MECH423 Lab 3 Report

Ratthamnoon Prakitpong
#63205165

1. DC Motor Control

1.1. Describe your apparatus and code in your report and include a screenshot of your C# program. Attach your MCU and C# code as appendices.

I assembled the DC motor, its driver, and MSP430 board together using jumper wires and breadboard. The specifics of how they are connected is shown in section 1.2.

To set motor direction, you set AIN1 and AIN2 of motor driver via P3.5 and P3.6 on the board. To set motor speed, you can change PWM duty cycle of input signal to PWMA of motor driver from P3.4/TB1.1 of board. The specifics of these two controls are sent from laptop to board via UART packages.

In the C# app, UART can be started by selected the correct port using the combo box on top-left corner. You can use the track bar to set PWM, between 0 and 65536. The speed will be displayed in textbox, which you can also type in numbers to precisely set speed. Use checkbox to set direction. Changing PWM will trigger a UART package to be sent to board.

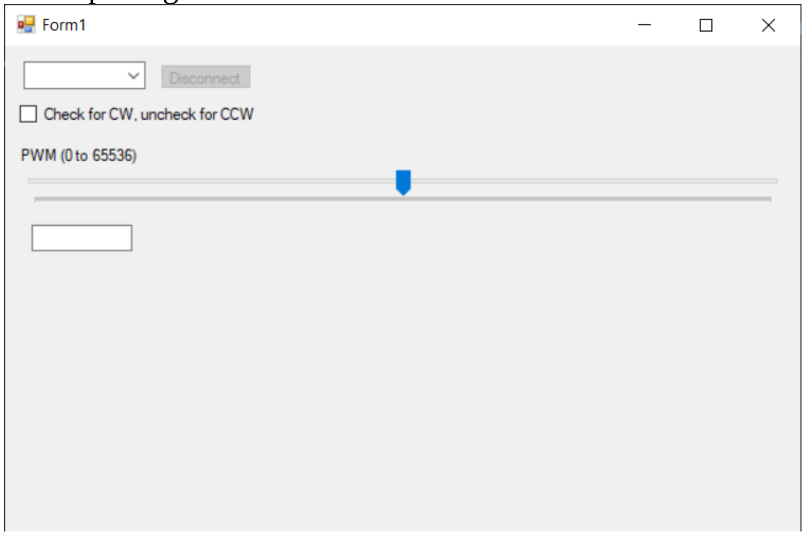


Fig 1.1: C# program window

The MCU and C# code are in Appendix 1.

1.2. Draw an electrical schematic diagram of the minimum components necessary to drive a DC motor including the MCU, motor driver, power supply, and accessories.

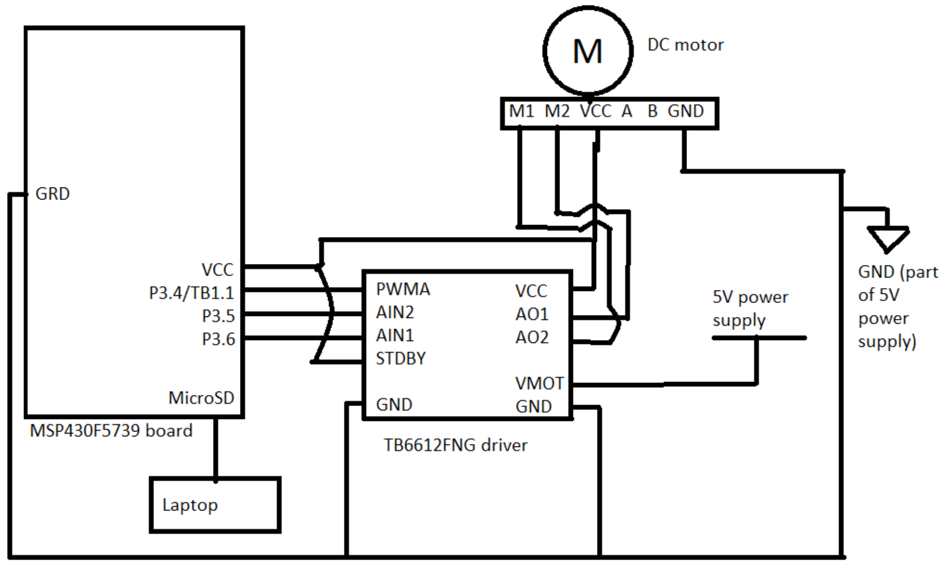


Fig 1.2: Electrical schematic diagram for exercise 1

Wire connections were made with jumper wires and a breadboard.

1.3 What is the minimum PWM duty cycle for the motor driver to generate a reasonable output waveform? Check using an oscilloscope. Why does this limit exist? Discuss in your report.

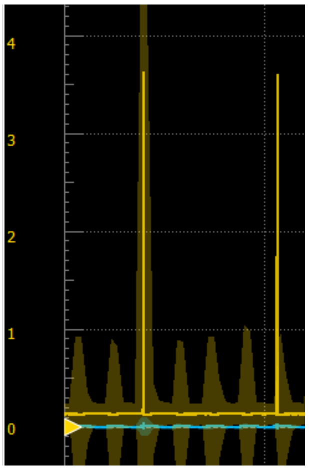


Fig 1.3: Minimum duty cycle

Minimum limit of PWM duty cycle is about 0.3% (~200/65536). At lower duty cycle, signal doesn’t have enough time to rise to peak before it has to come down.

1.4. What is the minimum PWM duty cycle for the motor to turn at no-load? Why does this limit exist? Discuss in your report.
Without load, minimum is at about 3% duty cycle (~1850/65536). This limit exists because a DC motor, in its equivalent circuit, is driven by an inductor. Without enough time for current to pass and voltage to build up, the motor will not turn.

2. Stepper Motor Control

2.1. Describe your apparatus and code in your report and include a screenshot of your C# program. Attach your MCU and C# code as appendices.

I assembled the stepper motor, h-bridge, and MSP430 board together using jumper wires and breadboard. The specifics of how they are connected is shown in section 2.2.

To control the motor, H-bridge takes 1 and 0 signals input its A-D channels from the board's P1.3-6. These 4 input channels accept 8 combinations of signals that can by sent in sequence; the sequence can be sent forward or backward to step clockwise or counterclockwise, respectively. This sequence is stored in 4 arrays (1 per channel) in the MCU code. How the sequence is sent (speed, direction, number of steps) is communicated to the board by UART.

In the C# app, UART can be started by selected the correct port using the combo box on top-left corner. You can use the track bar to set speed, with 10000 being fastest and 0 being slowest (arbitrary units). The speed will be displayed in textbox, which you can also type in numbers between 0 and 10000 to precisely set speed. Use checkbox to set direction. There are two buttons to step a single step, one in CW direction and the other in CCW direction.

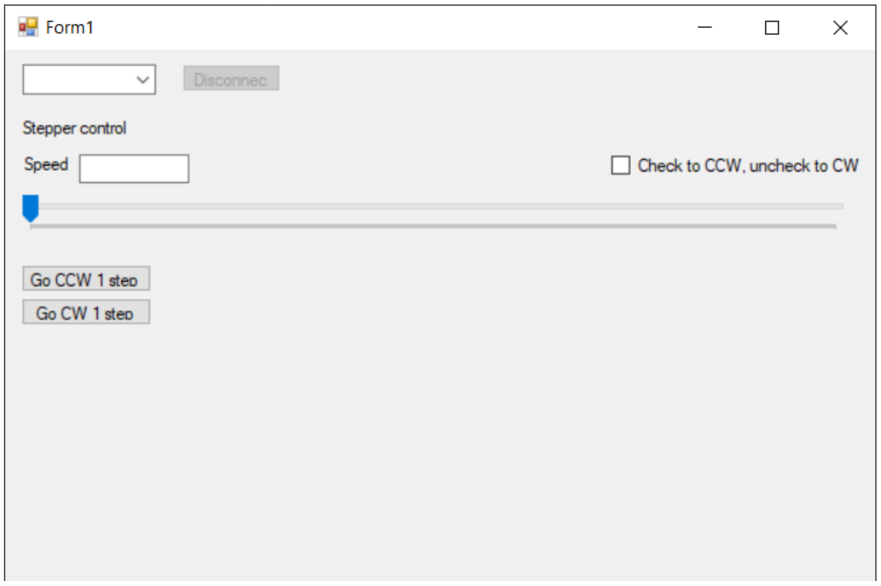


Fig 2.1: C# program window

The MCU and C# code are in Appendix 2.

2.2. Draw an electrical schematic diagram of the minimum components necessary to drive the stepper motor including the MCU, motor driver, power supply, and accessories.

