Lab 6

Final Lab

18-474 Embedded Control Systems

Shastri Ram

Athma Narayanan

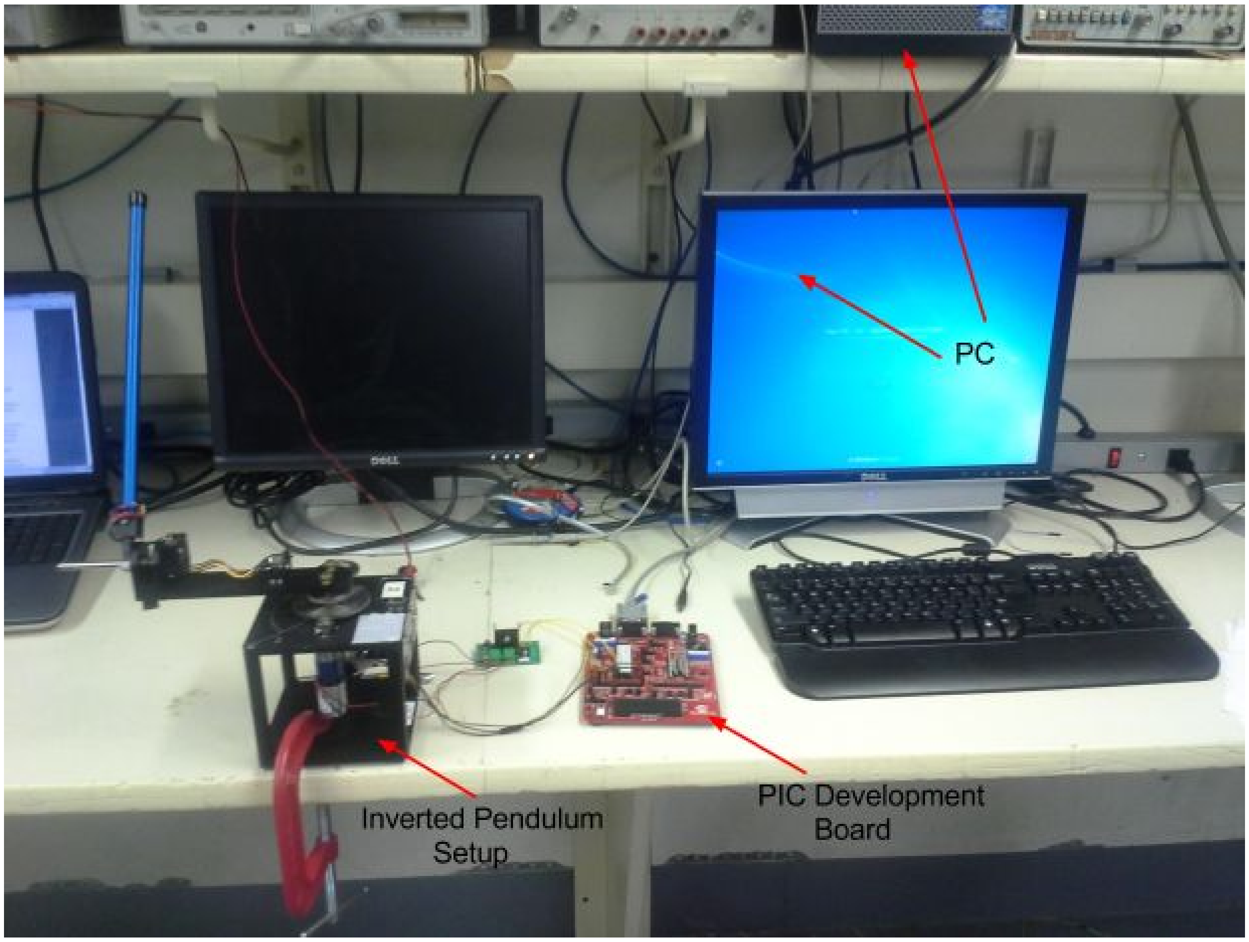
Concept

Proposed Feedback Concept

We propose a classical PID loop as the control for this system. Firstly, we would identify the model of the system using the Lagrangian method. From initial thought, we propose the system to be a single input, multiple output system. The single input will be the reference position of the pendulum which will be vertically up, that is, an angle of zero radians. The outputs of the system will be the angular position of the rotating arm and the angular position of the pendulum. Each of these would be fed back into its own PID loop. The system will be simulated on MATLAB using Simulink and then written in software. Some of the software, mainly the control loops will be run on a PC using MATLAB. MATLAB will send the control commands to the PIC development board via a serial connection. The PIC will then send the control voltage to the motor. In return, the PIC will send the position of the rotating arm and inverted pendulum via serial to MATLAB so that it can perform the calculations.

