GE420 Laboratory Assignment 10
Discrete Full State Feedback Control of the Furuta Pendulum

Goals for this Lab Assignment:

1. Simulate and implement a full state feedback controller to stabilize the Furuta pendulum experiment.
2. Modify the implemented controller to allow for set-point changes to the first link's position.
3. Implement a swing-up algorithm to take the pendulum from the stable hanging position to the unstable inverted position.

SYS/BIOS Objects Used:

- Any of your own choosing.

MATLAB Functions Used:

Place, Simulink ploting

Prelab:

Read through entire lab assignment.

Controller Design and Simulation:

You will design a full state feedback controller to stabilize the Furuta pendulum in the upright position. You will also simulate your controller with the given non-linear model of the Furuta pendulum. You must complete the pre-lab and demonstrate it to your TA before proceeding to the implementation exercise. Note that the Furuta pendulum Simulink block is found on all the Controls Lab PCs. If you want to perform simulations on another PC, you will need to download the Simulink files found at the SE420 web site.

The System

The Furuta pendulum is a two link inverted pendulum experiment (See Figure 1). More accurately it is a two link underactuated linkage with no elbow actuation. Your goal then for the control of this experiment is to stabilize or balance the second link with the first link's torque input. You will use pole placement techniques to design the controller. The only design specification is to stabilize the Furuta pendulum in the upright position. Computer simulations in Simulink will first be performed to ensure proper controller design. After your simulation works, you will implement the control algorithm on the DSP system.

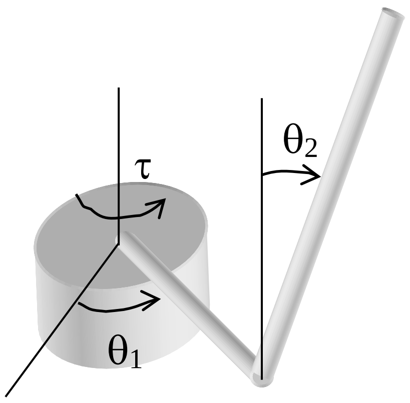
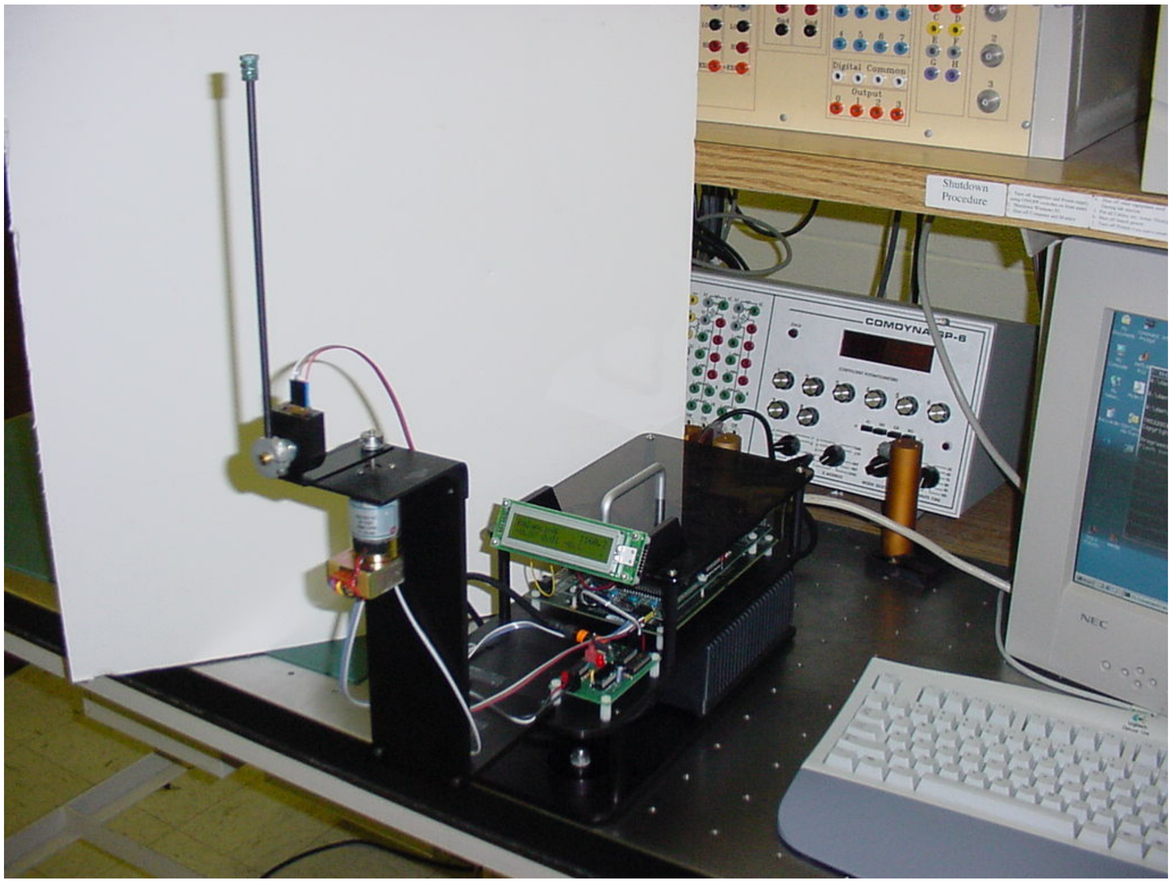