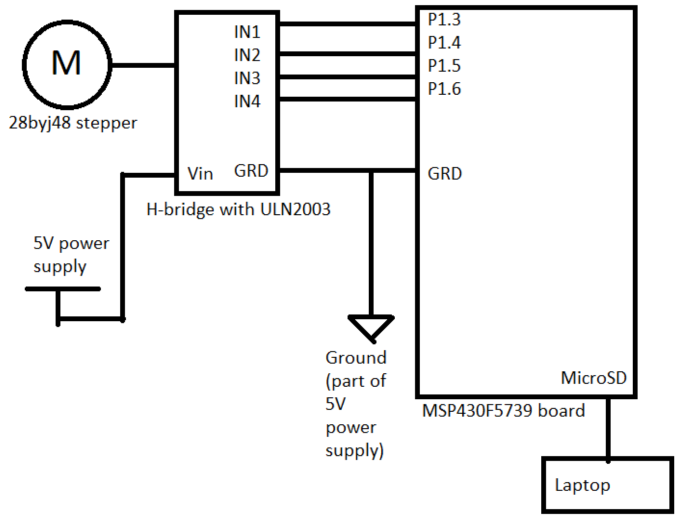


Fig 2.2: Electrical schematic diagram for exercise 2

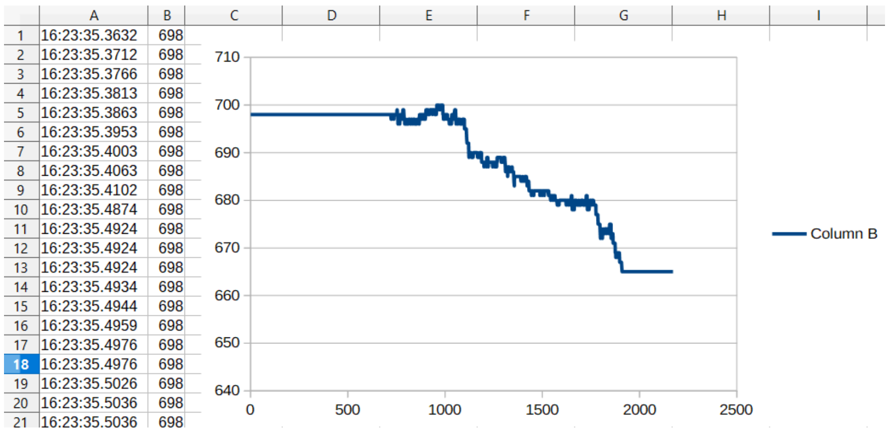
Wire connections were made with jumper wires and a breadboard.

2.3. What is the maximum speed of this stepper motor (in steps/s and rev/s)? Why does this limit exist? Discuss in your report.

Letting the motor go at its fastest and counting the rotations, it spins at around 18 rpm, so that is 0.3 rev/s. For any motor, there is a physical limit to how quickly current can travel to coils inside the motor and turn the motor shaft. This is even more present when there’s an h-bridge doing the intermediate control.

3. Encoder Reader

3.1. Manually turn the motor shaft and show the shaft position and rotational velocity data are correct. Save example data for your report.



	A	B
1	16:23:35.3632	698
2	16:23:35.3712	698
3	16:23:35.3766	698
4	16:23:35.3813	698
5	16:23:35.3863	698
6	16:23:35.3953	698
7	16:23:35.4003	698
8	16:23:35.4063	698
9	16:23:35.4102	698
10	16:23:35.4874	698
11	16:23:35.4924	698
12	16:23:35.4924	698
13	16:23:35.4924	698
14	16:23:35.4934	698
15	16:23:35.4944	698
16	16:23:35.4959	698
17	16:23:35.4976	698
18	16:23:35.4976	698
19	16:23:35.5026	698
20	16:23:35.5036	698
21	16:23:35.5036	698

Fig 3.1: Position data plotted

I used the C# code from exercise 4 to record data. This is from me turning the shaft in one direction. The trend is correct, as well as the general position, but I couldn't make it less noisy. I tried calculating the velocity but it was bad because of the noise from position data, so I didn't save it. A section of the data is in Appendix 3.

3.2. Describe your apparatus and code in your report and include a screenshot of your C# program. Attach your MCU and C# code as appendices.

I add flip flops to circuit from exercise 1, so that data could be read from encoder.

C# program sends 16-bit integer and direction as part of data package through UART, and receives position as read by encoder in return. It parses and plots that on the left plot, and calculate instantaneous velocity based on difference in position and difference in time when data was parsed. I chose to plot velocity every 15 data points received because plotting it every data point was too noisy. I used LiveCharts for plotting, due to my familiarity with it.

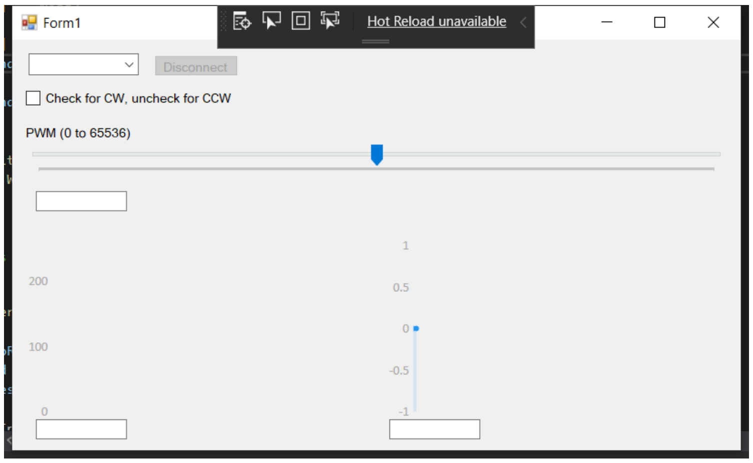


Fig 3.2: C# program window

MCU code has a timer interrupt that reads data from TA0 and TA1 (connected to encoder by flip flops). Since each encoder reads rotation is one direction (one positive, one negative), you can take the difference between encoder readings to get current position. Position and direction are sent back to C# program by UART. It can also receive a 16-bit number and direction for setting speed by changing PWM.

The MCU and C# code are in Appendix 3.

3.3. Draw an electrical schematic diagram of the minimum components necessary to obtain and process signals from the encoder, including the MCU, latches, power supply, and accessories.