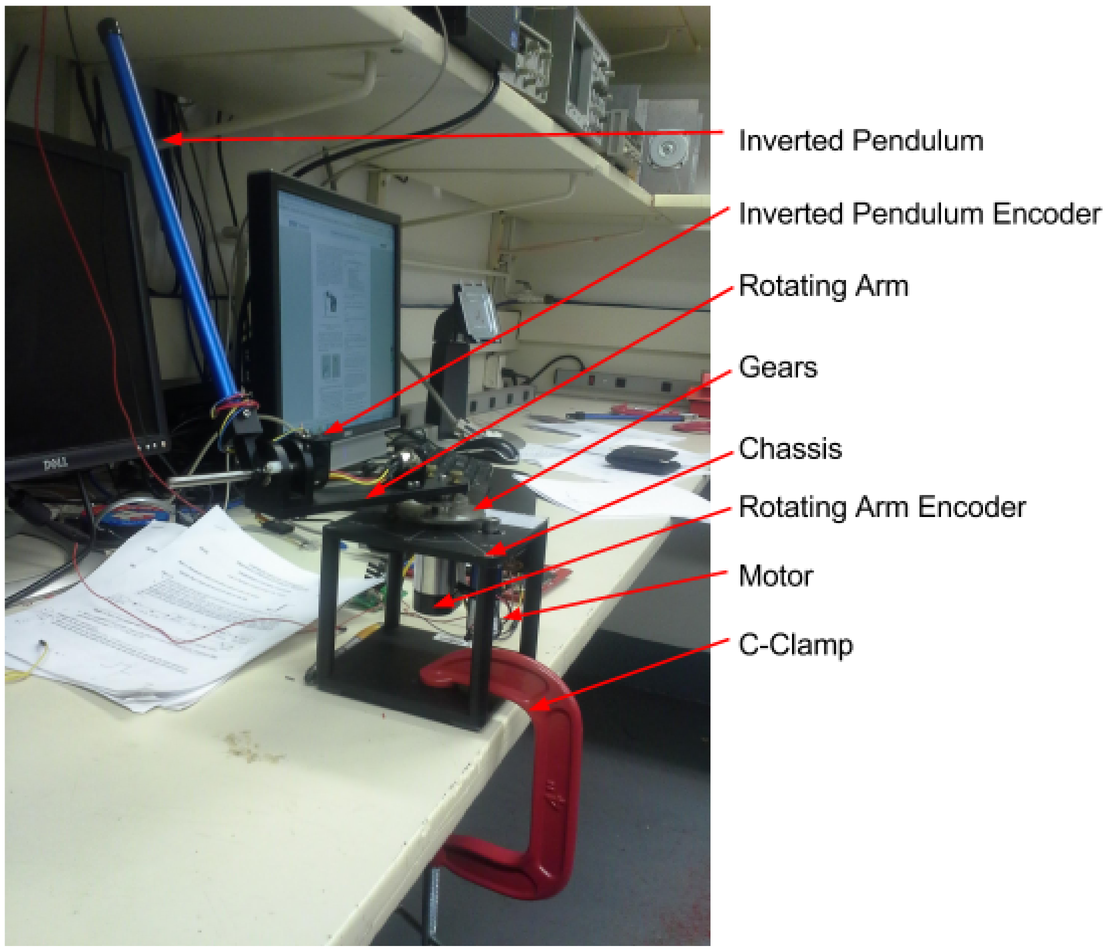
Hardware Setup

Overall Hardware Setup

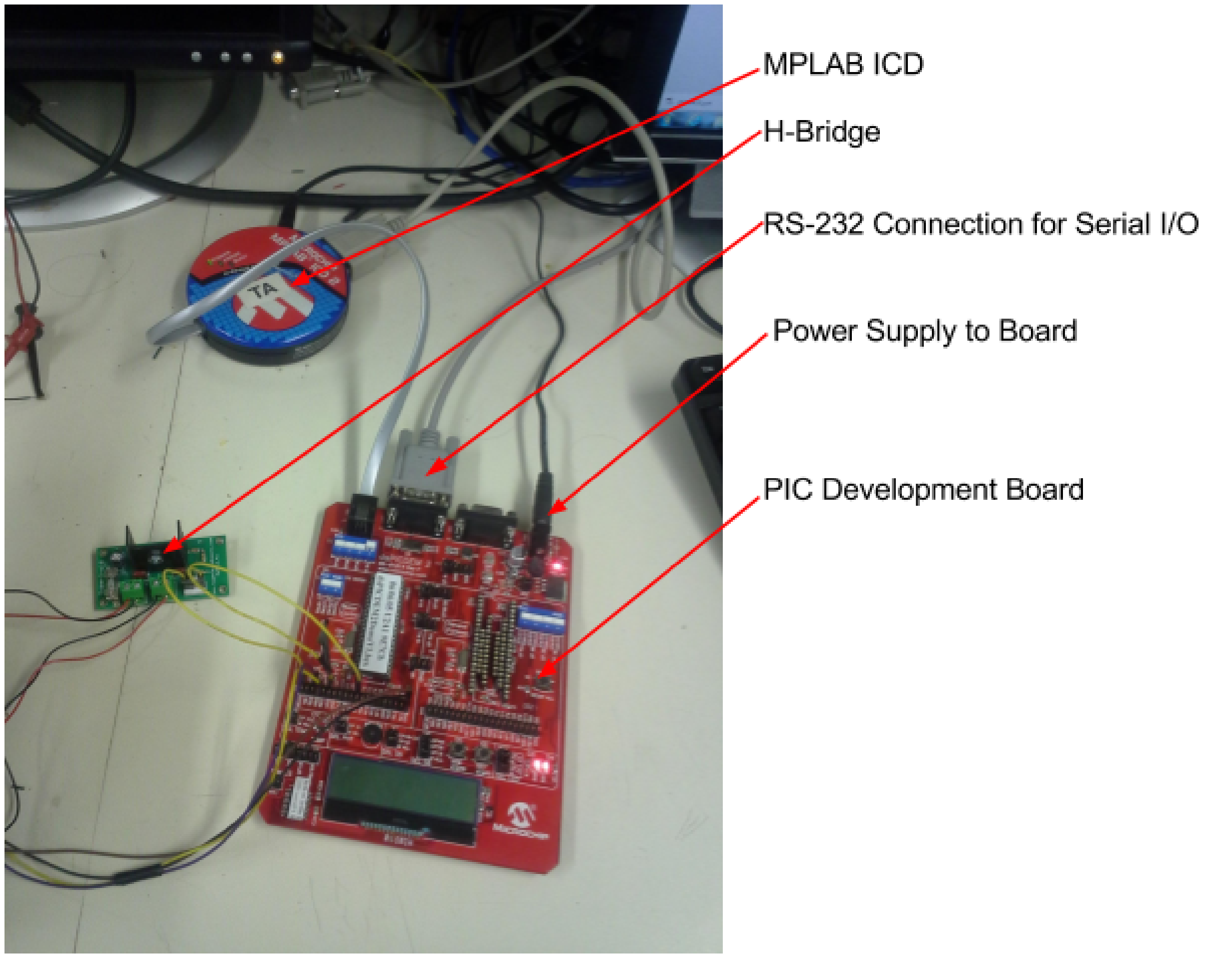


ya

Inverted Pendulum Setup



PIC Development Board Setup



Software design

The software will be divided into two portions, the MATLAB portion and the PIC portion. The MATLAB portion will run the graphical user interface and handle the serial communication, while the PIC portion of the code will run the control code for the lab.

