ME 446 Laboratory #2
PD and PID Joint Control

Report is due at the beginning of your lab time the week of March 13th. One report per group. Lab sessions will be held the weeks of February 20th, 27th, and March 6th.

Objectives

- Derive the equations of motion for a two link revolute linkage using Lagrangian dynamics.
- Create a Simulink simulation of this linkage.
- Challenge. Derive the full equations of motion for our three joint robot arm by adding the waist joint, θ_1, creating a three dimensional system.
- Learn how to implement velocity and integration calculations.
- Design and simulate PD joint control of two link planar robot.
- Implement and tune PD joint control for three of the CRS robot joints.
- Add integral control to the PD joint control.
- Implement and tune PD plus Feedforward Control.
- Using joints 1, 2 and 3 design a task space trajectory to make the CRS robot's end effect trace a figure eight or something else fun.

Part 1: Equations of Motion of Two Link Planar Arm

The System

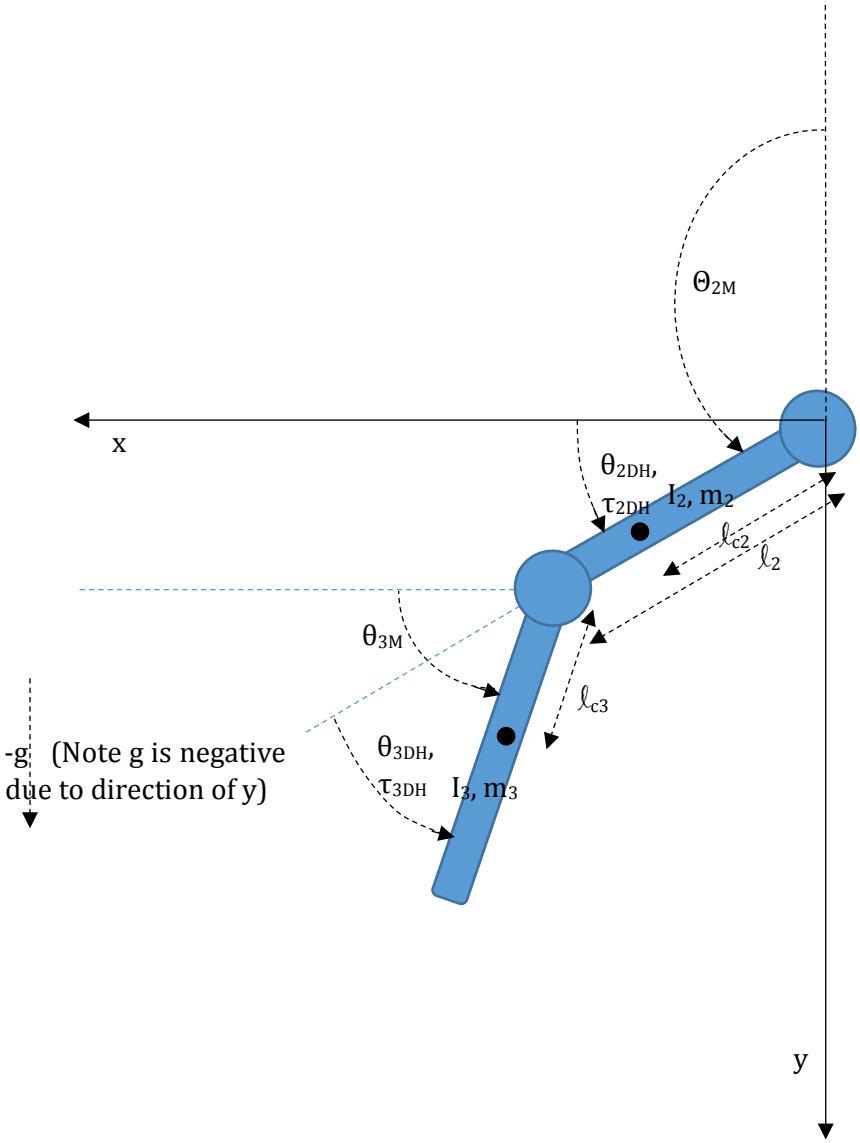


Figure 1.0 Physical Dimensions of Two Link Planar Arm

Procedure

1.1 Energy Equations

The parameters defining this planar arm can be reduced to five:

$$p_1 = m_2 l_{c2}^2 + m_3 l_2^2 + I_2$$
$$p_2 = m_3 l_{c3}^2 + I_3$$
$$p_3 = m_3 l_2 l_{c3}$$
$$p_4 = m_2 l_{c2} + m_3 l_2$$
$$p_5 = m_3 l_{c3}$$