Figure 1. Definition of the coordinates for the Furuta Pendulum.



The dynamic equations of the Furuta pendulum are nonlinear having the form:

$$M\ddot{q} + C\dot{q} + G = \tau$$

where

$$q = [\theta_1, \theta_2]^T$$

$$M = \begin{bmatrix} J_1 + m_2(L_1^2 + l_2^2 \sin^2(\theta_2)) & m_2 L_1 l_2 \cos(\theta_2) \\ m_2 L_1 l_2 \cos(\theta_2) & J_2 + m_2 l_2^2 \end{bmatrix}$$

$$C = \begin{bmatrix} m_2 l_2^2 \sin(\theta_2)\cos(\theta_2)\dot{\theta}_2 & m_2 l_2^2 \sin(\theta_2)\cos(\theta_2)\dot{\theta}_1 - m_2 L_1 l_2 \sin(\theta_2)\dot{\theta}_2 \\ -m_2 l_2^2 \sin(\theta_2)\cos(\theta_2)\dot{\theta}_1 & 0 \end{bmatrix}$$

$$G = \begin{bmatrix} 0 \\ -m_2 l_2 g \sin(\theta_2) \end{bmatrix}$$

$$\tau = [u, 0]^T$$

with J_1 the moment of inertia of link 1 about its center of mass.

J_2 the moment of inertia of link 2 about its center of mass.

L_1 the length of link 1

l_2 the from link 2's joint to the center of mass of link 2.

m_1 mass of link 1.

m_2 mass of link2.

However, for our control design, we are only interested in the linearized set of equations about the balancing equilibrium point [θ_1, θ_1', θ_2, θ_2'] = [setpt, 0, 0, 0] where setpt is the desired angle for θ_1. (For much of this lab we will set **setpt =** 0). Figure 1 defines the joint angles, joint 1's torque and their directions. Linearizing the nonlinear equations about the equilibrium point and substituting identified values for J_1, J_2, L_1, l_2, m_1 and m_2 yields the **continuous** state equations:

$$\begin{aligned} \dot{\delta x} &= A\delta x + B\delta u \\ y &= C\delta x \end{aligned}, \qquad (10.1)$$

where

$$\delta x = \begin{bmatrix} x_1 - setpt \\ x_2 - 0 \\ x_3 - 0 \\ x_4 - 0 \end{bmatrix} = \begin{bmatrix} \theta_1 - setpt \\ \dot{\theta}_1 - 0 \\ \theta_2 - 0 \\ \dot{\theta}_2 - 0 \end{bmatrix}, A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -52.0624 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 76.1771 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 34.4136 \\ 0 \\ -13.3529 \end{bmatrix},$$

and C is the identity matrix. Note though, that on the actual system we can only measure x_1 and x_3. So we will have to come up with an estimate of the velocity states x_2 and x_4.

Controller Design

Use either pole placement or the LQR algorithm to design a full state feedback controller to stabilize the above linearized system discretized with the 'c2d' function with a zero-order hold. Remember that this is a discrete control class so you must perform all controller designs in the discrete domain. That doesn't mean you can't gain insight into the problem from the continuous domain. Use the **place** function in MATLAB to calculate a set of gains that move the open loop unstable poles of the Furuta pendulum to your chosen stable set of discrete poles. As a second option the Linear Quadratic Regulator (LQR) design (type **help dlqr** in MATLAB) is a method for designing full state feedback controllers using a minimization algorithm. It designs an optimal controller minimizing the equation $J = \int \left(x^T Q x + u^T R u\right) dt.$ Q and R are weight matrices that indicate to the LQR algorithm the importance you have on the different states and inputs of your system. For example, using an identity matrix for Q weights all the states equally. Making the (3,3) element of Q be 5, weights the third state five times as much as the remaining states-thus the response of the third state is 5 times more important than the others.