MATLAB Portion

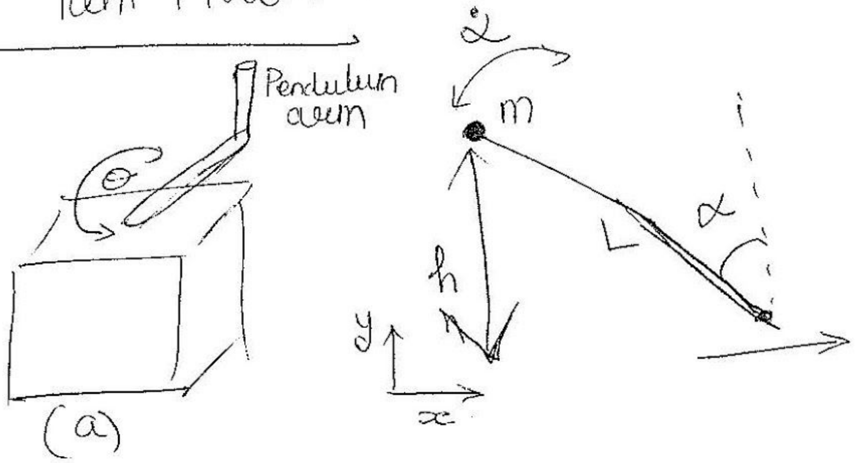
The MATLAB graphical user interface will be constructed using the MATLAB gui toolbox. The toolbox would allow the user to start and stop the program. Additionally it would allow the user to specify the numerator and denominator each of the gains. Matlab would send these values to the PIC via serial so that the user does not have to recompile and reflash the chip every time the values change.

PIC Portion

The PIC will be running the control code. It will read in the encoder value of the position of the pendulum, alpha. Additionally, it will read the potentiometer which gives the position of the rotating arm, theta. We aim to develop two PID loops to use to control the arm. One PID will set the control value for theta and the other PID will set the control signal for alpha. These values would be subtracted from each other and the result will be used as the PWM signal for the motor of the rotating arm.

The PIC may be programmed to send back encoder positions to store a log of values so that further analysis can be done.

2) Plant Model



① Velocity:

V_x: x component of velocity = V_{arm} ! V_x of pendulum

$= r\dot{\theta} + -\left(L\cos\alpha\,(\dot{\alpha})\right)$

$= r\dot{\theta} - L\cos(\alpha)(\dot{\alpha})$

V_y: y component of velocity $= -L\sin(\alpha)\,(\dot{\alpha})$

② Lagrange:

$L = K.E - P.E$

P.E = Potential energy = $mgh = mg\left(L\cos(\alpha)\right)$

$K.E = K.E_{HUB} + K.E_{Pendulum} + K.E_{x\ \text{Pendulum C.M}} + K.E_{y\ \text{Pendulum C.M}}$

Equations of Motions are given by.

$$\text{①}-\left\{\begin{aligned} &\frac{\delta}{\delta t}\left[\frac{\partial L}{\partial \dot{\theta}}\right]-\frac{\delta L}{\delta \theta}=\text{Torque}-\underbrace{\boxed{B_{eq}}}_{\text{damping coeffecient}}\dot{\theta} \\ &\frac{\delta}{\delta t}\left[\frac{\partial L}{\delta \dot{\alpha}}\right]-\frac{\delta L}{\delta \alpha}=0 \end{aligned}\right.$$

② —

$$\text{Torque}=\underset{\text{need to Find}}{\boxed{K_T}}{}^{*}\ \underbrace{V_m}_{\text{supplied Voltage}}$$

∴ Using Matlab, and Solving for ① + ② eqs leniarizing @ $\alpha=0$

$$\left(J_{HUB}+mr^2\right)\ddot{\theta}-mLr\,\ddot{\alpha}=T_{output}-B_{eq}\dot{\theta}$$

$$\frac{4}{3}mL^2\ddot{\alpha}-mLr\ddot{\theta}-mgL\alpha=0$$

$$\begin{bmatrix}\dot{\theta}\\ \dot{\alpha}\\ \ddot{\theta}\\ \ddot{\alpha}\end{bmatrix}=\begin{bmatrix}0&0&1&0\\0&0&0&1\\0&\frac{bd}{E}&-\frac{cG}{E}&0\\0&\frac{qd}{E}&-\frac{bG}{E}&0\end{bmatrix}\begin{bmatrix}\theta\\ \alpha\\ \dot{\theta}\\ \dot{\alpha}\end{bmatrix}+\begin{bmatrix}0\\0\\cK_T/R_m\\bK_T/R_m\end{bmatrix}V_m$$

Where,

1) $a = \underset{\substack{\uparrow \\ \text{unknown ①}}}{\underline{J_{HUB}}} + mr^2$

2) $b = mLr$

3) $c = \frac{4}{3} mL^2$

4) $d = mgL$

5) $E = ac - b^2$

6) $K_T = \underline{\eta_m} \, \eta_g \, \underline{K_t} \, \underline{K_g}$ $\Rightarrow$ ④ unknowns

7) $G = \dfrac{K_T - \boxed{B_{eq} R_m}}{R_m E}$ $\Rightarrow$ ⑤ ⑥ unknowns

Plant Model is Given as

$$\begin{bmatrix} \dot{\theta} \\ \dot{\alpha} \\ \ddot{\theta} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & bd/E & -cG/E & 0 \\ 0 & ad/E & -bG/E & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \alpha \\ \ddot{\theta} \\ \ddot{\alpha} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ cK_T/ER_m \\ bK_T/ER_m \end{bmatrix} V_m$$

<u>Parameters</u>

1. α – Pendulum position – From Encoder
2. $\dot{\alpha}$ – Pendulum velocity
3. $\ddot{\alpha}$ – Pendulum Acc
4. B_{eq} – Viscous Damping coeff. – Unknown