Show that the kinetic energy for the planar arm can be written:

$$K = \tfrac{1}{2}\dot{\theta}_{2DH}^2 p_1 + (\tfrac{1}{2}\dot{\theta}_{2DH}^2 + \dot{\theta}_{2DH}\dot{\theta}_{3DH} + \tfrac{1}{2}\dot{\theta}_{3DH}^2) p_2 + (\cos\theta_{3DH}\dot{\theta}_{2DH}^2 + \cos\theta_{3DH}\dot{\theta}_{2DH}\dot{\theta}_{3DH}) p_3$$

Show with a picture that the potential energy for the planar arm can be written:

$$V = -p_4 g \sin\theta_{2DH} - p_5 g \sin(\theta_{2DH} + \theta_{3DH})$$

1.2 Dynamic Equations in terms of DH Thetas.

Using the Lagrangian method, show that the equations of motion for this linkage are:

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

with

$$\theta = \begin{bmatrix} \theta_{2DH} \\ \theta_{3DH} \end{bmatrix}, \dot{\theta} = \begin{bmatrix} \dot{\theta}_{2DH} \\ \dot{\theta}_{3DH} \end{bmatrix}, \ddot{\theta} = \begin{bmatrix} \ddot{\theta}_{2DH} \\ \ddot{\theta}_{3DH} \end{bmatrix}, \tau = \begin{bmatrix} \tau_{M\,2DH} \\ \tau_{M\,3DH} \end{bmatrix}$$

and

$$D(\theta) = \begin{bmatrix} p_1 + p_2 + 2p_3\cos\theta_{3DH} & p_2 + p_3\cos\theta_{3DH} \\ p_2 + p_3\cos\theta_{3DH} & p_2 \end{bmatrix},$$

$$C(\theta,\dot{\theta}) = \begin{bmatrix} -p_3\sin(\theta_{3DH})\dot{\theta}_{3DH} & -p_3\sin(\theta_{3DH})\dot{\theta}_{3DH} - p_3\sin(\theta_{3DH})\dot{\theta}_{2DH} \\ p_3\sin(\theta_{3DH})\dot{\theta}_{2DH} & 0 \end{bmatrix},$$

$$g(\theta) = \begin{bmatrix} -p_4 g\cos\theta_{2DH} - p_5 g\cos(\theta_{2DH} + \theta_{3DH}) \\ -p_5 g\cos(\theta_{2DH} + \theta_{3DH}) \end{bmatrix}$$

and finally in state space form can be written: (for serial linkages D is always invertible)

$$\begin{bmatrix} \ddot{\theta}_{2DH} \\ \ddot{\theta}_{3DH} \end{bmatrix} = D(\theta)^{-1}\tau - D(\theta)^{-1}C(\theta,\dot{\theta})\dot{\theta} - D(\theta)^{-1}g(\theta)$$

$$x_1 = \theta_{2DH}, x_2 = \dot{\theta}_{2DH}, x_3 = \theta_{3DH}, x_4 = \dot{\theta}_{3DH}$$
$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = \ddot{\theta}_{2DH}$$
$$\dot{x}_3 = x_4$$
$$\dot{x}_4 = \ddot{\theta}_{3DH}$$

1.3 Dynamic Equations in terms of Motor Thetas.

For our controller simulations and actual implementation it will be convenient to just focus on the motor angles instead of the DH angles. Also for our simulations we will only focus on θ_{2M} and θ_{3M} as simulating θ_{1M} complicates the dynamic equations. Perform the following derivation to show that the equations of motion in terms of motor thetas match the equations below.

