**Abstract:**

The purpose of this project is to create a hardware setup of an inverted pendulum, which is a classic controls theory problem. This project will attempt to incorporate various full system designs with three diverse departments of engineering software, hardware, and mechanical design. Main subsystems that comprise this design: (a) a controller and its interface to the mechanical system (b) the feedback control which includes encoder, sensors and a method to read (c) the mechanical system. After configuring the I/O pins of the motor controller will determine sets of requirements of the subsystems to simplify the design process. The feedback control consists of encoders, which were sampled by the concept of sampling theory by an analog to digital converter, to measure the displacement of the cart and the angle of the pendulum. The motor drive control uses pulse width modulation to vary the speed of the motor. Atmel Mega 2560 microcontroller will be programed with a control algorithm. The final system will result in a cart which balances a pendulum for a limited period.

**Aims of the project and sub-systems:**

An inverted pendulum is not a stable system and requires a constant external force for it to maintain its balance. It’s a classic controls problem and deals with the development of a system to balance the pendulum.

The aim of the project is to create a control system that allows us to balance an Inverted Pendulum, by building a closed loop control system.

The concepts are based on theory from Lab 12, as well as other labs to successfully do a practical implementation of the controls problem. The project demonstrates the capability of PID Control theory, along with an introduction in interfacing motor controls, through the BLDC Drive that’s being used for this application.

The project consists of Mechanical, Electrical hardware, and software implementation of algorithms to balance the pendulum.

1. The mechanical system consists of a cart, with gears, which are driven by a BLDC motor, a screw joint for the pendulum, and providing mounting for the motor.
2. The Electrical hardware consists of the following: Feedback Sensors for the rotor angle of the motor and the pendulum’s angle via Encoders, A Motor and Motor Drive System (in this case, a BLDC and BLDC Drive), the actual target MCU, where the algorithm is deployed, an LCD for status display, etc
3. In the software implementation, we do the Pin Mapping and functionality implementation of the Motor and Motor drive, as well as the closed loop control system that allows us to balance the pendulum, via code deployed on our ATMega2560.