Approximate the velocities of θ_1 and θ_2, by using a derivative approximation **100s/(s+100)** and emulating with the Tustin rule. Perform the gain calculations in the discrete domain. Assume a ZOH for the plant at a sample rate of 5ms. Calculate a gain matrix K that will stabilize the plant in the inverted position. Make sure your controller's gains do not go outside the gain limits $|K_1|$ and $|K_3| < 100$, $|K_2|$ and $|K_4| < 20$.

Simulation

Two Simulink blocks are given to ease the simulation of the Furuta pendulum in MATLAB. These blocks, **Furuta** and **furutaanim**, can be found in the library **pend_blks.mdl**. Just type pend_blks at the Matlab command prompt. After opening that file, the blocks should be dropped into your own simulation, just as with all other Simulink libraries.

The **Furuta** Simulink block models the nonlinear equations of the Furuta Pendulum. In the block's parameters you must specify the linkage's five inertial parameters and initial position. Use the default inertial parameters, but make sure to enter your equilibrium point (or values close to your equilibrium point) as the initial position (the initial position for all angles should be zero). The **Furuta** block has two inputs. The first input is the control torque applied to link 1. The second input is a fictitious torque input applied to link 2. We will use this torque input to simulate a finger tapping the balanced link. This input can be left open or a time varying signal can be applied to it to explore the robustness of the controller to pulse inputs. The four outputs of the Furuta block are the four states of the Furuta [θ_1, θ_1', θ_2, θ_2']. You will be approximating your derivative states from θ_1 and θ_2, so you will not connect to the θ_1' and θ_2' outputs. The **furutaanim** block is an animation of the Furuta Pendulum. Add it to your simulation to see an animation of your control run. The inputs to **furutaanim** are the position states θ_1 and θ_2. Also, make sure to add a +/-10 (PWM Units) **saturation block** before link 1's torque input. The saturation block is required to simulate the limited torque of our small motor.

When your controller is working, experiment with the robustness of your controller by trying different starting conditions and simulating a finger tap. Use the Simulink **Pulse Generator** block to simulate the finger tap. Set the **Pulse Generator** block to use a period of 4 seconds, a pulse width of 4%, and a phase delay of 4 seconds. Start the amplitude of the pulse at 0.1 and increase it until it causes problems for your controller. Your controller should at a minimum survive a pulse of amplitude 2.5. Tune your controller gains to ensure that this specification is met without violating the control gain limits mentioned earlier.

Laboratory Exercise

Implement the Full State Feedback Controller on the Furuta Plant

Preparation

Using the SYS/BIOS objects of your choosing, implement the full state feedback controller designed in the prelab. Below we have listed a number of steps that will be helpful in the implementation of your controller. Read through all the steps because they are not listed in any certain order of importance and all are relevant to your controller implementation.

a. Because the Furuta is a non-linear system and we are controlling it with a linear controller there is only a small region around our operating point that the controller is valid (or works). Outside of that region your controller is violently unstable. For that reason you will need to add some safety checks to switch off control effort when the linkage gets too far from the operating point.
    a. The first limit is *if* $|u| > 30$*, then* $u = 0$, limiting the control effort to 30. (Note that the maximum control effort that can be applied to the motor amp is 10. The limit of 30 is needed in order to allow you to be able to tap the link without the control shutting off. When the control effort is between 10 and 30 the output to the motor will be saturated.)
    b. The second is *if* $|\theta_2| > 0.6$ *radians, then* $u = 0$.