5. g - gravity of acc

6. J_{HUB} - moment of Inertia of arm and pendulum - Unknown

7. m - mass of pendulum

8. r - rotating arm length

9. R_m - armature resistance

10. Torque - Applied torque - $\underline{K_T} \times V_{motor}$ $\hookrightarrow$ Constant $\rightarrow$ Unknown

11. θ - load shaft position - From encoder

12. $\dot{\theta}$ - load shaft velocity

13. $\ddot{\theta}$ - load shaft acceleration

14. η_m - motor efficiency

15. η_g - gearbox efficiency

16. K_t - Motor-torque constant

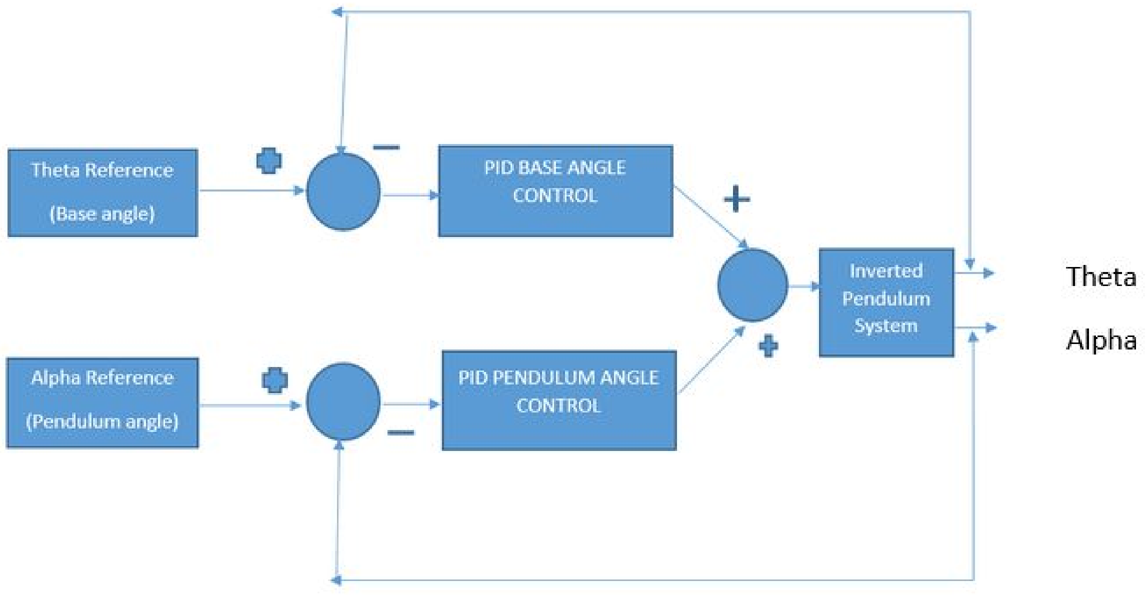
17. K_g - Motor-load constant.

} Unknowns.

Control architecture

Using the plant model from Step 3, draw a complete block diagram of the feedback controller that will be used to control the position of the system.

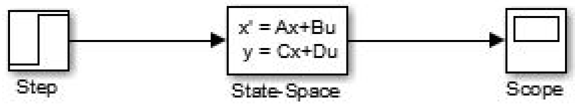
Your controller should use the signals that are available from the sensor(s). Explain your design procedure.

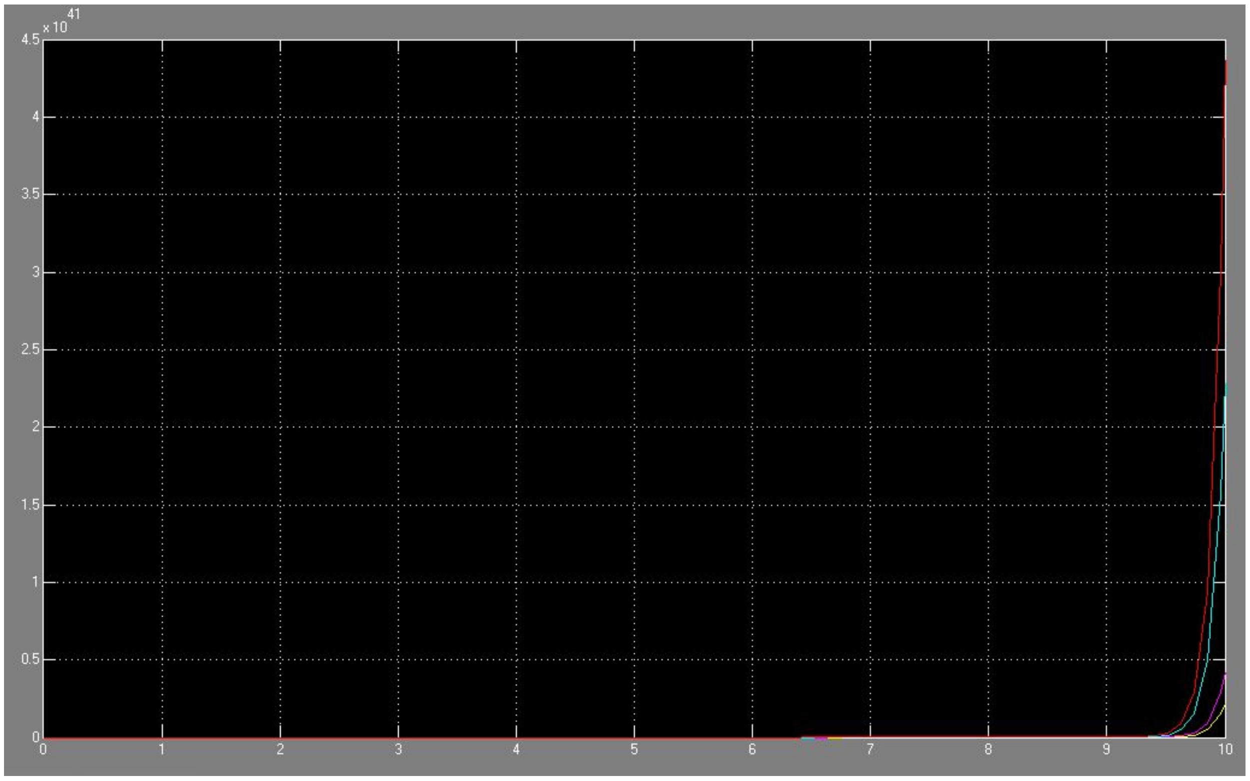


We intend to use a parallel 2 DOF PID controller for each angle(Theta,Alpha)