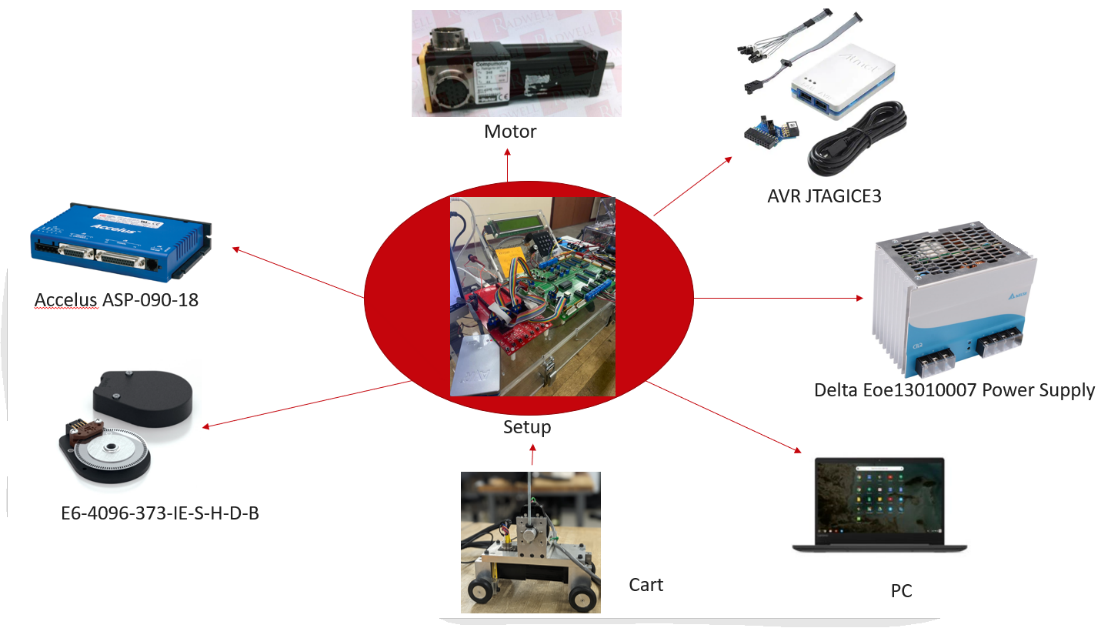
**Deliverables:**

1. Successful integration of the BLDC Motor Controller to Microchip, via correct pin mapping and functionality verification.
2. Successfully operational movement of the Cart along x-axis.
3. Actual balancing of the pendulum operation by the hardware setup.
4. Compensation algorithm for group delay for asynchronous sampling
5. BLDC motor feedback motion control system
   1. Encoder electrical wiring
   2. BLDDC-Motor electrical connections
   3. Motor Controller interfaces
   4. Encoder Interface for Motor and Pendulum
6. PID Control flow for Motion control and pendulum
7. C program for:
   1. Sampling of encoder analog signals from channel A and B, for both Motor and Pendulum
   2. Rollover compensation
   3. Position estimation from analog signals.
   4. Writing motion commands for the cart, which balances the pendulum.

