 second response time.
   2. For Joint 1 and Joint 2, have those joints step (but filtered) back and forth between 0.25 radians and 0.85 radians.
   3. For Joint 3, have that joint step (but filtered) back and forth between 0.3 radians and -0.3 radians.
   4. Since the steps for Joint 1 and Joint 2 are different then the step for Joint 3, you will have to call the given filter function "implement_discrete_tf" twice. One dedicated to the Joint 1/Joint 2 trajectory and one dedicated to the Joint 3 trajectory. Also you will have to copy the given trajectory structure called "steptraj_t trajectory = {....." and give it a new name like "traj2". Then in your first call to "implement_discrete_tf" for Joint 1/Joint 2 trajectory pass "&trajectory" and in your second call to "implement_discrete_tf" for Joint 3 pass "&traj2". (Ask your instructor if this is confusing, and ask why this is necessary.)

   Then to compare, you will leave your PD controller gains for both controllers unchanged and check the error responses with the new step locations. In Code Composer's "Expressions" window, use the controller switching global integer to switch back and forward between the two controller calculations. Compare both the error peaks and the steady state error. Pay attention to whether gravity is compensated better with the Inverse Dynamics Controller.

3. By comparing these two controllers can you say anything about the parameters we are using? Does Inverse Dynamics perform better? For example has gravity effects been improved?

Report: (Minimal Requirements)

1. Include the final version of your C code.
2. Include any Matlab M-files you created (if any)
3. In your own words, explain the inverse dynamics control algorithm.

4. Answer the questions found in the lab.
5. Trajectory response plots and error plots.
6. You observations about the inverse dynamic controller and the system parameters used.