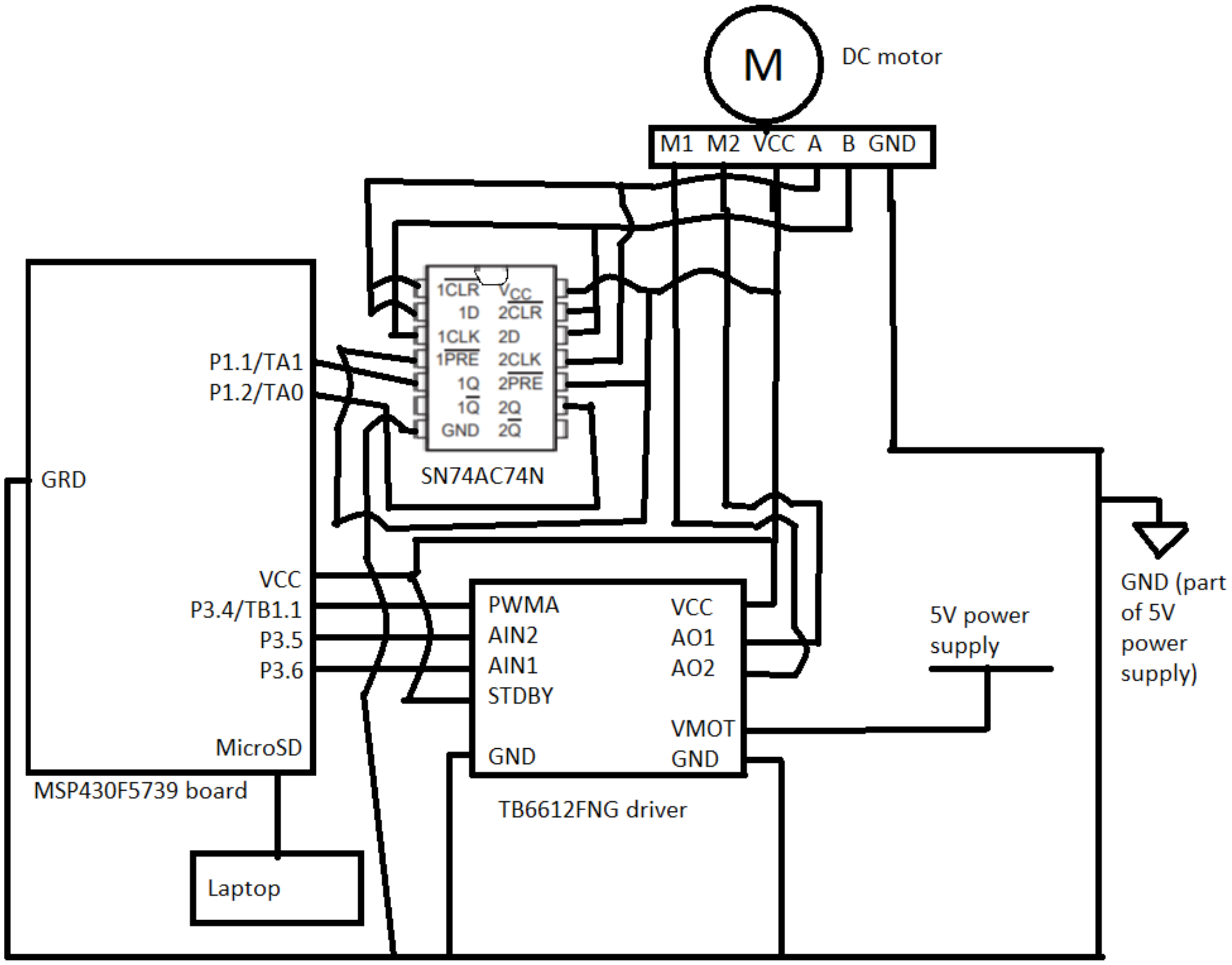


Fig 3.3: Electrical schematic diagram for exercise 3

4. Closing the Loop

Unfortunately my board is fried before I could finish this part. I honestly don't know how or why it happened. For the rest of the report, it'll be a mix of what I've done and my best guess on what I can do it to get the exercise working right.

4.1. Modify the C# and microprocessor code from Exercise 4 to apply a step input to the motor (e.g. PWM duty cycle changing from 0-25%, 0-50%, 0-100%, etc) while simultaneously acquiring the shaft position.

I modified exercise 3 C# program so that it can save position data. To get to percent duty cycle, I can input specific values into PWM (25% => 65536 * 0.25 = 16384).

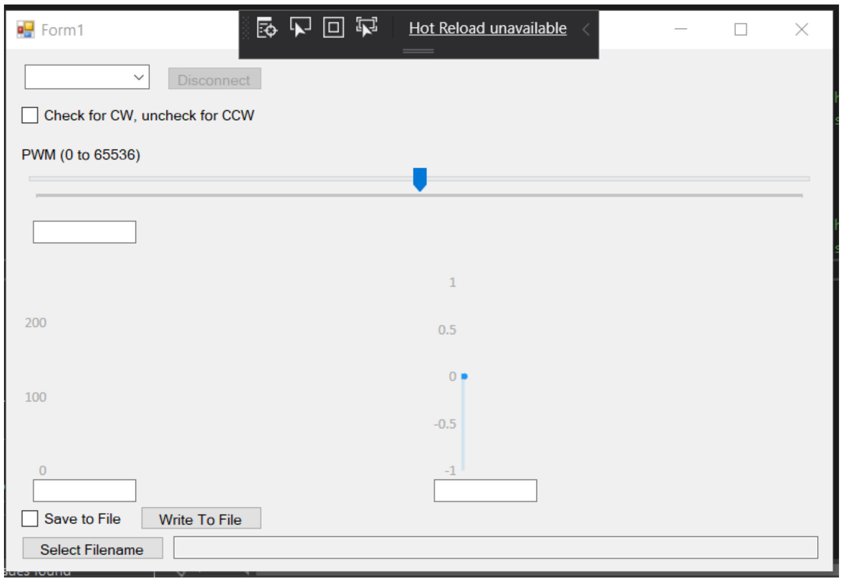


Fig 4.1: Modified exercise 3 C# program

4.2. Transfer function, block diagram, proportional control, system properties.

After saving data, I can do some noise reduction and get velocity over time. Plotting that, I can likely approximate that as a first order system.

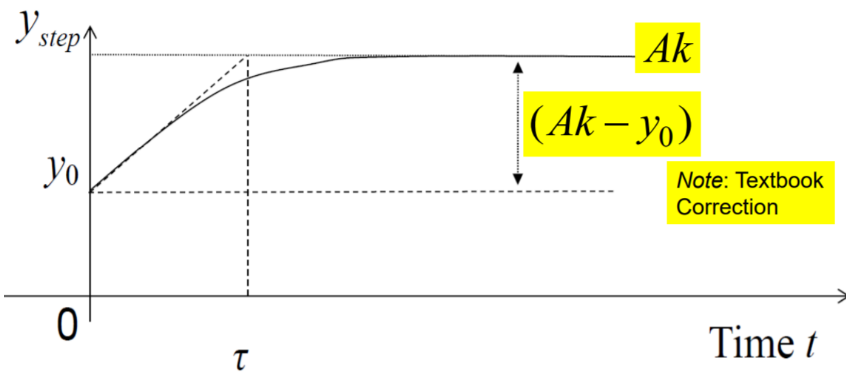


Fig 4.2: First order, step response (from MECH420 lectures)

The transfer function of a first order system is:

$$\frac{Y(s)}{U(s)} = H(s) = \frac{k}{\tau s + 1}$$

Where Y is the velocity, and U is the duty cycle. I can get value of T and k from the plot; T is the initial slope of the plot, k is gain, and A is the magnitude of input step (corresponds to duty cycle).

To control shaft position instead of velocity, I can say that Y is the position, and use the same transfer function times 1/s to represent converting from velocity to position (in reality, I'll test whether the modified transfer function works or not).

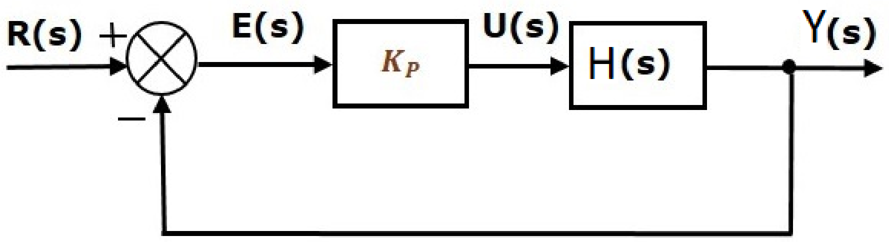


Fig 4.3: Block diagram of proportional controller (modified from tutorialspoint.com)

Kp is the conversion factor between position and duty cycle. It can be an arbitrary number initially, but over tests, it can be modified so that steady-state error is minimized. After Kp is selected, we will know the relationship of R(s) which is reference position, and Y(s) which is actual position. The relationship is:

$$\frac{Y(s)}{R(s)} = \frac{KpH(s)}{1 + KpH(s)}$$

Therefore, we can adjust our input reference position such that the actual position is what we want.

MCU code has to be modified so that duty cycle is continually adjusted as position of the shaft changes (I've put the code I have so far in Appendix 4). We can plot position against time and read rise time, overshoot, and settling time from the plot, similar to how we read values from the velocity plot.

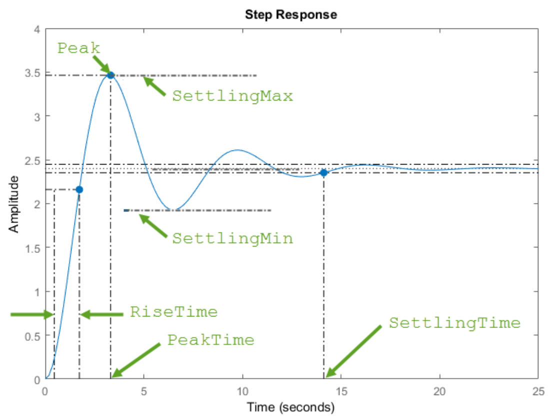


Fig 4.4: Second order, step response (from Mathworks.com)

5. 2-axis Control

5.1. Describe your approach, apparatus and code in your report and include a screenshot of your C# program and a picture of your piece of paper. Attach your MCU and C# code as appendices.

The circuit schematic is just a combination of exercise 3 and exercise 2's.