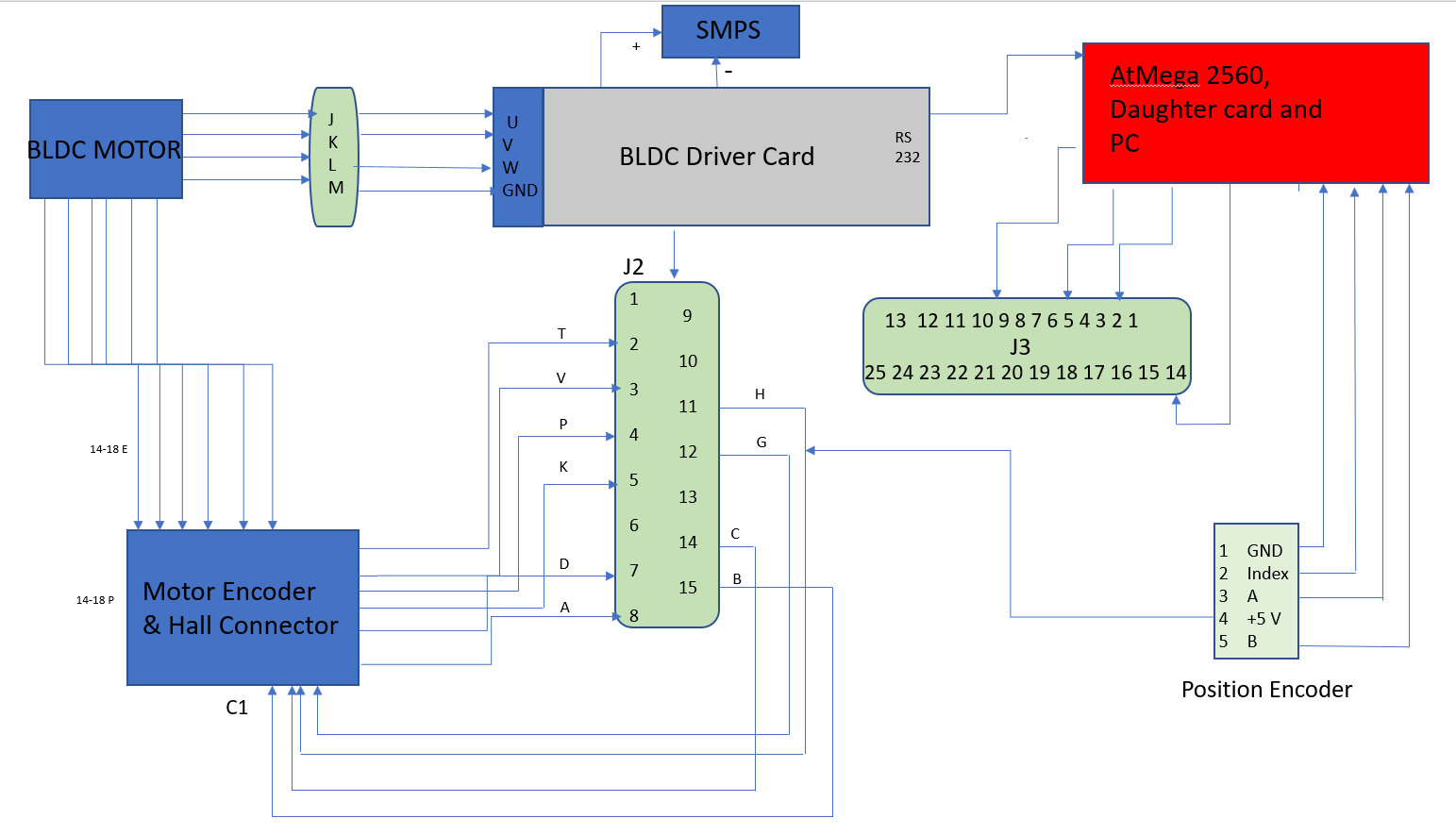
**Equipment Required:**

1. BLDC Motor
2. BLDC Motor Controller
3. Regulated Power Supply
4. Cart type setup, and a pendulum with fixtures.
5. Microchip Studio
6. CodeVisionAVR
7. JTAG in-circuit emulator
8. AVR Development Platform
9. Arduino MEGA2560 Rev3
10. JTAG/ISP/EXP Board Rev2
11. RS-232 to USB
12. USB Cable(s)
13. Oscilloscope

**Hardware System Overview:**



**Connections Diagram:**



**Physical Connections:**

**J3 of BLDC Card to ATMega2560:**

PIN2- TO ADC(0) +VE

PIN5- PORTB any pin

PIN10- ANY GND PIN

PIN14- TO ADC(0) -VE

**PENDULUM ENCODER TO ENC0 of daughter card:**

GND to ENC0 gnd

A to ENC0 A pin

B to ENC0 B pin

+5V- either through BLDC Card or to ENC0 +ve

**Connections between BLDC Motor and BDLC Driver are through 14-18 and 14-12 connectors from Amphenol, with mating connectors available.**

**Labs Covered:**

1. **Passive Circuits, Filters, & Op-Amps Op-Amp Signal Conditioning:**
   1. Content from this lab is used to make the circuit diagrams.
2. Not used
3. **Processors, C Programming Fundamentals Programming & Debugging Environment**
4. Content from this lab is used to program the microcontroller in C language.
5. **C Programming Fundamentals, Digital I/O, Digital Logic & Computer Math**

a. Content from this lab is used to advance c programming for the microcontroller

b. Configure the Digital I/O pins for the motor controller.

1. **Interrupts, timers, advanced C Concepts:**
   1. Content from these labs will be used in setting up various timers for position sampling, computing PD Difference equations, and uploading positional feedback inside the ISR (From lecture 13
2. Not used
3. **State Transition Diagram Techniques & Keypad/Displays or HMI/LCD**
4. LCD will be used to display the status and position of the cart.
5. Not used
6. **A/D, Sampling Theory, D/A, Reconstruction A/D-D/A Lab with Aliasing** 
   1. D/A Converter used
7. **Encoders, Encoder Counters, LVDTs Encoder counter Driver lab - Encoder to LCD**
8. Content from this lab will be applied to the Encoder next to the pendulum to monitor the position of the pendulum. There will be another encoder used to determine the position of the wheel.
9. Not used
10. **Close Loop Position Servo Control Motor Control Lab with Data Logging**
11. From this module we will apply the concepts of control theory.
12. **Fixed point Motor Controller**
    1. Implement the controller using fixed point math.

Total correlation to course material 89/13 = 69.2%.

**Before the laboratory:**

1. Identify and document the Output bit patterns (global addresses) required to address the three encoder counters and the encoder counter reset on the Daughter card with the decoder’s gate signal both set and cleared.