Show your C code with these three checks to the TA before your initial run.

b. You should use a 5 ms sample period.

c. Print your states x_1 and x_3 and control effort u to the LCD.

d. Remember that you are approximating your velocities from θ_1 and θ_2 using the transfer function **100s/(s+100)** and emulating with the Tustin rule.

e. You will not be implementing the "swing-up" control for the Furuta until the end of the lab, so initially you will have to manually move the pendulum from the hanging "down" position to the inverted balancing position. You will need to make sure that each time before you run your DSP controller both link 1 and link 2 are resting still at the starting position $\theta_1=0$, $\theta_2=\pi$. (The "down" position). We have to do this because the optical encoders that we are using for feedback are incremental type optical encoders. Your program, by calling **init_EQEPs(),** initializes the encoders to a known state, $\theta_1=0$, $\theta_2=\pi$, each time your controller is restarted.

f. Your controller $u=-K^*\delta x$, only allows for one balancing position for link2, $x_3=0$. We know that any multiple of 2π is a valid balancing point. Below I list some C code that implements a modulo 2π divide of θ_2 that will allow for multiple turns of link 2. Add this code to your program and use **x3kmod** (or whatever you call it) as your x_3 state in the calculation of u.

```
if (x3k > PI) {
      x3kmod = x3k - 2.0*PI*floor((x3k+PI)/(2*PI));
} else if (x3k < -PI) {
      x3kmod = x3k - 2.0*PI*ceil((x3k-PI)/(2*PI));
} else {
      x3kmod = x3k;
}
```

g. Run your controller on the Furuta pendulum and confirm that it works. For a first check, make sure the motor amplifier is **OFF**, run your code on the DSP and watch the control effort. The control effort **should be zero** for nearly all arm positions other than near vertical. If this is not the case, you have a mistake and your code and must fix it before continuing to the remainder of the lab.

h. After step h is working properly and your system is balancing, add code to implement the friction compensation that you identified in Lab 7. Does your controller perform better with friction compensation added? Demo this controller to your TA.

Set-point change.

Make a float variable that works as the set point value that is part of the x1 delta state. Then in CCS expressions window change the set point to different values. Link one should step to the new set point. How well does your controller track to the set point? Show a step response to your TA by plotting the actual data response in Simulink as in previous labs.

Implement a swing-up algorithm for the Furuta pendulum that swings the linkage from the hanging down position to the inverted balancing position.

Now derive a swing-up algorithm to move the unactuated pendulum from its hanging down position to the inverted position. When close to the inverted position the C code should switch to the balancing controller to catch and balance the pendulum. There are many methods for swinging the linkage to the balancing position. For this exercise you will use part of the method proposed by Iwashiro, Furuta and Aström[1]. In this paper an energy control algorithm is derived to swing a Furuta style pendulum to a desired energy value. You will use this derivation to develop a control algorithm that always increases the energy of the pendulum causing it to gradually swing to the inverted position.

Consider Figure 2, a point mass pendulum with length "l" and mass "m" accelerated in the horizontal direction. The input to the system for this analysis is the acceleration of the hinge of the pendulum ($u = \ddot{x}$). On the actual system you will use torque applied to the first link to generate this acceleration.

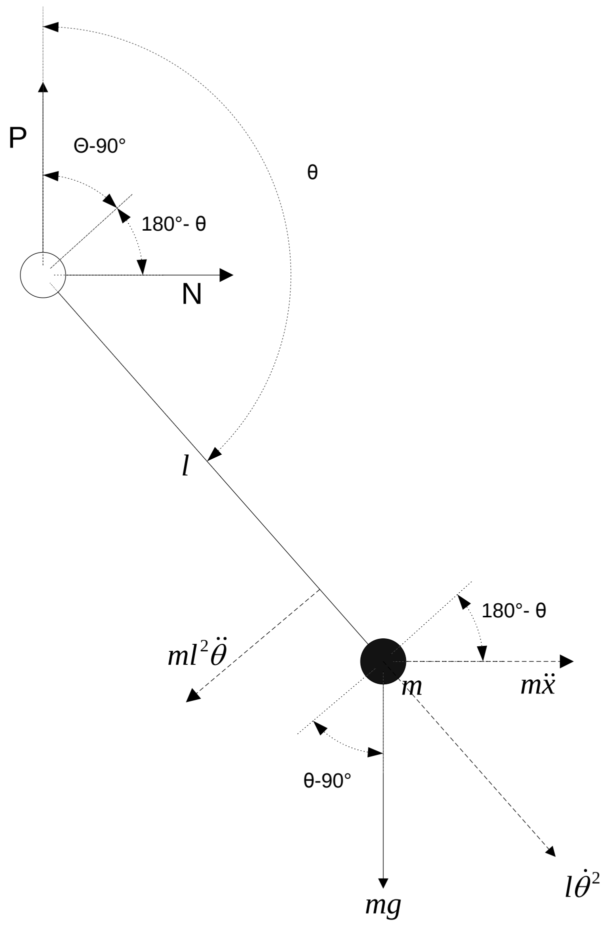


Figure 2 Free body diagram of point mass pendulum accelerated at value $\ddot{x}$.