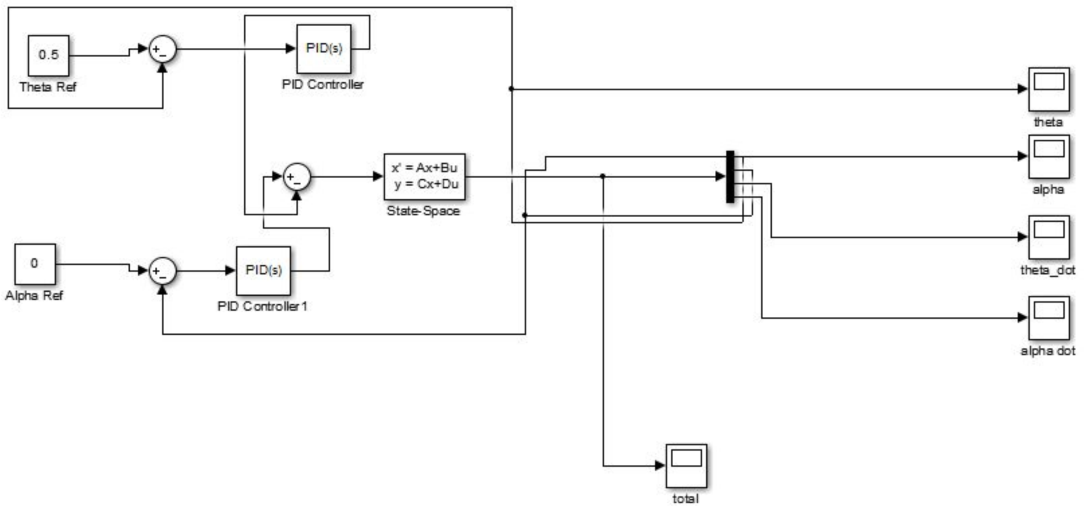
Simulink simulation

Step response of Simulink model of the dynamic system with no control

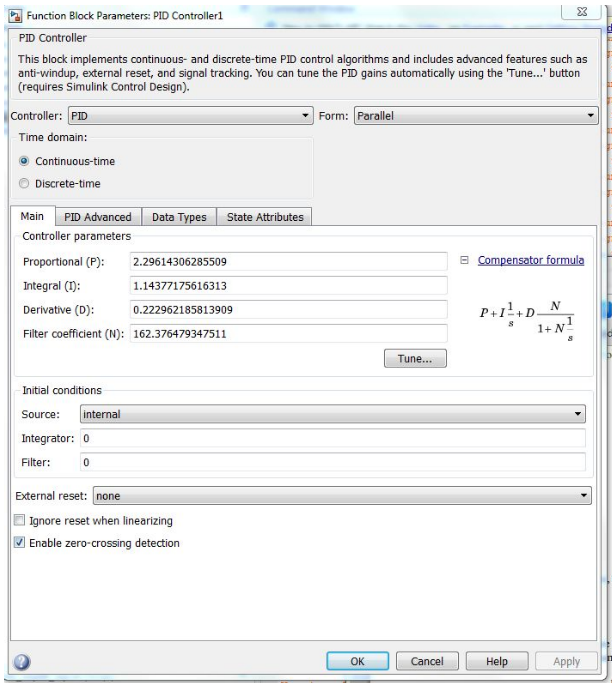




Full Control Model



Gains for PID Controller PID1 Controller



Function Block Parameters: PID Controller1

PID Controller

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PID Form: Parallel

Time domain:

- [✓ Continuous-time]
- Discrete-time

Main | PID Advanced | Data Types | State Attributes

Controller parameters

Proportional (P):	2.29614306285509
Integral (I):	1.14377175616313
Derivative (D):	0.222962185813909
Filter coefficient (N):	162.376479347511

Compensator formula

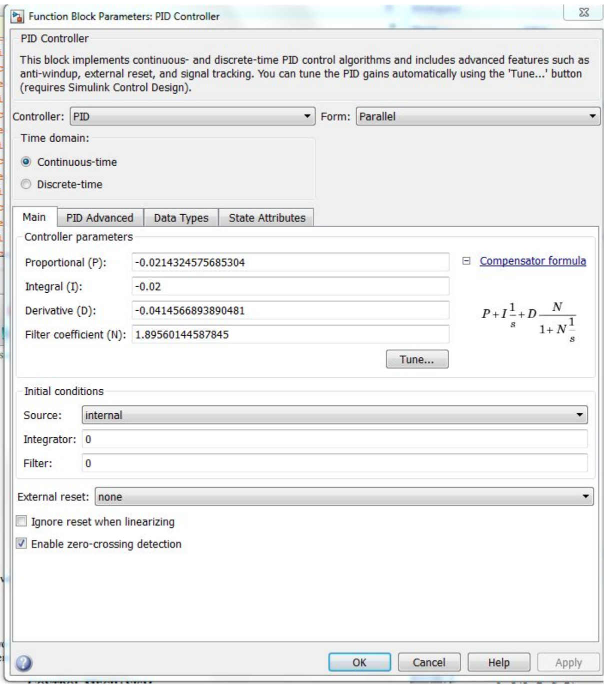
$$P+I\frac{1}{s}+D\frac{N}{1+N\frac{1}{s}}$$

Initial conditions

Source:	internal
Integrator:	0
Filter:	0

External reset: none

- ☐ Ignore reset when linearizing
- ☑ Enable zero-crossing detection



Function Block Parameters: PID Controller

PID Controller

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PID Form: Parallel

Time domain:

- [✓ Continuous-time]
- Discrete-time

Main | PID Advanced | Data Types | State Attributes

Controller parameters

Proportional (P):	-0.0214324575685304
Integral (I):	-0.02
Derivative (D):	-0.0414566893890481
Filter coefficient (N):	1.89560144587845

Compensator formula

$$P+I\frac{1}{s}+D\frac{N}{1+N\frac{1}{s}}$$

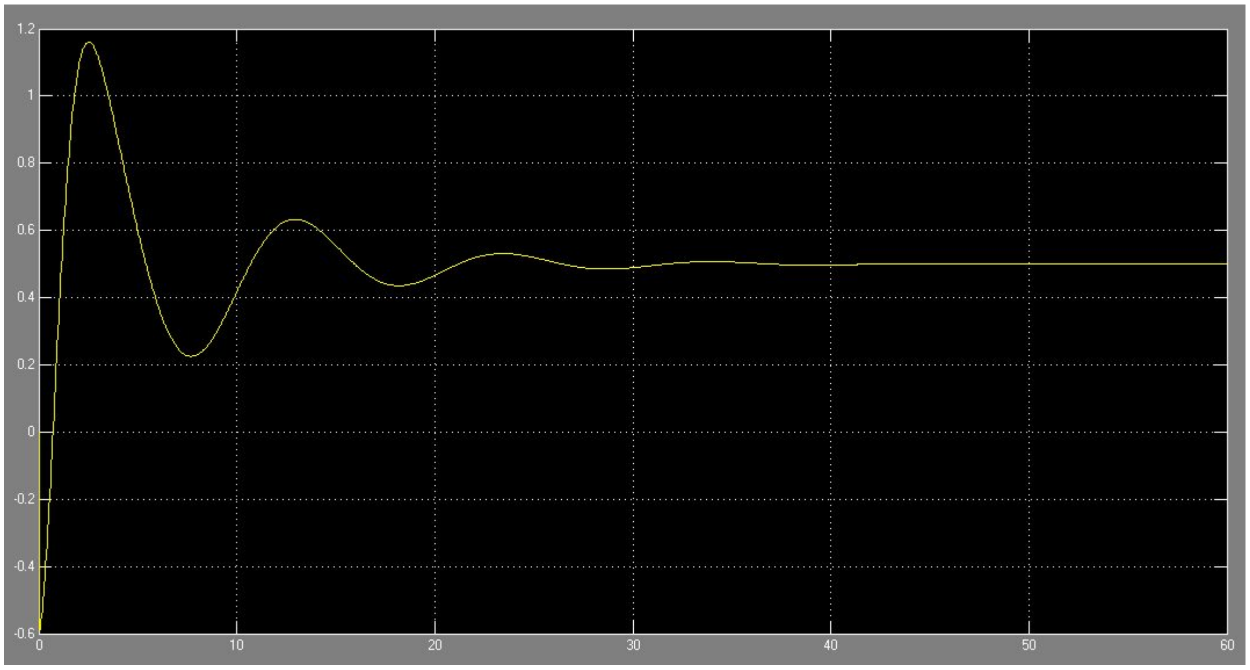
Initial conditions

Source:	internal
Integrator:	0
Filter:	0

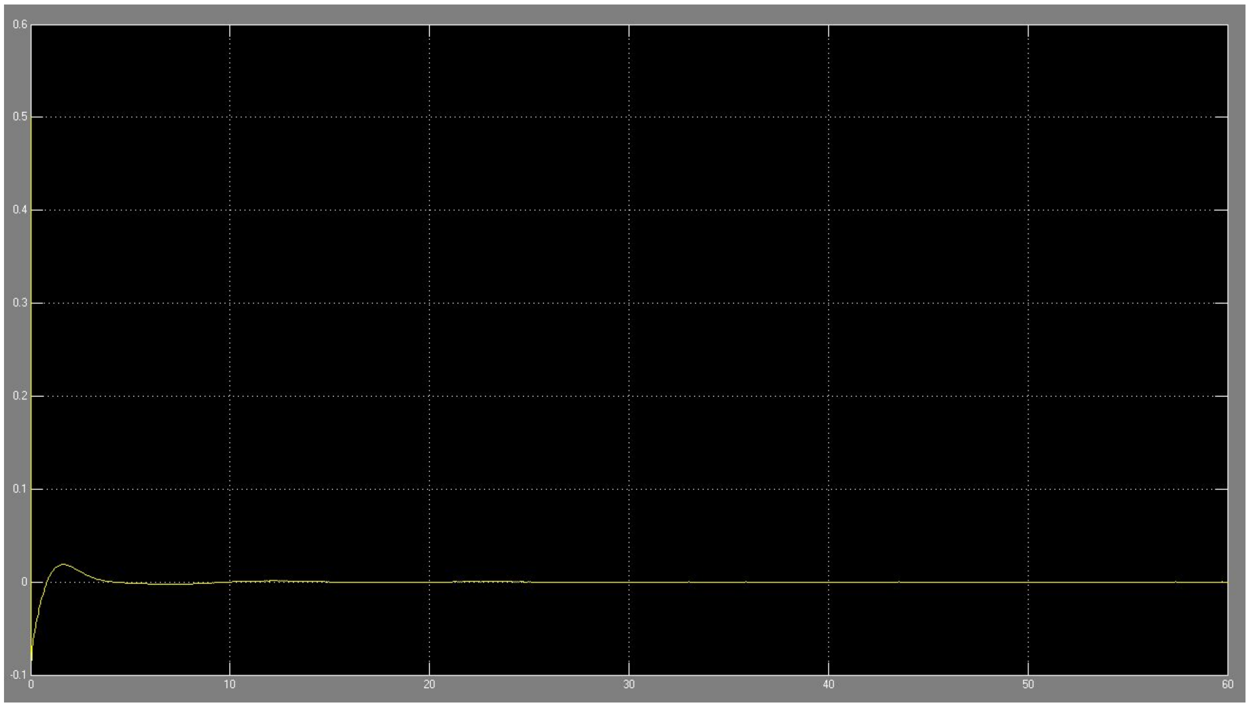