OE ENC0 with gate bit cleared:0x06 – with gate bit set: 0x26

OE ENC0 with gate bit cleared:0x07 – with gate bit set: 0x27

OE ENC1 with gate bit cleared:0x08 – with gate bit set: 0x28

OE ENC1 with gate bit cleared:0x09 – with gate bit set: 0x29

OE ENC2 with gate bit cleared:0x0A – with gate bit set: 0x2A

OE ENC2 with gate bit cleared:0x0B – with gate bit set: 0x2B

1. Identify and sketch the Physical Connections between the BLDC Motor, BLDC Card (Motor Driver) and the development board setup.

**In the Laboratory:**

1. Use the CodeWizardAVR to create a project framework for the ATmega2560 with a clock frequency of 16.0000 MHz. Feel free to browse through the tabs you have available to configure in the CodeWizardAVR. Use Timer1 to set up a compare match interrupt at a rate of 100 Hz. Configure the appropriate ports on the ATmega2560 so that the ADDR and DATA 5x2 port headers on the JTAG/ISP/EXP Board Rev2 can be used to drive the ADDR & DATA inputs on the STK500/600 daughter card. Configure the ADDR port to allow for driving the 4-16 decoder on the STK500/600 daughter card. Configure the appropriate port for interfacing to a 4x40 character LCD. Configure PORTA to use the switch bank, and PORTB as a digital enable signal. Document which ports are used and all specified initialization code here.

PORTK is mapped to ADDR1; PORTL is mapped to DATA1; PORTC is mapped to LCD

PORTK [7:0] ->Drives decoder (output)

PORTC-> LCD of Daughter card

PORTA->SWITCH

PORTB-> BLDC CARD ENABLE SIGNAL

#include <mega2560.h>

#include <math.h>

#include <encoder\_DA.h>

#include <write\_DA.h>

#include <reset\_encoder.h>

#include <stdlib.h>

#include <motorsystem.h>

#include <wrlatch.h>

#include <string.h>

/\* the LCD module is connected to PORTC \*/

#asm

.equ \_\_lcd\_port=0x08

#endasm

/\* now you can include the LCD Functions \*/

#include <lcd4x40.h>

#include <delay.h>

// Timer1 output compare A interrupt service routine

interrupt [*TIM1\_COMPA*] void timer1\_compa\_isr(void)

{

// Place your code here

}

// Input/Output Ports initialization

// Port A initialization

// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In

*DDRA*=(0<<*DDA7*) | (0<<*DDA6*) | (0<<*DDA5*) | (0<<*DDA4*) | (0<<*DDA3*) | (0<<*DDA2*) | (0<<*DDA1*) | (0<<*DDA0*);

// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T

*PORTA*=(1<<*PORTA7*) | (1<<*PORTA6*) | (1<<*PORTA5*) | (1<<*PORTA4*) | (1<<*PORTA3*) | (1<<*PORTA2*) | (1<<*PORTA1*) | (1<<*PORTA0*);

// Port B initialization

// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In

*DDRB*=(1<<*DDB7*) | (1<<*DDB6*) | (1<<*DDB5*) | (1<<*DDB4*) | (1<<*DDB3*) | (1<<*DDB2*) | (1<<*DDB1*) | (1<<*DDB0*);

// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T

*PORTB*=(1<<*PORTB7*) | (1<<*PORTB6*) | (1<<*PORTB5*) | (1<<*PORTB4*) | (1<<*PORTB3*) | (1<<*PORTB2*) | (1<<*PORTB1*) | (1<<*PORTB0*);

// Port C initialization

// Function: Bit7=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out Bit0=Out

*DDRC*=(0<<*DDC7*) | (0<<*DDC6*) | (0<<*DDC5*) | (0<<*DDC4*) | (0<<*DDC3*) | (0<<*DDC2*) | (0<<*DDC1*) | (0<<*DDC0*);

// State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0

*PORTC*=(1<<*PORTC7*) | (1<<*PORTC6*) | (1<<*PORTC5*) | (1<<*PORTC4*) | (1<<*PORTC3*) | (1<<*PORTC2*) | (1<<*PORTC1*) | (1<<*PORTC0*);

// Port K initialization

// Function: Bit7=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out Bit0=Out