As you should have found in Lab 1, the relationship between DH thetas and motor thetas is

$$\theta_{2DH} = \theta_{2M} - {}^{\pi}/_{2}$$

$$\theta_{3DH} = \theta_{3M} - \theta_{2M} + {}^{\pi}/_{2}$$

Also since τ_{3M} acts relative to the same base as τ_{2M} we have the relationship that

$$\tau_{2DH} = \tau_{2M} + \tau_{3M}$$

$$\tau_{3DH} = \tau_{3M}$$

Substituting these four equations, along with the derivative and second derivative of θ_{2DH} and θ_{3DH}, into the DH equations of motion, show that the new equations of motion become.

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

with

$$\theta = \begin{bmatrix} \theta_{2M} \\ \theta_{3M} \end{bmatrix}, \dot{\theta} = \begin{bmatrix} \dot{\theta}_{2M} \\ \dot{\theta}_{3M} \end{bmatrix}, \ddot{\theta} = \begin{bmatrix} \ddot{\theta}_{2M} \\ \ddot{\theta}_{3M} \end{bmatrix}, \tau = \begin{bmatrix} \tau_{M2} \\ \tau_{M3} \end{bmatrix}$$

$$D(\theta) = \begin{bmatrix} p_1 & -p_3 \sin(\theta_{3M} - \theta_{2M}) \\ -p_3 \sin(\theta_{3M} - \theta_{2M}) & p_2 \end{bmatrix},$$

$$C(\theta,\dot{\theta}) = \begin{bmatrix} 0 & -p_3 \cos(\theta_{3M} - \theta_{2M})\dot{\theta}_{3M} \\ p_3 \cos(\theta_{3M} - \theta_{2M})\dot{\theta}_{2M} & 0 \end{bmatrix},$$

$$g(\theta) = \begin{bmatrix} -p_4 g \sin\theta_{2M} \\ -p_5 g \cos\theta_{3M} \end{bmatrix}$$

and finally in state space form can be written: (for serial linkages D is always invertible)

$$\begin{bmatrix} \ddot{\theta}_{2M} \\ \ddot{\theta}_{3M} \end{bmatrix} = D(\theta)^{-1}\tau - D(\theta)^{-1}C(\theta,\dot{\theta})\dot{\theta} - D(\theta)^{-1}g(\theta)$$

$$x_1 = \theta_{2M}, x_2 = \dot{\theta}_{2M}, x_3 = \theta_{3M}, x_4 = \dot{\theta}_{3M}$$

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \ddot{\theta}_{2M}$$

$$\dot{x}_3 = x_4$$

$$\dot{x}_4 = \ddot{\theta}_{3M}$$

Part 2: Simulation

In Simulink, produce a nonlinear simulation of the equations of motion you just derived using motor theta. Use the following five parameters

p = [0.0300(PWM units*s^2)

0.0128(PWM units*s^2)

0.0076(PWM units*s^2)

0.0753(PWM units*s^2/m)

0.0298(PWM units*s^2/m)]

These parameters are a best guess using a 3D model of the robot linkage. The SolidWorks files for this 3D model can be found at N:\labs\me446\3DModel. Also the M-file used to find these parameters is found at http://coecsl.ece.illinois.edu/me446/ID_CRS.m.

Use an all integrator block diagram with two integrators for $\ddot{\theta}_{2M}, \dot{\theta}_{2M}, \theta_{2M}$ and two integrators for $\ddot{\theta}_{3M}, \dot{\theta}_{3M}, \theta_{3M}$. Then use function blocks to implement the nonlinear equations. You may find the last few pages of this ME 340 lab helpful in creating the Simulink simulation. http://coecsl.ece.illinois.edu/me446/ME340Lab7_handout.pdf Note though that the equations in this ME 340 lab are different due to a different definition of θ_{2M} and θ_{3M}. Have your TA verify that your simulation is correct before going to the next section.

Part 3: How to Calculate and Implement Velocity needed in the PID Controller.