External reset: none

- ☐ Ignore reset when linearizing
- ☑ Enable zero-crossing detection

Theta Response

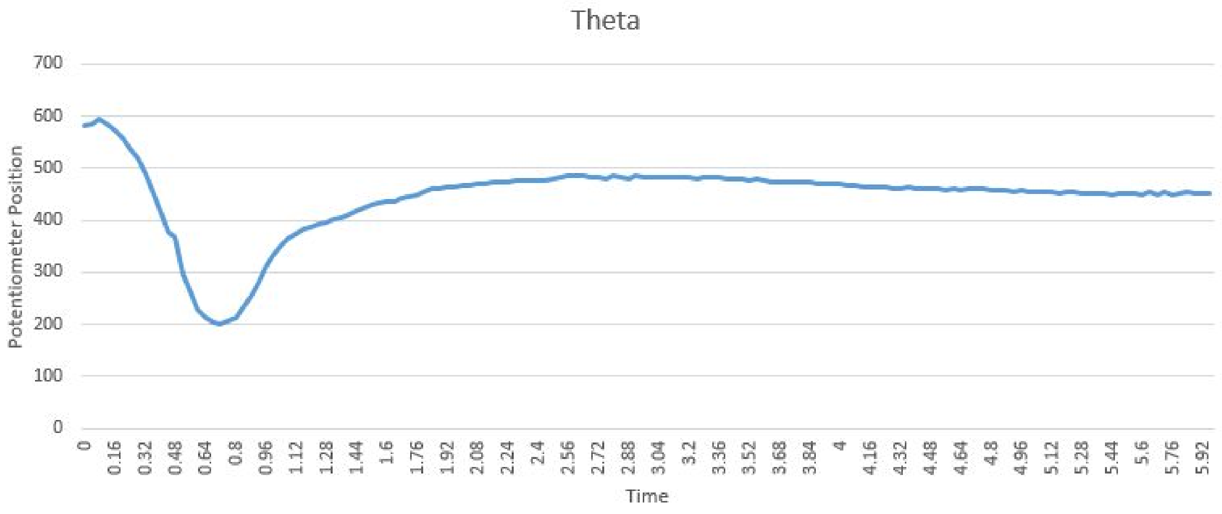


Alpha Response

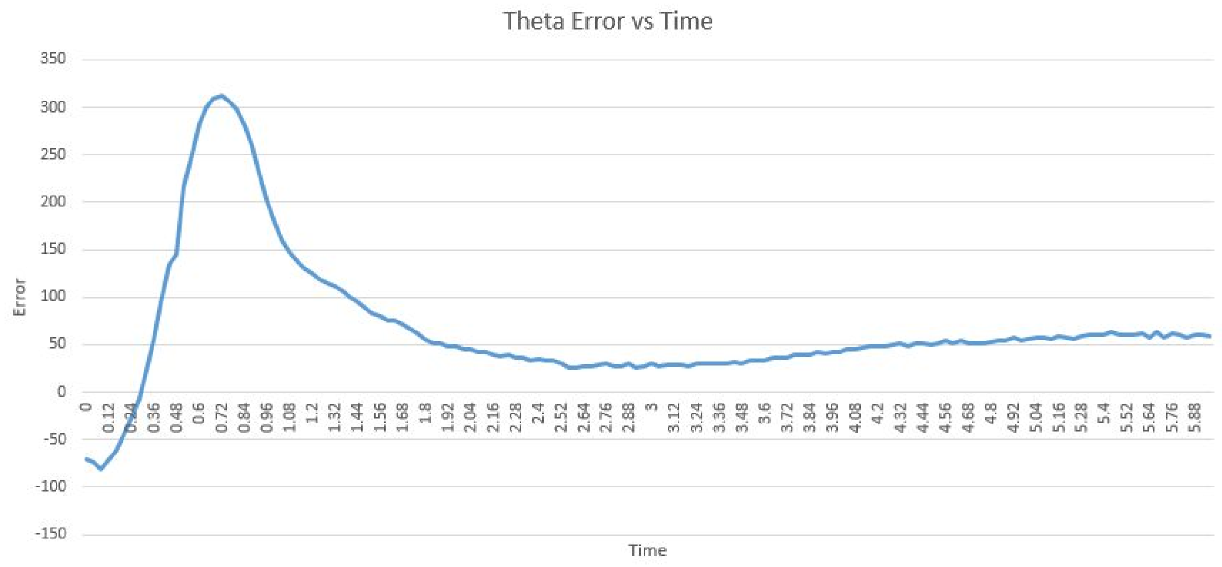


Experimental Results

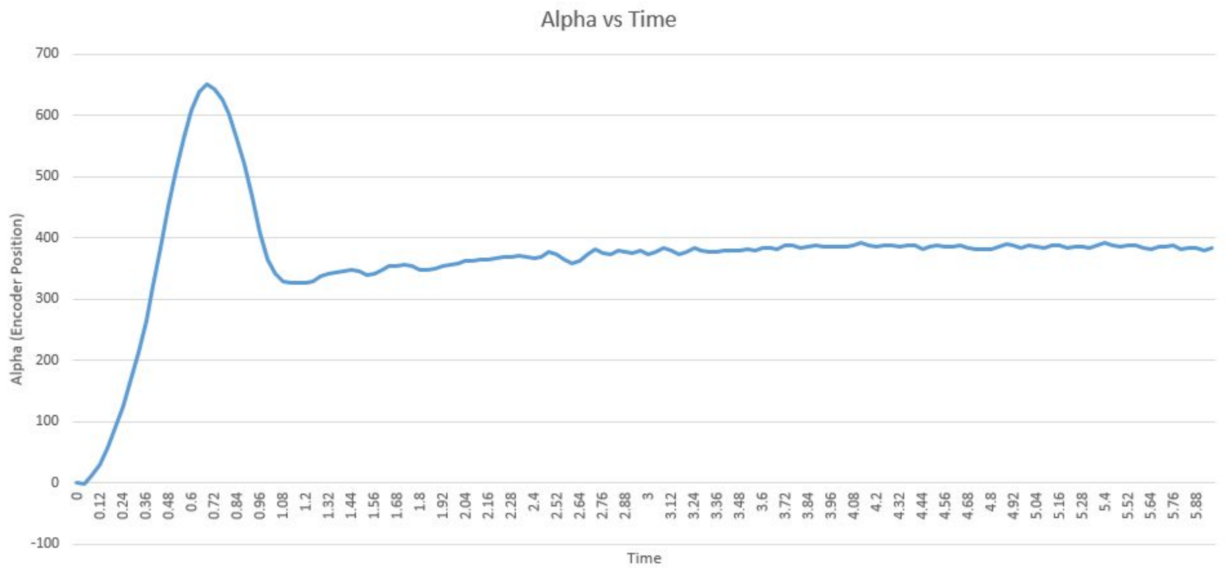
The following shows a graph of theta vs time. Theta is the angle of the rotating arm. It can be seen that it settles down around a steady value around 450.