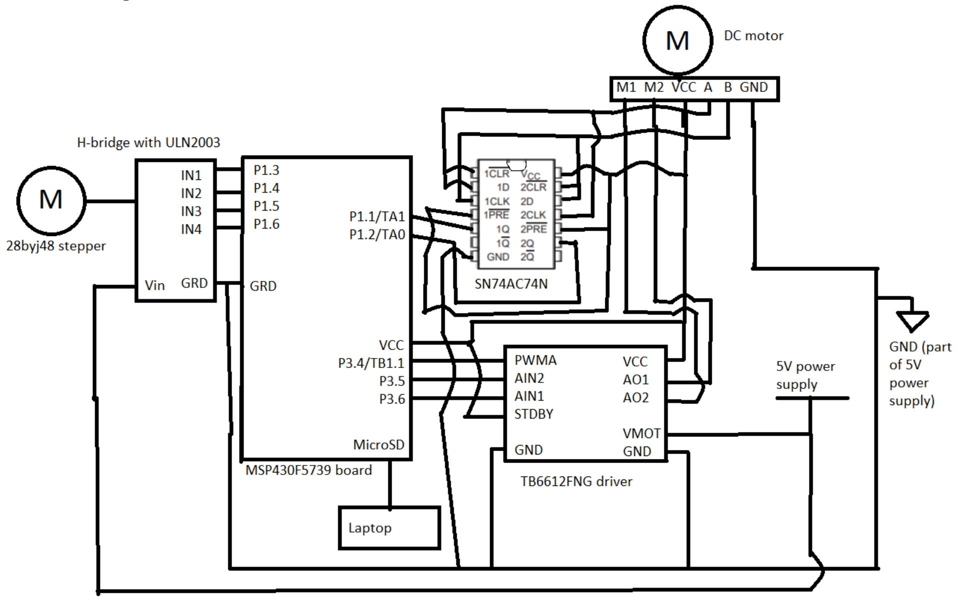


Fig 5.1: Electrical schematic diagram for exercise 5

My approach was to break this exercise down in steps:

1. Create C# program that sends x translation, y translation, and velocity as UART package in format that is usual to the labs (based on Lab 2). Since maximum travel length of the aluminum extrusion is about 19 cm, we can say that the range of x and y values are 0 to 38 (representing translation of -19 to 19 cm). Velocity is a percentage, so it'll be from 0 to 100. Each of these values can fit into 8-bit integers, which works well for us.
2. Modify exercise 2 MCU code such that it reads x translation and velocity and moves platform accordingly.
3. Modify exercise 4 MCU code such that it reads y translation and velocity and moves platform accordingly.
4. Put MCU code together, making sure that velocities are adjusted for distance travelled so that both platforms arrive at the same time.

This approach worked well, because I could work on part of exercise 5 without fully finishing exercise 4, meaning that I can alternate between exercises if I got stuck somewhere, and come back to where I got stuck later when I had some time to think. This approach has paid off somewhat, since it meant I had exercise 5 partly working and tested before my board got fried while working on exercise 4. I got it done up to step 2.

For step 1, the way my C# program worked is that the trackbars update the values in their respective textboxes. Send button reads values from the three textboxes, and sends them as a package by UART to the board.

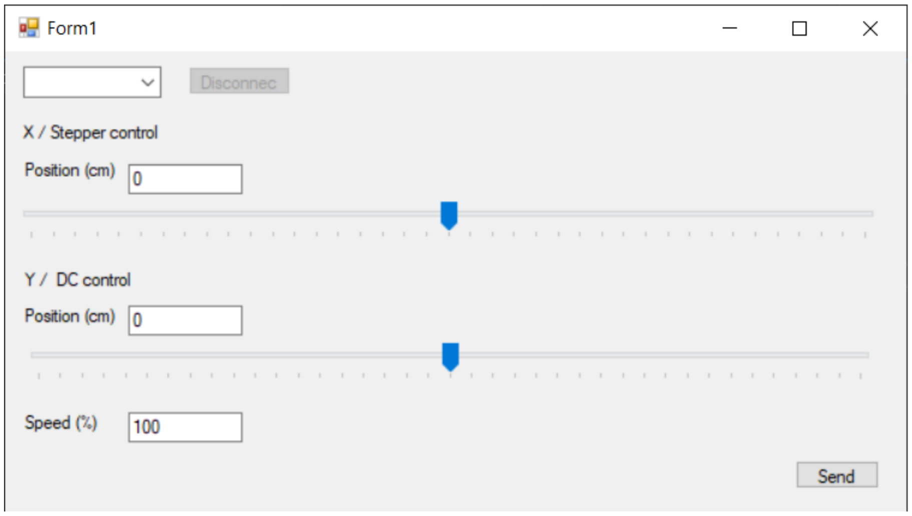


Fig 5.2: Exercise 5 C# program

For step 2, I experimentally determined the distance travelled per steps ratio, and added a counter to my exercise 2 MCU code so that when the counter hits the number of steps required to travel the commanded distance, the stepper motor stops stepping.

C# code and MCU code for stepper motor are attached in Appendix 5.

5.2. What are the advantages/disadvantages of using a stepper motor? What are the advantages/ disadvantages of using a DC motor? If you were designing a similar machine would you use stepper or DC motors and why? How would you implement a CW or CCW curve? What about a non-symmetrical curve? Discuss in your report.

	Advantage	Disadvantage
Stepper motor	• Precise steps. • High torque at low speed. • Easy to control.	• No position feedback. • Need h-bridge (although stepper motor and h-bridge are often paired together as one unit).
DC motor	• Faster. • No h-bridge. • Easy to approximate model.	• No position feedback without encoder. • With encoder, BOM count increases. • Imprecise.

In a similar 2-axis machine, I would use stepper motors since it can move the platforms more precisely, and at lower cost (no BOM count increase). Looking at this from a wisdom-of-the-crowd angle, most 3D printers of similar size also use stepper motors.

A difficult but precise way to implement curve is to adjust velocity as platforms are moving to end position. Any other curves can be drawn this way too. Majority of software changes are in done MCU code, modified so that non-linear motion can be done.

A simple but less precise way is to discretizing the curve, breaking it down into points that can be travelled to linearly. This way, the controls of the of two motors are kept the same, but the C# program that feeds the x and y translation distance has be modified so that it can discretize a curve before sending commands off to MCU.