To make the derivation simpler by not having to take into account the centripetal force, form the first equation of motion for this free body diagram by summing the forces perpendicular to the pendulum. The acceleration of the point mass is due to both $\ddot{x}$ and $l\ddot{\theta}$ so

$$P\cos(\theta - 90) + N\cos(180 - \theta) - mg\cos(\theta - 90) = m\ddot{x}\cos(180 - \theta) - ml\ddot{\theta} \qquad (10.2)$$

The second equation of motion that will help with this derivation is to sum the torques about the point mass of this idea pendulum. In this case since we are assuming the entire mass of the system is at this one point, the moment of inertia for the rest of the system is zero. Therefore the sum of the torques about the point mass is equal to zero.

$$Pl\cos(\theta - 90) + Nl\cos(180 - \theta) = 0 \tag{10.3}$$

Dividing equation 10.3 by l and substituting into equation 10.2 we have

$$ml\ddot{\theta} = m\ddot{x}\cos(180 - \theta) + mg\cos(\theta - 90).$$

Simplifying with $\cos(\theta - 90) = \sin(\theta)$ and $\cos(180 - \theta) = -\cos(\theta)$ the dynamic equation becomes

$$ml\ddot{\theta} = mg\sin(\theta) - m\ddot{x}\cos(\theta). \tag{10.4}$$

Continuing our derivation, the equation of energy for this pendulum is

$$E = \frac{1}{2}ml^2\dot{\theta}^2 + mgl\cos(\theta).$$

In order to make sure that our control is always increasing the energy of the pendulum, the derivative of the energy should always be positive.

$$\frac{dE}{dt} = ml^2\dot{\theta}\ddot{\theta} - mgl\sin(\theta)\dot{\theta} = (ml\ddot{\theta})l\dot{\theta} - mgl\sin(\theta)\dot{\theta}$$

Substituting in equation 10.4

$$\frac{dE}{dt} = (mg\sin(\theta))l\dot{\theta} - (m\ddot{x}\cos(\theta))l\dot{\theta} - mgl\sin(\theta)\dot{\theta}$$

Canceling like terms

$$\frac{dE}{dt} = -m\ddot{x}l\cos(\theta)\dot{\theta}$$

Finally defining our input to be $u = \ddot{x}$, we have

$$\frac{dE}{dt} = -mul\cos(\theta)\dot{\theta} \tag{10.5}$$

Looking at equation 10.5 with constants m and l, feedback θ and $\dot{\theta}$, and input u, to make $\frac{dE}{dt}$ always positive the following rule must be held

```
$if\,(\cos(\theta)\dot{\theta} > 0)\{$
$\rightarrow u = negative;$
$\}else\{$
$\rightarrow u = positive;$
$\}$
```

Following this rule, a swing-up control can be implemented. Iwashiro, Furuta and Aström[1] implements a control law that follows this rule and also looks at the error between a desired energy value and the actual energy. If you would like to implement this algorithm feel free. *(Use ml^2 = 0.0697 and ml = 0.3976. Note these parameters take into account the* *torque gain converting PWM units to Newton-Meters.)* But first implement a simpler form. Find a constant “c” so that the following algorithm swings the pendulum to the balancing point.

```
$if\,(\cos(\theta)\dot{\theta} > 0)\{$
$\rightarrow u = -c;$
$\}else\{$
$\rightarrow u = c;$
$\}$
```

Use the shell code given here to implement this control law along with your balancing control. Your demo should swing the linkage from the resting down position to the inverted balancing position and then catch and balance it there. If the pendulum is hit with a large tap the swing-up control should switch back on and swing the pendulum back up to the balancing position.

Lab Check Off:

1. Demo your working Simulink simulation using the non-linear Furuta model.
2. Demo your working full state feedback controller with friction compensation added.
3. Demo your controller accepting a θ_1 set-point change from MATLAB.
4. Demo your swing-up and catch code.
5. Hand in a printout of the MATLAB commands you used to design the different parts of the final controller.

[1] M. Iwashiro, K. Furuta, K. J. Aström, Energy Base Control of Pendulum, Proceedings of the 1996 IEEE International Conference on Control Applications, Dearborn, MI, September 15-16, 1996..