3.1 How to find velocity.

The CRS robot has sensors (optical encoders) to measure the angle at each of its joints, but it does not have a velocity sensor. Taking advantage of the fact that our lab() function is called exactly every 1ms, current and previous angle measurements can be used to find and an approximation of the velocity of the system. Global variables can be used to save previous sampled thetas and previous velocity calculations. Velocity can then be calculated using the simple equation:

$\dot{\theta} = \frac{\theta_{current} - \theta_{previous}}{T}$. We will call this "raw" velocity. It is a good approximation but can be quite noisy. Normally some filtering is performed on this raw velocity calculation to get rid of some of the noise. But too much filtering causes phase delay which can hinder your control implementation. In my experience 2nd or 3rd order filters is the limit for velocity filtering. More complicated filters such as a Butterworth filter can be used but many times a simple averaging filter will suffice. There are also Laplace domain derivative approximations like 100s/(s+100) that can be discretized using the Tustin rule (same as the Trapezoidal integration rule) to achieve a discrete velocity approximation. This requires some digital control knowledge. (See Appendix if you wish) So unless you have taken a class in digital control I do not recommend you use this method for this lab. We will initially use averaging to filter the raw velocity calculation and only try the other mentioned ideas if we find the filter velocity to still be too noisy. To get an idea on how to implement this averaging filter, study the next two partial listings of C code. Explain to your TA, using terms like finite and infinite data (check out FIR and IIR links listed in the appendix below), what the difference is between these two methods. Which do you think filters the velocity better? Remember that the Lab() function is called every 1ms.

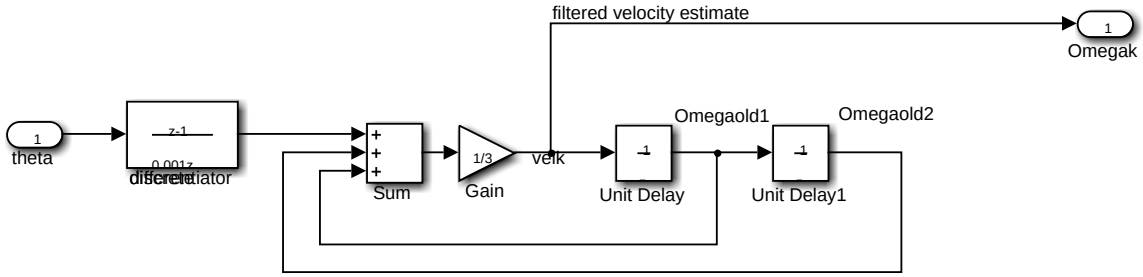
First Method of Filtering Velocity

```
float Theta1_old = 0;
float Omega1_raw = 0;
float Omega1_old1 = 0;
float Omega1_old2 = 0;
float Omega1 = 0;

// This function is called every 1 ms
void lab(float thetamotor1,float thetamotor2,float thetamotor3,float *tau1,float *tau2,float
*tau3) {

        Omega1_raw = (thetamotor1 - Theta1_old)/0.001;
        Omega1 = (Omega1_raw + Omega1_old1 + Omega1_old2)/3.0;

        Theta1_old = thetamotor1;

        //order matters here.   Why??
        Omega1_old2 = Omega1_old1;
        Omega1_old1 = Omega1_raw;
}
```

Second Method of Filtering Velocity

```
float Theta1_old = 0;
float Omega1_old1 = 0;
float Omega1_old2 = 0;
float Omega1 = 0;

// This function is called every 1 ms
void lab(float thetamotor1,float thetamotor2,float thetamotor3,float *tau1,float *tau2,float
*tau3) {

        Omega1 = (thetamotor1 - Theta1_old)/0.001;
        Omega1 = (Omega1 + Omega1_old1 + Omega1_old2)/3.0;

        Theta1_old = thetamotor1;

        //order matters here.   Why??
        Omega1_old2 = Omega1_old1;
        Omega1_old1 = Omega1;
}
```

This infinite average method can be implemented in Simulink with the blocks



Part 4: Simulation and PD control of Link two and Link three of the CRS Robot.

Using your simulation model from part 2, design and simulate a PD controllers that control motor 2 and motor 3. The control inputs τ_{2M} and τ_{3M} are computed as

$$\tau_{2M} = K_{P1} * \left(\theta_{2M}^{d} - \theta_{2M}\right) - K_{D1}\dot{\theta}_{2M}$$

$$\tau_{3M} = K_{P2} * \left(\theta_{3M}^{d} - \theta_{3M}\right) - K_{D2}\dot{\theta}_{3M}$$

Where θ_{2M}^{d} and θ_{3M}^{d} are step inputs from 0 to π/6 radians from t = 0s to t = 1s, followed by a step back to 0 from t = 1 to t = 2. Since all real motors have limited torque capability, saturate the torque to +/- 5. Start out with Kp1 and Kp2 equal to 1 and KD1 and KD2 equal to zero. Tune these four gains until you achieve a rise time less than 300 ms, percent overshoot less than 1% and steady state error minimal. Produce plots of your step responses along with the torque applied to the system. Also produce plots of tracking errors, $e_1 = \theta_{2M}^{d} - \theta_{2M}$,$e_2 = \theta_{3M}^{d} - \theta_{3M}$.