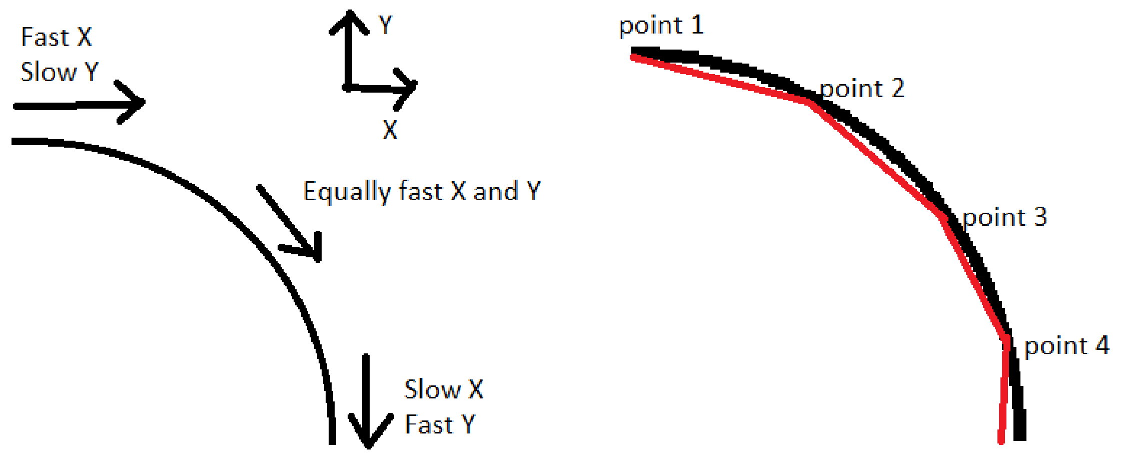


Fig 5.3: CW curve drawing with velocity control vs with discretization

Appendix 1

MCU code:

```
#include <msp430.h>

/**
 * main.c
 */

#define LEN 51
#define TRUE 1
#define FALSE 0

#define PJ_ALLON (BIT0 + BIT1 + BIT2 + BIT3) // bits 0-3
#define PJ_ALLOFF 0

volatile unsigned int read = 0;
volatile unsigned char readByte = 0;
volatile unsigned int write = 0;
volatile unsigned char buffer[51] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0 };
volatile unsigned int yeet = 0;

unsigned int plus1mod51(unsigned int i)
{
    // range: 0~51*2-1
    if (i + 1 < LEN)
    {
        return i + 1;
    }
    else
    {
        return (i + 1) - LEN;
    }
}

int readBuf()
{
    //read from buffer
    if (read == write)
    {
        return FALSE;
    }
    else
    {
        readByte = buffer[read];
        // doesn't delete old data from buffer, just wait for it to be overwritten later
        read = plus1mod51(read);
        return TRUE;
    }
}

int writeBuf(char n)
{
    if (read == plus1mod51(write))
    {
        // is full
        return FALSE;
    }
    else
    {
        buffer[write] = n;
        write = plus1mod51(write);
        return TRUE;
    }
}

int getBufCount()
{
    if (write >= read)
    {
        return write - read;
    }
    else
    {
        // read < write == write at beginning of array
        return (LEN - read) + write; // i think this is correct lol
    }
}

int parseInt16(char upperByte, char lowerByte, char escByte)
{
    if (escByte == 0x03)
    {
        upperByte = 0xFF;
        lowerByte = 0xFF;
    }
    else if (escByte == 0x02)
    {
        upperByte = 0xFF;
    }
    else if (escByte == 0x01)
    {
        lowerByte = 0xFF;
    }
    int ret = 0;
    ret |= (int) upperByte << 8;
    ret |= (int) lowerByte;
    return ret;
}

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer

    // Configure clocks
    CSCTL0 = 0xA500;                    // Write password to modify CS registers
    CSCTL1 = DCOFSEL0 + DCOFSEL1;           // DCO = 8 MHz
    CSCTL2 = SELM0 + SELM1 + SELA0 + SELA1 + SELS0 + SELS1; // MCLK = DCO, ACLK =
DCO, SMCLK = DCO

    // Configure ports for UCA0
    P2SEL0 &= ~(BIT0 + BIT1);
    P2SEL1 |= BIT0 + BIT1;

    // Configure UCA0
    UCA0CTLW0 = UCSSEL0;
    UCA0BRW = 52;
    UCA0MCTLW = 0x4900 + UCOS16 + UCBRF0;
    UCA0IE |= UCRXIE;

    // global interrupt enable
    _EINT();

    P3DIR |= BIT5 + BIT6;
    P3OUT |= BIT5;
    P3OUT &= ~BIT6;

    // Set Timer B
    P3DIR |= BIT4;
    P3SEL1 &= ~(BIT4);
    P3SEL0 |= BIT4;
    //P3DIR |= BIT4 + BIT5;
    //P3SEL1 &= ~(BIT4 + BIT5);
    //P3SEL0 |= BIT4 + BIT5;

    // Set PWM out
    TB1CCR0 = 65536;
    TB1CCTL1 = OUTMOD_7;
    TB1CCR1 = 30000;
    //TB1CCTL2 = OUTMOD_7;
    //TB1CCR2 = 250;
```

```
TB1CTL = TBSSEL_2 + MC_2 + TBCLR;

while (1)
{
    if (getBufCount() >= 5)
    {
        readByte = 0;
        while (!(readByte == 0xFF))
        { // find start byte and remove it
            readBuf();
        }
        char dir = 0;
        char byte1 = 0;
        char byte2 = 0;
        char escByte = 0;

        readBuf();
        dir = readByte;
        if (dir == 0) {
            // set ccw
            P3OUT |= BIT5;
            P3OUT &= ~BIT6;
        } else {
            // set cw
            P3OUT |= BIT6;
            P3OUT &= ~BIT5;
        }
        readBuf();
        byte1 = readByte;
        readBuf();
        byte2 = readByte;
        readBuf();
        escByte = readByte;
        int i = parseInt16(byte1, byte2, escByte);
        yeet = i; // for checking in debugger
        if (i > 65536)
        {
            TB1CCR1 = 65536;
        }
        else
        {
            TB1CCR1 = i;
        }

    }
}

return 0;
```

```
}

#pragma vector = USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
    unsigned char RxByte;
    RxByte = UCA0RXBUF;
    //write to buffer
    if (writeBuf(RxByte) == FALSE)
    {
        // send 0xFF 3 times as error msg if write fail
        while ((UCA0IFG & UCTXIFG) == 0)
            ;
        UCA0TXBUF = 0xFF;
        while ((UCA0IFG & UCTXIFG) == 0)
            ;
        UCA0TXBUF = 0xFF;
        while ((UCA0IFG & UCTXIFG) == 0)
            ;
        UCA0TXBUF = 0xFF;
    }

}
```

C# code, Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO.Ports;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace _1_cs
{
    public partial class Form1 : Form
    {
        private SerialPort serialPort = null;

        public Form1()
        {
            InitializeComponent();

            comboBox1.Items.AddRange(SerialPort.GetPortNames()); //set up combo box
            this.FormClosing += CloseSerial; // set up auto serial port close, in case user didn't click
disconnect

            disconnectBtn.Enabled = false; // turn disconnect button off since there's nothing to disconnect
now
        }

        private void CloseSerial(object sender, FormClosingEventArgs e)
        {
            if (serialPort != null && serialPort.IsOpen)
            {
                serialPort.Close(); // close port if not closed before app is closed
            }
        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            string portName = comboBox1.SelectedItem.ToString();

            if (serialPort == null) // if the previous port is not closed and null, then don't make a new
connection
            {
                serialPort = new SerialPort(portName, 9600, Parity.None, 8, StopBits.One);
                while (!serialPort.IsOpen)
                {
                    serialPort.Open();
                } // TA recommended something like this in case open doesn't actually open
                serialPort.ReadTimeout = 300;
                serialPort.DiscardInBuffer(); // remove data before reading is supposed to start
                serialPort.DataReceived += new SerialDataReceivedEventHandler(serialDataHandler);

                disconnectBtn.Enabled = true; // now that there's something to disconnect, turn disconnect
button on
            }
        }

        private void serialDataHandler(object sender, SerialDataReceivedEventArgs e)
        {
            // do nothing
        }

        private void button1_Click(object sender, EventArgs e)
        {
            serialPort.Close();
            serialPort = null; // close and null the serial port

            disconnectBtn.Enabled = false; // turn off disconnect button since there's nothing to disconnect
now
```

```
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            byte[] send = new byte[5];
            send[0] = 0xFF;
            if (checkBox1.Checked)
            {
                send[1] = 0x00; // cw
            }
            else
            {
                send[1] = 0x01; // ccw
            }
            int intValue = int.Parse(textBox1.Text);
            send[2] = (byte)(intValue >> 8);
            send[3] = (byte)intValue;
            send[4] = 0x00;
            if (send[2] == 0xFF)
            {
                send[4] = 0x02;
            }
            if (send[3] == 0xFF)
            {
                send[4] = 0x01;
                if (send[2] == 0xFF)
                {
                    send[4] = 0x03;
                }
            }
            Console.WriteLine(send.ToString());
            serialPort.Write(send, 0, 5);
        }
    }
}
```

C# code, Form1.Designer.cs

```
namespace _1_cs
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.trackBar1 = new System.Windows.Forms.TrackBar();
            this.label1 = new System.Windows.Forms.Label();
            this.checkBox1 = new System.Windows.Forms.CheckBox();
            this.comboBox1 = new System.Windows.Forms.ComboBox();
            this.disconnectBtn = new System.Windows.Forms.Button();
            this.textBox1 = new System.Windows.Forms.TextBox();
            ((System.ComponentModel.ISupportInitialize)(this.trackBar1)).BeginInit();
            this.SuspendLayout();
            //
            // trackBar1
            //
            this.trackBar1.Location = new System.Drawing.Point(12, 113);
            this.trackBar1.Maximum = 65536;
            this.trackBar1.Name = "trackBar1";
            this.trackBar1.Size = new System.Drawing.Size(776, 56);
            this.trackBar1.TabIndex = 0;
            this.trackBar1.Value = 32768;
            bool clicked = false;
            this.trackBar1.Scroll += (s, e) =>
            {
                if (clicked)
                    return;
            };

            this.trackBar1.MouseDown += (s,
                                e) =>
            {
```

```
            clicked = true;
        };
        trackBar1.MouseUp += (s,
                        e) =>
        {
            if (!clicked)
                return;

            clicked = false;
            textBox1.Text = trackBar1.Value.ToString();
        };
        //
        // label1
        //
        this.label1.AutoSize = true;
        this.label1.Location = new System.Drawing.Point(12, 93);
        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(123, 17);
        this.label1.TabIndex = 1;
        this.label1.Text = "PWM (0 to 65536)";
        //
        // checkBox1
        //
        this.checkBox1.AutoSize = true;
        this.checkBox1.Location = new System.Drawing.Point(15, 54);
        this.checkBox1.Name = "checkBox1";
        this.checkBox1.Size = new System.Drawing.Size(233, 21);
        this.checkBox1.TabIndex = 2;
        this.checkBox1.Text = "Check for CW, uncheck for CCW";
        this.checkBox1.UseVisualStyleBackColor = true;
        //
        // comboBox1
        //
        this.comboBox1.FormattingEnabled = true;
        this.comboBox1.Location = new System.Drawing.Point(18, 15);
        this.comboBox1.Name = "comboBox1";
        this.comboBox1.Size = new System.Drawing.Size(121, 24);
        this.comboBox1.TabIndex = 3;
        this.comboBox1.SelectedIndexChanged += new
System.EventHandler(this.comboBox1_SelectedIndexChanged);
        //
        // disconnectBtn
        //
        this.disconnectBtn.Location = new System.Drawing.Point(156, 17);
        this.disconnectBtn.Name = "disconnectBtn";
        this.disconnectBtn.Size = new System.Drawing.Size(92, 23);
        this.disconnectBtn.TabIndex = 4;
        this.disconnectBtn.Text = "Disconnect";
```