*DDRK*=(1<<*DDK7*) | (1<<*DDK6*) | (1<<*DDK5*) | (1<<*DDK4*) | (1<<*DDK3*) | (1<<*DDK2*) | (1<<*DDK1*) | (1<<*DDK0*);

// State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0

*PORTK*=(0<<*PORTK7*) | (0<<*PORTK6*) | (0<<*PORTK5*) | (0<<*PORTK4*) | (0<<*PORTK3*) | (0<<*PORTK2*) | (0<<*PORTK1*) | (0<<*PORTK0*);

// Port L initialization

// Function: Bit7=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out Bit0=Out

*DDRL*=(1<<*DDL7*) | (1<<*DDL6*) | (1<<*DDL5*) | (1<<*DDL4*) | (1<<*DDL3*) | (1<<*DDL2*) | (1<<*DDL1*) | (1<<*DDL0*);

// State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0

*PORTL*=(0<<*PORTL7*) | (0<<*PORTL6*) | (0<<*PORTL5*) | (0<<*PORTL4*) | (0<<*PORTL3*) | (0<<*PORTL2*) | (0<<*PORTL1*) | (0<<*PORTL0*);

// Timer/Counter 1 initialization

// Clock source: System Clock

// Clock value: 2000.000 kHz

// Mode: CTC top=OCR1A

// OC1A output: Disconnected

// OC1B output: Disconnected

// OC1C output: Disconnected

// Noise Canceler: Off

// Input Capture on Falling Edge

// Timer Period: 10.001 ms

// Timer1 Overflow Interrupt: Off

// Input Capture Interrupt: Off

// Compare A Match Interrupt: On

// Compare B Match Interrupt: Off

// Compare C Match Interrupt: Off

*TCCR1A*=(0<<*COM1A1*) | (0<<*COM1A0*) | (0<<*COM1B1*) | (0<<*COM1B0*) | (0<<*COM1C1*) | (0<<*COM1C0*) | (0<<*WGM11*) | (0<<*WGM10*);

*TCCR1B*=(0<<*ICNC1*) | (0<<*ICES1*) | (0<<*WGM13*) | (1<<*WGM12*) | (0<<*CS12*) | (1<<*CS11*) | (0<<*CS10*);

*TCNT1H*=0x00;

*TCNT1L*=0x00;

*ICR1H*=0x00;

*ICR1L*=0x00;

*OCR1AH*=0x4E;

*OCR1AL*=0x20;

*OCR1BH*=0x00;

*OCR1BL*=0x00;

*OCR1CH*=0x00;

*OCR1CL*=0x00;

// Timer/Counter 1 Interrupt(s) initialization

*TIMSK1*=(0<<*ICIE1*) | (0<<*OCIE1C*) | (0<<*OCIE1B*) | (1<<*OCIE1A*) | (0<<*TOIE1*);

// Globally enable interrupts

#asm("sei")

1. Generate a framework application. (Give the C source file generated the name “main.c”. You may give the .prj & .cwp files the name Project.\*)

Done

1. Attach the appropriate (from those configured above) 5x2 port header on the JTAG/ISP/EXP BOARD REV2 to the “Address Bus” connector on the Daughter Card.

Done

1. Attach an appropriate 5x2 port header on the JTAG/ISP/EXP BOARD REV2 to the “Data Bus” connector on the Daughter Card.

Done

1. Attach the Pendulum’s encoder to ENC1 channel on the daughter card.

Done

1. Attach the appropriate connections between the BLDC Card and the motor, as well as BLDC Card to the development board and D/A Channel of the Daughter Card, from the connections diagram shown above.