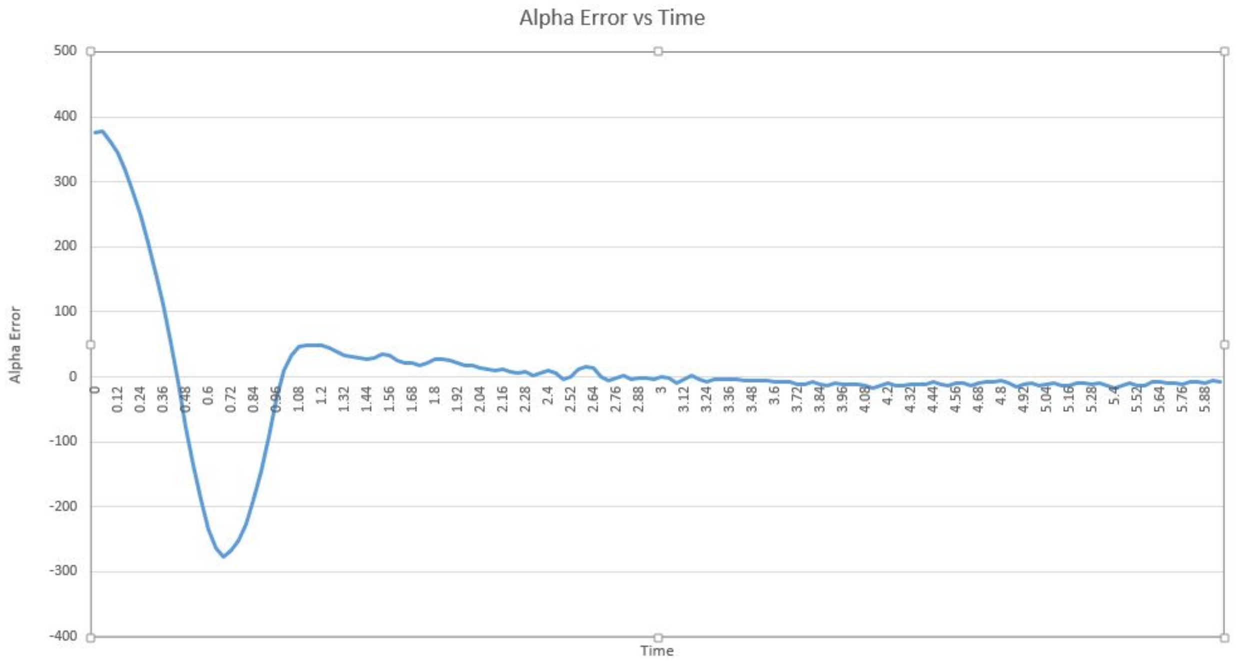
The reference position for theta was 512. This difference between the reference position and the actual position that it settled down to must be due to the two different set of gains, that is alpha gains and theta gains, working together. There is a greater preference to the alpha gains because the main aim is to balance the pendulum. Below shows the graph of theta error vs time. Though the error does not go to zero, due to the reason described above, the error goes to a constant value.



The following graph shows alpha, that is the pendulum position vs time. The alpha reference value was 375, which is the position that the pendulum would be balanced. It can be seen that the alpha values approach the reference value and stay within a small range of it.



The following graph shows alpha error vs time. It can be seen that the alpha reduces to zero and stays very close to zero. This is what we wanted to achieve.



In the end, our implementation was successful. The pendulum balanced quite well and was very responsive to disturbances. When perturbed, the system corrected itself and returned to the desired reference position very quickly.

The problems we encountered with this lab were not the control system itself, but the quirkiness of the PICC board itself. Initially we had all our values as long, which we realized did not work because it took too long for calculations to occur. The fix for this was to use all values as integers.

Sometimes, the board would not read the encoders or the potentiometer well and then when the system did get a reading, it would send very large pwm values with would drive the system crazy. The wires would get caught in the gears and we had to replace and resolder wires and the encoder pins multiple times.

For the GUI, we inputted a float number into the interface and MATLAB converted it to a numerator and denominator and then sent that through serial. The system would not work at all, but when we hardcoded the gains into the PICC board, the system would work sometimes. There were two problems here. Firstly there was some typecasting that we needed to do in MATLAB before we sent the values to the PICC. Secondly, if the numerator and denominator were too large values, the control code would not be evaluated well because either overflow or underflow occurred. Thus the fix for this was to send the numerator and denominator values ourselves instead of having MATLAB calculate it and also ensure that the values were small values. For instance if we wanted a gain to be 0.1, it is better to send it as numerator = 1, denominator = 10, instead of numerator = 100, denominator = 1000.

Once these little bugs were fixed, the system performed flawlessly.

Tasks and Milestones.

Task	Date Accomplished
Equipment testing and interface with PIC	26th April
Matlab Interface+GUI	3rd May
PID control addition and testing	4th May
Fine tuning	5-6th May

Concerns

The main thing we would have needed was probably more time. This would have enabled us to fine tune even more to get an even better response. We would have probably included code in the PICC and GUI to automatically obtain and plot the response of the system to aid us in debugging and tuning gains.