Part 5: PD and PID Control of the CRS Robot.

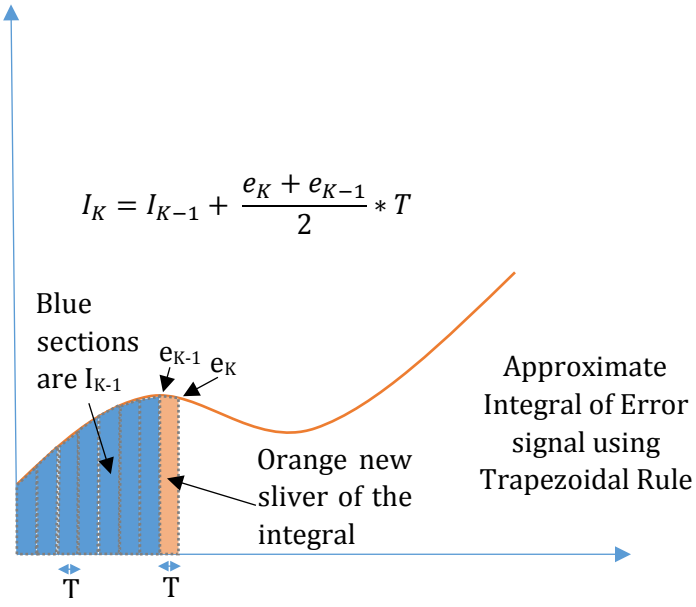
Part 5.1 Implementation of PD Control on Link One, Two and Three of the CRS Robot.

Now in Code Composer Studio starting from where you left off in Lab 1, implement PD control laws for the three joints of the CRS robot. Start with the gains found in simulation and then tune the gains to achieve Part 2's design specifications of 300ms rise time and 1% over shoot. Have the desired reference step back and forth between 0 radians and π/6 for all three joints. Make sure to saturate the torque

output between +/- 5. You should make your three Kp gains and your three Kd gains accessible to Matlab so they can be tuned while the robot is running. Also collect response data in an array that Matlab can upload and then plot.

5.2 How to implement an integral approximation.

There are a number of different integration rules that can be used to estimate an integral. For this exercise we will use the trapezoidal approximation to implement the integral portion of the PID controller. In the PID controller the error signal is integrated. Looking at the below figure you should be able to see how the trapezoidal rule at each new time step T sums up the new trapezoid sliver of area under the curve giving the equation $I_K = I_{K-1} + \frac{e_K + e_{K-1}}{2} * T$ to approximate the integral. The integral normally does not need filtering because its output is naturally smoother than then its input data.



Part 5.3 Add Integral Control.

Now add an integral term to your three PD control loops. Integration is a summing operator and must be monitored otherwise it could sum up to very large values. This problem is called integral windup.

You found above that the PD control, once tuned, did a very good job in controlling the robot's links. Looking close at your step response plots though, you probably did see some steady state error. We are going to add integral term to our controller to attempt to improve this steady state error.

Not only does integration have the problem of integral windup, it also has issues of creating large overshoots in position control systems. So the way we are going to implement integral control in this lab is to only turn on the integral term when error to desired position is small. How small is a tuning parameter.

So add to your PD controllers the integral term $K_I * I_K$, where I_K is the integral of the tracking error and calculated using the trapezoidal integration rule. Add this integral term to your PD control only if the tracking error is less than some thresh hold. **And very important**, if the tracking error is greater than the thresh hold make sure to zero the integral I_K and any previous I_{K-1}. To check for integral windup a very simple way is to just monitor the torque command to the joint's motor. If the absolute value of the torque command to the motor is greater than the maximum of 5, then do not integrate further and leave the integral the value it had the previous sample. Tune your PID controller's K_P, K_D, K_I and integral switch point to achieve the desired response. Create plots of your step responses and tracking error and determine if integral control improved the system response.