Done

1. Add your previously generated digital input/output and DAC code to the project.

char write\_DA(char channel, char value)// This function has two input parameters channel number and value to load D/A

{

switch(channel)

{

case 0:

{

*PORTK*=0x20; // address of D/A channel gate signal high

*DDRL*=0xFF; // the data bus PORT as an output

*PORTL*=value; // the output value on the data bus PORT

delay\_us(2); // Wait for the data bus to stabilize delay

*PORTK*=0x00; // Clear the decoder gate signal (enables decoder)

delay\_us(2); // Wait for the decoder output to propagate to D/A chip

*PORTK*=0x20; // Set the gate signal to disable the decoder chip

}break;

case 1:

{

*PORTK*=0x21;

*DDRL*=0xFF;

*PORTL*=value;

delay\_us(2);

*PORTK*=0x01;

delay\_us(2);

*PORTK*=0x21;

}break;

case 2:

{

*PORTK*=0x22;

*DDRL*=0xFF;

*PORTL*=value;

delay\_us(2);

*PORTK*=0x02;

delay\_us(2);

*PORTK*=0x22;

}break;

case 3:

{

*PORTK*=0x23;

*DDRL*=0xFF;

*PORTL*=value;

delay\_us(2);

*PORTK*=0x03;

delay\_us(2);

*PORTK*=0x23;

}break;

default:

return 0;

}

}

1. Develop a function that takes in an 8-bit number representing the encoder channel number, reads the value from the correct encoder channel and returns the full encoder count (i.e. full dynamic range) in addition to an error if an invalid channel number is used. Valid channels are 0-2.

#include<mega2560.h>

#include<delay.h>

signed int encoder\_DA(char enc\_channel)//function that takes in an encoder channel number, reads the value from the correct encoder channel and returns the full encoder count

{

signed int count\_value;

char value,value1;

switch(enc\_channel)

{

case 0:

{

*DDRL*=0x00;

*PORTK*=0x06; //sets the address to the first encoder

delay\_us(2);

value=*PINL*; // reads the upper 4 of the 12-bit encoder's value

*PORTK*=0x26; //sets the address to the first encoder with gate bit set

*DDRL*=0x00;

*PORTK*=0x07;

delay\_us(2);

value1=*PINL*; // reads the subsequent lower 8 bits of the 12-bit encoder's value

*PORTK*=0x27;

count\_value= (((signed int) value)<<8 ) + value1 ;

return count\_value;

}break;

case 1:

{

*DDRL*=0x00;

*PORTK*=0x08;

delay\_us(2);

value=*PINL*;

*PORTK*=0x28;

*DDRL*=0x00;

*PORTK*=0x09;

delay\_us(2);

value1=*PINL*;

*PORTK*=0x29;

count\_value= (((signed int) value)<<8 ) + value1 ;

return count\_value;

}break;

case 2:

{

*DDRL*=0x00;

*PORTK*=0x0A;

delay\_us(2);

value=*PINL*;

*PORTK*=0x2A;

*DDRL*=0x00;

*PORTK*=0x0B;

delay\_us(2);

value1=*PINL*;

*PORTK*=0x2B;

count\_value= (((signed int) value)<<8 ) + value1 ;

return count\_value;

}break;

default: //invalid channel

return 0;

}

}

1. Generate a code segment to read an encoder channel repetitively and implement the rollover handling algorithm. Demonstrate this code to the instructor. Do this for the Pendulum as well.

presentPosition=encoder\_DA(0);

delta\_position = presentPosition - pastPosition ;

if (delta\_position > 2048)

{

delta\_position= delta\_position-4096;

}

if (delta\_position <- 2048)

{

delta\_position=delta\_position+4096;

}

positionActual = positionActual+delta\_position;

pastPosition=presentPosition;

//PENDULUM ENCODER POSITION

pend\_presentPosition=encoder\_DA(1);

pend\_delta\_position = pend\_presentPosition - pend\_pastPosition ;

if (pend\_delta\_position > 2048)

{

pend\_delta\_position= pend\_delta\_position-4096;

}

if (pend\_delta\_position <- 2048)

{

pend\_delta\_position=pend\_delta\_position+4096;

}

pend\_positionActual = pend\_positionActual+pend\_delta\_position;

pend\_pastPosition=pend\_presentPosition;

1. Add some code to display the rpm of delta position of the pendulum.

unsigned char \*str="Hello World";

rpm = ( (float)pend\_delta\_position\*100\*60)/(4\*500); //typecasted to allow the result be stored in a float to variable before the assignment

*ftoa*(rpm,2,str);

//code to clean up the lcd print statements

counter++;

if(counter==20)

{

lcd\_clear();

lcd\_puts(str);

counter=0;