```
            this.disconnectBtn.UseVisualStyleBackColor = true;
            this.disconnectBtn.Click += new System.EventHandler(this.button1_Click);
            //
            // textBox1
            //
            this.textBox1.Location = new System.Drawing.Point(26, 166);
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(100, 22);
            this.textBox1.TabIndex = 5;
            this.textBox1.TextChanged += new System.EventHandler(this.textBox1_TextChanged);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(800, 450);
            this.Controls.Add(this.textBox1);
            this.Controls.Add(this.disconnectBtn);
            this.Controls.Add(this.comboBox1);
            this.Controls.Add(this.checkBox1);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.trackBar1);
            this.Name = "Form1";
            this.Text = "Form1";
            ((System.ComponentModel.ISupportInitialize)(this.trackBar1)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.TrackBar trackBar1;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.CheckBox checkBox1;
        private System.Windows.Forms.ComboBox comboBox1;
        private System.Windows.Forms.Button disconnectBtn;
        private System.Windows.Forms.TextBox textBox1;
    }
}
```

Appendix 2

MCU code:

```
#include <msp430.h>

/**
 * main.c
 */

#define LEN 51
#define TRUE 1
#define FALSE 0

#define PJ_ALLON (BIT0 + BIT1 + BIT2 + BIT3) // bits 0-3
#define PJ_ALLOFF 0

volatile unsigned int read = 0;
volatile unsigned char readByte = 0;
volatile unsigned int write = 0;
volatile unsigned char buffer[51] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0 };
volatile unsigned int yeet = 0;

unsigned int plus1mod51(unsigned int i)
{
    // range: 0~51*2-1
    if (i + 1 < LEN)
    {
        return i + 1;
    }
    else
    {
        return (i + 1) - LEN;
    }
}

int readBuf()
{
    //read from buffer
    if (read == write)
    {
        return FALSE;
    }
    else
    {
        readByte = buffer[read];
        // doesn't delete old data from buffer, just wait for it to be overwritten later
        read = plus1mod51(read);
        return TRUE;
    }
}

int writeBuf(char n)
{
    if (read == plus1mod51(write))
    {
        // is full
        return FALSE;
    }
    else
    {
        buffer[write] = n;
        write = plus1mod51(write);
        return TRUE;
    }
}

int getBufCount()
{
    if (write >= read)
    {
        return write - read;
    }
    else
    {
        // read < write == write at beginning of array
        return (LEN - read) + write; // i think this is correct lol
    }
}

int parseInt16(char upperByte, char lowerByte, char escByte)
{
    if (escByte == 0x03)
    {
        upperByte = 0xFF;
        lowerByte = 0xFF;
    }
    else if (escByte == 0x02)
    {
        upperByte = 0xFF;
    }
    else if (escByte == 0x01)
    {
        lowerByte = 0xFF;
    }
    int ret = 0;
    ret |= (int) upperByte << 8;
    ret |= (int) lowerByte;
    return ret;
}

void stepStepper(int p3, int p4, int p5, int p6)
{
    if (p3 == TRUE)
    {
        P1OUT |= BIT3;
    }
    else
    {
        P1OUT &= ~BIT3;
    }
    if (p4 == TRUE)
    {
        P1OUT |= BIT4;
    }
    else
    {
        P1OUT &= ~BIT4;
    }
    if (p5 == TRUE)
    {
        P1OUT |= BIT5;
    }
    else
    {
        P1OUT &= ~BIT5;
    }
    if (p6 == TRUE)
    {
        P1OUT |= BIT6;
    }
    else
    {
        P1OUT &= ~BIT6;
    }
}

unsigned int plus1loop8(state) {
    if (state == 7) {
        return 0;
    } else {
        return state + 1;
    }
}

unsigned int minus1loop8(state) {
    if (state == 0) {
        return 7;
    } else {
        return state - 1;
    }
}

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer

    // Configure clocks
    CSCTL0 = 0xA500;// Write password to modify CS registers
    CSCTL1 = DCOFSEL0 + DCOFSEL1;// DCO = 8 MHz
    CSCTL2 = SELM0 + SELM1 + SELA0 + SELA1 + SELS0 + SELS1;// MCLK = DCO, ACLK =
DCO, SMCLK = DCO

    // Configure ports for UCA0
    P2SEL0 &= ~(BIT0 + BIT1);
    P2SEL1 |= BIT0 + BIT1;

    // Configure UCA0
    UCA0CTLW0 = UCSSEL0;
    UCA0BRW = 52;
    UCA0MCTLW = 0x4900 + UCOS16 + UCBRF0;
    UCA0IE |= UCRXIE;

    // global interrupt enable
    _EINT();

    // Stepper set up
    P1DIR |= BIT3 + BIT4 + BIT5 + BIT6;
    int count = 0;
    int countUpTo = 5000;
    unsigned int state = 0;
    int direction = TRUE;

    int p3[8] =
    {   FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE};
    int p4[8] =
    {   FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE};
    int p5[8] =
    {   FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, FALSE, FALSE};
    int p6[8] =
    {   TRUE, TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE};
    stepStepper(p3[state], p4[state], p5[state], p6[state]);

    while (1)
    {
        yeet = countUpTo; // for checking value in debug
        // do stepping
        if (countUpTo != 0) {
            if (count >= countUpTo)
            {
                if (direction == TRUE) // true = cw
                {
                    state = plus1loop8(state);
                }
                else
                {
                    state = minus1loop8(state);
                }
                stepStepper(p3[state], p4[state], p5[state], p6[state]);
                count = 0;
            }
            else
            {
                count = count + 1;
            }
        }

        // read uart buffer
        if (getBufCount() >= 5)
        {
            readByte = 0;
            while (!(readByte == 0xFF))
            { // find start byte and remove it
                readBuf();
            }
            char dir = 0;
            char byte1 = 0;
            char byte2 = 0;
            char escByte = 0;

            readBuf();
            dir = readByte;
            readBuf();
            byte1 = readByte;
            readBuf();
            byte2 = readByte;
            readBuf();
            escByte = readByte;
```

```
            int i = parseInt16(byte1, byte2, escByte);

            if (dir == 2)
            {
                // set cw
                direction = TRUE;
                countUpTo = i;
            }
            else if (dir == 3)
            {
                // set ccw
                direction = FALSE;
                countUpTo = i;
            } else if (dir == 4) { // cw step
                state = plus1loop8(state);
                stepStepper(p3[state], p4[state], p5[state], p6[state]);
                countUpTo = 0;
            } else if (dir == 5) { // ccw
                state = minus1loop8(state);
                stepStepper(p3[state], p4[state], p5[state], p6[state]);
                countUpTo = 0;
            }
            count = 0;

        }
    }

    return 0;
}

#pragma vector = USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
    unsigned char RxByte;
    RxByte = UCA0RXBUF;
    //write to buffer
    if (writeBuf(RxByte) == FALSE)
    {
        // send 0xFF 3 times as error msg if write fail
        while ((UCA0IFG & UCTXIFG) == 0)
        ;
        UCA0TXBUF = 0xFF;
        while ((UCA0IFG & UCTXIFG) == 0)
        ;
        UCA0TXBUF = 0xFF;
        while ((UCA0IFG & UCTXIFG) == 0)
        ;
        UCA0TXBUF = 0xFF;
    }