Part 6: PID Plus Feedforward Control.

Part 6.1 Implementation of Feedforward PD Controller.

1. Write a Matlab script file that finds two sets of cubic polynomial coefficients in order to generate a cubic polynomial trajectory from zero radians to 0.5 radians in one second and the from 0.5 radians to 0 radians in the next one second. We will use this same trajectory for all three joints of the CRS robot. This trajectory should therefore satisfy

$$\theta^d(0) = 0 \;\; ; \;\; \dot{\theta}^d(0) = 0$$

$$\theta^d(1) = 0.5 \;\; ; \;\; \dot{\theta}^d(1) = 0$$

$$\theta^d(2) = 0 \;\; ; \;\; \dot{\theta}^d(2) = 0$$

   Section 5.5.1 in "Robot Modeling and Control" and section 8.2.1 of "Robot Dynamics and Control" second edition will help you find these coefficients.

2. Using the coefficients you just found, write a Matlab M-file function that takes a parameter t seconds. Given t, this function should return $\theta^d(t), \dot{\theta}^d(t), \ddot{\theta}^d(t)$. If t is greater than 2 seconds, [0,0,0] should be returned. With this function, generate plots of the desired theta and its first and second derivatives. Note that you will be writing a similar function in C to generate the desired trajectory.
3. Implement on the CRS robot a PID plus feedforward control to have the joints follow the desired cubic polynomial trajectory. See Section 6.4 of "Robot Modeling and Control" or Section 10.4 for "Robot Dynamics and Control" second edition. Write a C function, that given t, returns that point along the

polynomial trajectory just like the M-file function you created above. Then using that trajectory implement the below control law. Ignoring friction, each joint's control law will have the form

$$\tau = J\ddot{\theta}^d + K_P(\theta^d - \theta) + K_I \int (\theta^d - \theta) + K_D(\dot{\theta}^d - \dot{\theta})$$

Use J1=0.0167, J2 = 0.03 and J3 = 0.0128. This is identical to the controller you implemented in 3.2 above except that the desired trajectory has been added. Also note that the sign in front of KD is now positive. Explain why it makes sense that with step input trajectories as used in sections 2 and 3 the KD term is appropriate to be $-K_D\dot{\theta}$ but now that we are generating polynomial trajectories the derivative term is $+K_D(\dot{\theta}^d - \dot{\theta})$. Also remember to implement the integral in the same fashion as before only applying integral control when the error is small and zeroing the integral when error is large. Because we are using polynomial trajectories the error should remain small as the joint moves along the trajectory. For this reason integral control will be active more of the time and may need some retuning along with the "small" error threshold for the integral. The K_P and K_D gains may also have to be retuned to meet specifications.

Part 7: Follow a repeating trajectory.

7.1 Follow a trajectory of your choice.

For this exercise come up with new trajectories for each joint that make the CRS robot arm repeatedly follow a fun trajectory. One idea would be to code your inverse kinematic equations that given an x,y,z point in space returns the three motor angles. With this code you could have the robot follow a straight line or a figure 8. As a safety check first code your trajectory equations in a Matlab M-file and with plots make sure the trajectory never takes the robot arm outside of its angle limits. Then code the same trajectory in C and see how well the robot can follow your trajectory.

Report: (Minimal Requirements)

1. Show all derivations of part 1.
2. Screen shot of your Simulink Simulation Model created in Part 2. Should be well commented. Your TA should have verified that this simulation is working correctly.
3. Include plots from your controller Simulink simulations
4. Include the final version of your C code that followings desired trajectories.
5. Include any Matlab M-files you created
6. Answer the questions found in the lab.

7. Explain how you generated your last “fun” trajectory.
8. Did you notice any performance differences between the different control methods?

Appendix

1. Tustin Rule https://en.wikipedia.org/wiki/Bilinear_transform
2. FIR filter https://en.wikipedia.org/wiki/Finite_impulse_response
3. IIR filter https://en.wikipedia.org/wiki/infinite_impulse_response