}

1. Command the motor with several voltages over the range of D/A output and test for speed and direction changes.

write\_DA(0,128); // range [0,255]

1. Modify the new project to implement the PD Control Law (on E) difference equation using fixed point mathematics, modifying variable types as necessary. Include your fixed point algorithm code and applicable variable declarations in your report. Write some code to convert your floating point coefficients into fixed point values. Demonstrate your code properly converts your coefficients into the correct fixed point format for the given variables. Include this code inline here. Include code to create a reference position (ie of the pendulum) for the controller.

if(*PINA*==0x7f) //Dont forget to change to portA for BLDC CARD!

{

r\_N=pend\_positionActual; // assigns the vertical position's value as a reference every time the code is compiled and button pressed

start\_bit=1;

}

if(start\_bit==1)

{

e\_N=r\_N-pend\_positionActual;

// Fixed point Math Code

prod = (long)K\_0\_fixed\* (long)e\_N;

prod1 = (long)K\_1\_fixed\* (long)delta\_position;

m\_N = ((prod>>4) +(prod1))>>16;

m\_N=(K\_0 \* e\_N) +(K\_1\*e\_past);

e\_past=e\_N;

if(m\_N>127)

{

m\_N=127;

}

if(m\_N<-128)

{

m\_N=-128;

}

wrlatch(0xff);

write\_DA(0,m\_N+128);

}

else

{

wrlatch(0xff);

write\_DA(0,127);

}

//IN MAIN

K\_0=(K\_p+(K\_D/Ts));

K\_1=-1\*(K\_D/Ts);

K\_0\_fixed=K\_0\*pow(2,12);

K\_1\_fixed=K\_1\*pow(2,12);

1. Demonstrate functionality during final presentation!

Done

**After the laboratory:**

1. What are some of the main challenges faced by the team during the project?

Firstly, team started mapping I/O pins of the system. As there were more than seven complex devices to route the wiring, which was a challenge as there was high chances of shorting the circuit with numerous open-ended wires in the compact space. The Designing a controller that is close to ideal is a challenging design problem. Secondly, selecting the algorithm which will satisfy our requirements and setting up the gain values to which the pendulum will balance. Thirdly, motor calibrations and resolving the errors in the motor software called Copley.

1. What are some areas of future improvement for the project?

There are majorly three improvements areas which will help this project that were applying Linear Quadratic Regulator (LQR), Serial Peripheral Interface (SPI) and tuning the driving the cart.

There is an equations LQR for determining the control law algorithm which will apply control efforts into the system. LQR will provide high sampling rate and is an easiest implementation with minimal calculations.

Serial Peripheral Interface is the most widely used communication protocols, this will consume less power as compared to I2C but, also provide high hit rates. However, SPI incorporation requires higher number of wires in the bus which implies more pins on the microcontroller.

Tuning the phase of the motor driver to provide enough torque to drive chassis and balance the pendulum with the small displacement of the pendulum. Copley software needs to be explored to improve the motor driver tuning.

**References:**

* Erick L. Oberstar. ME 445 Lecture Notes, Fall 2022.
* <https://www.instructables.com/Inverted-Pendulum-Control-Theory-and-Dynamics/>
* <https://www.hackster.io/tloinny/open-source-inverted-pendulum-419c28>
* <https://hackaday.io/project/181812-inverted-pendulum-on-a-cart>
* <https://people.ece.cornell.edu/land/courses/eceprojectsland/STUDENTPROJ/2004to2005/jss67/Final_Report.pdf>
* <https://www.seas.upenn.edu/~jiyuehe/rotary-inverted-pendulum/index.html>
* <https://andreweib.wordpress.com/inverted-pendulum-project/>
* <https://wiki.aalto.fi/display/MEX/Inverted+Pendulum>
* <https://create.arduino.cc/projecthub/zjor/inverted-pendulum-on-a-cart-199d6f>
* <https://www.youtube.com/watch?v=qjhAAQexzLg>
* https://uwmadison.app.box.com/s/kr7vnqp3dv3ag6x2g88390gqo73hjzth