}
```

C# code, Form1.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO.Ports;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        private SerialPort serialPort = null;
        bool clicked = false;

        public Form1()
        {
            InitializeComponent();

            comboBox1.Items.AddRange(SerialPort.GetPortNames()); //set up combo box

            this.FormClosing += CloseSerial; // set up auto serial port close, in case user didn't click
disconnect

            disconnectBtn.Enabled = false; // turn disconnect button off since there's nothing to disconnect
now

            this.trackBar1.Scroll += (s, e) =>
            {
                if (clicked)
                    return;
            };

            this.trackBar1.MouseDown += (s,
                                    e) =>
            {
                clicked = true;
            };
            trackBar1.MouseUp += (s,
                                    e) =>
        {
            if (!clicked)
                return;

            clicked = false;
            textBox1.Text = trackBar1.Value.ToString();
        };
    }


    private void CloseSerial(object sender, FormClosingEventArgs e)
    {
        if (serialPort != null && serialPort.IsOpen)
        {
            serialPort.Close(); // close port if not closed before app is closed
        }
    }

    private void serialDataHandler(object sender, SerialDataReceivedEventArgs e)
    {
        // do nothing
    }

    int maxSpeed = 18; // experimentally determined

    private void setSpeed(byte command, int speed)
    {
        byte[] send = new byte[5];
        send[0] = 0xFF;
        send[1] = command;
        int intValue = speed;
        send[2] = (byte)(intValue >> 8);
        send[3] = (byte)intValue;
        send[4] = 0x00;
        if (send[2] == 0xFF)
        {
            send[4] = 0x02;
        }
        if (send[3] == 0xFF)
        {
            send[4] = 0x01;
            if (send[2] == 0xFF)
            {
                send[4] = 0x03;
            }
        }
        Console.WriteLine(send.ToString());
        serialPort.Write(send, 0, 5);
        }

        private void button1_Click(object sender, EventArgs e)
        {
            // cw
            setSpeed(0x02, 0);
            setSpeed(0x04, 100); // ccw
        }

        private void button2_Click(object sender, EventArgs e)
        {
            // ccw
            setSpeed(0x02, 0);
            setSpeed(0x05, 100); // cw
        }

        private void disconnectBtn_Click(object sender, EventArgs e)
        {
            serialPort.Close();
            serialPort = null; // close and null the serial port

            disconnectBtn.Enabled = false; // turn off disconnect button since there's nothing to disconnect
now

        }

        private void comboBox1_SelectedIndexChanged_1(object sender, EventArgs e)
        {
            string portName = comboBox1.SelectedItem.ToString();

            if (serialPort == null) // if the previous port is not closed and null, then don't make a new
connection
            {
                serialPort = new SerialPort(portName, 9600, Parity.None, 8, StopBits.One);
                while (!serialPort.IsOpen)
                {
                    serialPort.Open();
                } // TA recommended something like this in case open doesn't actually open
                serialPort.ReadTimeout = 300;
                serialPort.DiscardInBuffer(); // remove data before reading is supposed to start
                serialPort.DataReceived += new SerialDataReceivedEventHandler(serialDataHandler);

                disconnectBtn.Enabled = true; // now that there's something to disconnect, turn disconnect
button on
            }
        }

        private void textBox1_TextChanged_1(object sender, EventArgs e)
        {
            byte command;
            if (checkBox1.Checked)
            {
                command = 0x02; // ccw
            }
            else
            {
                command = 0x03; // cw
            }
            int intValue = int.Parse(textBox1.Text);
            int speed = 10000 - intValue + maxSpeed;
            setSpeed(command, speed);
        }
    }
}
```

C# code, Form1.Designer.cs:

```
namespace WindowsFormsApp1
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
```

```
{
    this.label1 = new System.Windows.Forms.Label();
    this.trackBar1 = new System.Windows.Forms.TrackBar();
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.label2 = new System.Windows.Forms.Label();
    this.button1 = new System.Windows.Forms.Button();
    this.button2 = new System.Windows.Forms.Button();
    this.checkBox1 = new System.Windows.Forms.CheckBox();
    this.comboBox1 = new System.Windows.Forms.ComboBox();
    this.disconnectBtn = new System.Windows.Forms.Button();
    ((System.ComponentModel.ISupportInitialize)(this.trackBar1)).BeginInit();
    this.SuspendLayout();
    //
    // label1
    //
    this.label1.AutoSize = true;
    this.label1.Location = new System.Drawing.Point(12, 58);
    this.label1.Name = "label1";
    this.label1.Size = new System.Drawing.Size(105, 17);
    this.label1.TabIndex = 0;
    this.label1.Text = "Stepper control";
    //
    // trackBar1
    //
    this.trackBar1.Location = new System.Drawing.Point(5, 120);
    this.trackBar1.Maximum = 10000;
    this.trackBar1.Name = "trackBar1";
    this.trackBar1.Size = new System.Drawing.Size(776, 56);
    this.trackBar1.TabIndex = 1;
    //
    // textBox1
    //
    this.textBox1.Location = new System.Drawing.Point(68, 89);
    this.textBox1.Name = "textBox1";
    this.textBox1.Size = new System.Drawing.Size(100, 22);
    this.textBox1.TabIndex = 2;
    this.textBox1.TextChanged += new System.EventHandler(this.textBox1_TextChanged_1);
    //
    // label2
    //
    this.label2.AutoSize = true;
    this.label2.Location = new System.Drawing.Point(13, 89);
    this.label2.Name = "label2";
    this.label2.Size = new System.Drawing.Size(49, 17);
    this.label2.TabIndex = 3;
    this.label2.Text = "Speed";
    //
    // button1
```

```
            //
            this.button1.Location = new System.Drawing.Point(15, 182);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(120, 23);
            this.button1.TabIndex = 4;
            this.button1.Text = "Go CCW 1 step";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // button2
            //
            this.button2.Location = new System.Drawing.Point(15, 211);
            this.button2.Name = "button2";
            this.button2.Size = new System.Drawing.Size(120, 23);
            this.button2.TabIndex = 5;
            this.button2.Text = "Go CW 1 step";
            this.button2.UseVisualStyleBackColor = true;
            this.button2.Click += new System.EventHandler(this.button2_Click);
            //
            // checkBox1
            //
            this.checkBox1.AutoSize = true;
            this.checkBox1.Location = new System.Drawing.Point(558, 88);
            this.checkBox1.Name = "checkBox1";
            this.checkBox1.Size = new System.Drawing.Size(223, 21);
            this.checkBox1.TabIndex = 6;
            this.checkBox1.Text = "Check to CCW, uncheck to CW";
            this.checkBox1.UseVisualStyleBackColor = true;
            //
            // comboBox1
            //
            this.comboBox1.FormattingEnabled = true;
            this.comboBox1.Location = new System.Drawing.Point(16, 12);
            this.comboBox1.Name = "comboBox1";
            this.comboBox1.Size = new System.Drawing.Size(121, 24);
            this.comboBox1.TabIndex = 7;
            this.comboBox1.SelectedIndexChanged += new
System.EventHandler(this.comboBox1_SelectedIndexChanged_1);
            //
            // disconnectBtn
            //
            this.disconnectBtn.Location = new System.Drawing.Point(163, 12);
            this.disconnectBtn.Name = "disconnectBtn";
            this.disconnectBtn.Size = new System.Drawing.Size(90, 23);
            this.disconnectBtn.TabIndex = 8;
            this.disconnectBtn.Text = "Disconnect";
            this.disconnectBtn.UseVisualStyleBackColor = true;
            this.disconnectBtn.Click += new System.EventHandler(this.disconnectBtn_Click);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(800, 450);
            this.Controls.Add(this.disconnectBtn);
            this.Controls.Add(this.comboBox1);
            this.Controls.Add(this.checkBox1);
            this.Controls.Add(this.button2);
            this.Controls.Add(this.button1);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.textBox1);
            this.Controls.Add(this.trackBar1);
            this.Controls.Add(this.label1);
            this.Name = "Form1";
            this.Text = "Form1";
            ((System.ComponentModel.ISupportInitialize)(this.trackBar1)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TrackBar trackBar1;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.CheckBox checkBox1;
        private System.Windows.Forms.ComboBox comboBox1;
        private System.Windows.Forms.Button disconnectBtn;
    }
}
```

Appendix 3:

MCU code:

```
#include <msp430.h>

/**
 * main.c
 */

#define LEN 51
#define TRUE 1
#define FALSE 0

#define PJ_ALLON (BIT0 + BIT1 + BIT2 + BIT3) // bits 0-3
#define PJ_ALLOFF 0

volatile unsigned int read = 0;
volatile unsigned char readByte = 0;
volatile unsigned int write = 0;
volatile unsigned char buffer[51] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0 };
volatile unsigned int yeet = 0;

volatile unsigned int readEncoder = FALSE;

unsigned int plus1mod51(unsigned int i)
{
    // range: 0~51*2-1
    if (i + 1 < LEN)
    {
        return i + 1;
    }
    else
    {
        return (i + 1) - LEN;
    }
}

int readBuf()
{
    //read from buffer
    if (read == write)
    {
        return FALSE;
    }
    else
```

```
   {
      readByte = buffer[read];
      // doesn't delete old data from buffer, just wait for it to be overwritten later
      read = plus1mod51(read);
      return TRUE;
   }
}

int writeBuf(char n)
{
   if (read == plus1mod51(write))
   {
      // is full
      return FALSE;
   }
   else
   {
      buffer[write] = n;
      write = plus1mod51(write);
      return TRUE;
   }
}

int getBufCount()
{
   if (write >= read)
   {
      return write - read;
   }
   else
   {
      // read < write == write at beginning of array
      return (LEN - read) + write; // i think this is correct lol
   }
}

int parseInt16(char upperByte, char lowerByte, char escByte)
{
   if (escByte == 0x03)
   {
      upperByte = 0xFF;
      lowerByte = 0xFF;
   }
   else if (escByte == 0x02)
   {
      upperByte = 0xFF;
   }
   else if (escByte == 0x01)
```

```
    {
        lowerByte = 0xFF;
    }
    int ret = 0;
    ret |= (int) upperByte << 8;
    ret |= (int) lowerByte;
    return ret;
}

void readBufAndSetPWM()
{
    readByte = 0;
    while (!(readByte == 0xFF))
    { // find start byte and remove it
        readBuf();
    }
    char dir = 0;
    char byte1 = 0;
    char byte2 = 0;
    char escByte = 0;

    readBuf();
    dir = readByte;
    if (dir == 0)
    {
        // set ccw
        P3OUT |= BIT5;
        P3OUT &= ~BIT6;
    }
    else
    {
        // set cw
        P3OUT |= BIT6;
        P3OUT &= ~BIT5;
    }
    readBuf();
    byte1 = readByte;
    readBuf();
    byte2 = readByte;
    readBuf();
    escByte = readByte;
    int i = parseInt16(byte1, byte2, escByte);
    yeet = i; // for checking in debugger
    if (i > 65536)
    {
        TB1CCR1 = 65536;
    }
    else
    {
        TB1CCR1 = i;
    }

}

void readEncoderFn()
{
    unsigned int encoder0 = TA0R;
    unsigned int encoder1 = TA1R;
    unsigned int pos = 0;
    int dir = 2;
    if (encoder0 >= encoder1)
    {
        pos = encoder0 - encoder1;
        dir = 0;
    }
    else
    {
        pos = encoder1 - encoder0;
        dir = 1;
    }
    int lowerByte = pos;
    int upperByte = pos >> 8;
    int escByte = 0;
    if (lowerByte == 0xFF && upperByte == 0xFF)
    {
        escByte = 0x03;
        lowerByte = 0x00;
        upperByte = 0x00;
    }
    else if (lowerByte == 0xFF)
    {
        escByte = 0x01;
        lowerByte = 0x00;
    }
    else if (upperByte == 0xFF)
    {
        escByte = 0x02;
        upperByte = 0x00;
    }

    while ((UCA0IFG & UCTXIFG) == 0)
        ;
    UCA0TXBUF = 0xFF;
    while ((UCA0IFG & UCTXIFG) == 0)
        ;
    UCA0TXBUF = dir;
    while ((UCA0IFG & UCTXIFG) == 0)
        ;
    UCA0TXBUF = upperByte;
    while ((UCA0IFG & UCTXIFG) == 0)
        ;
    UCA0TXBUF = lowerByte;
    while ((UCA0IFG & UCTXIFG) == 0)
        ;
    UCA0TXBUF = escByte;
}

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer

    // CLOCK
    // Configure clocks
    CSCTL0 = 0xA500;                    // Write password to modify CS registers
    CSCTL1 = DCOFSEL0 + DCOFSEL1;           // DCO = 8 MHz
    CSCTL2 = SELM0 + SELM1 + SELA0 + SELA1 + SELS0 + SELS1; // MCLK = DCO, ACLK =
DCO, SMCLK = DCO

    // UART
    // Configure ports for UCA0
    P2SEL0 &= ~(BIT0 + BIT1);
    P2SEL1 |= BIT0 + BIT1;
    // Configure UCA0
    UCA0CTLW0 = UCSSEL0;
    UCA0BRW = 52;
    UCA0MCTLW = 0x4900 + UCOS16 + UCBRF0;
    UCA0IE |= UCRXIE;
    // global interrupt enable
    _EINT();

    // PWM
    P3DIR |= BIT5 + BIT6;
    P3OUT |= BIT5;
    P3OUT &= ~BIT6;
    // Set Timer B
    P3DIR |= BIT4;
    P3SEL1 &= ~(BIT4);
    P3SEL0 |= BIT4;
    // Set PWM out
    TB1CCR0 = 65536;
    TB1CCTL1 = OUTMOD_7;
    TB1CCR1 = 30000;
    //TB1CCR1 = 0;
    TB1CTL = TBSSEL_2 + MC_2 + TBCLR;
    // READ ENCODER
    // Set Timer A
    P1DIR &= ~(BIT1 + BIT2);
    P1SEL1 &= ~(BIT1 + BIT2);
    P1SEL0 |= (BIT1 + BIT2);
    // Set Timer A for capture
    TA1CTL |= TASSEL_0 + MC_2 + TACLR;
    TA0CTL |= TASSEL_0 + MC_2 + TACLR;

    // INTERRUPT
    TB2CTL |= TBSSEL_1 + MC_1;
    TB2CCR0 = 300000;     // arbitrary interval
    TB2CCTL0 = CCIE;    // enable interrupt

    while (1)
    {
        if (getBufCount() >= 5)
        {
            readBufAndSetPWM();
        }
        if (readEncoder == TRUE)
        {
            readEncoderFn();
            readEncoder = FALSE;
        }
    }

    return 0;
}

#pragma vector = USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
    unsigned char RxByte;
    RxByte = UCA0RXBUF;
    //write to buffer
    if (writeBuf(RxByte) == FALSE)
    {
        // send 0xFF 3 times as error msg if write fail
        while ((UCA0IFG & UCTXIFG) == 0)
            ;
        UCA0TXBUF = 0xFF;
        while ((UCA0IFG & UCTXIFG) == 0)
            ;
        UCA0TXBUF = 0xFF;
        while ((UCA0IFG & UCTXIFG) == 0)
            ;
        UCA0TXBUF = 0xFF;
    }
}

volatile unsigned int a = 0;

#pragma vector = TIMER2_B0_VECTOR
__interrupt void TIMER2_B0_ISR(void)
{
    readEncoder = TRUE;
}
```

C# code, Form1.cs:

```
using LiveCharts;
using LiveCharts.Configurations;
using LiveCharts.WinForms;
using LiveCharts.Wpf;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO.Ports;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace _1_cs
{
    public partial class Form1 : Form
    {
        private SerialPort serialPort = null;
        bool clicked = true;
        Queue<int> serialIn = new Queue<int>();

        public ChartValues<MeasureModel> ChartValues_pos { get; set; }
        public ChartValues<MeasureModel> ChartValues_vel { get; set; }

        private int numOfPointsOnChart = 100;

        public Form1()
        {
            InitializeComponent();

            // serial
            #region
            comboBox1.Items.AddRange(SerialPort.GetPortNames()); //set up combo box
        this.FormClosing += CloseSerial; // set up auto serial port close, in case user didn't click
disconnect
        disconnectBtn.Enabled = false; // turn disconnect button off since there's nothing to disconnect
now
        #endregion

        // dc motor pwm
        #region
        this.trackBar1.Scroll += (s, e) =>
        {
           if (clicked)
              return;
        };

        this.trackBar1.MouseDown += (s,
                        e) =>
        {
           clicked = true;
        };
        trackBar1.MouseUp += (s,
                        e) =>
        {
           if (!clicked)
              return;

           clicked = false;
           textBox1.Text = trackBar1.Value.ToString();
        };
        #endregion

        // livecharts
        #region

        //To handle live data easily, in this case we built a specialized type
        //the MeasureModel class, it only contains 2 properties
        //DateTime and Value
        //We need to configure LiveCharts to handle MeasureModel class
        //The next code configures MEasureModel  globally, this means
        //that livecharts learns to plot MeasureModel and will use this config every time
        //a ChartValues instance uses this type.
        //this code ideally should only run once, when application starts is reccomended.
        //you can configure series in many ways, learn more at
http://lvcharts.net/App/examples/v1/wpf/Types%20and%20Configuration

        var mapper = Mappers.Xy<MeasureModel>()
           .X(model => model.index)   //use DateTime.Ticks as X
           .Y(model => model.Value);         //use the value property as Y
```

```
//lets save the mapper globally.
Charting.For<MeasureModel>(mapper);

//the ChartValues property will store our values array
ChartValues_pos = new ChartValues<MeasureModel>();
posChart.Series = new SeriesCollection
{
   new LineSeries
   {
      Values = ChartValues_pos,
      PointGeometrySize = 5,
      StrokeThickness = 4
   }
};
posChart.AxisY.Clear();
posChart.AxisY.Add(new Axis
{
   MinValue = 0,
   MaxValue = 255
});
posChart.AxisX.Add(new Axis
{
   DisableAnimations = true,
});
SetAxisLimits_posChart(0);

//the ChartValues property will store our values array
ChartValues_vel = new ChartValues<MeasureModel>();
velChart.Series = new SeriesCollection
{
   new LineSeries
   {
      Values = ChartValues_vel,
      PointGeometrySize = 5,
      StrokeThickness = 4
   }
};
velChart.AxisY.Clear();
velChart.AxisY.Add(new Axis
{
   MinValue = -1,
   MaxValue = 1
});
velChart.AxisX.Add(new Axis
{
   DisableAnimations = true,
});
for (int i = 0; i < numOfPointsOnChart; i++)
```

```
            {
                ChartValues_vel.Add(new MeasureModel
                {
                    index = i,
                    Value = 0
                });
            }
            SetAxisLimits_velChart(0);

            #endregion
        }

        // serial stuff
        #region
        private void CloseSerial(object sender, FormClosingEventArgs e)
        {
            if (serialPort != null && serialPort.IsOpen)
            {
                serialPort.Close(); // close port if not closed before app is closed
            }
        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            string portName = comboBox1.SelectedItem.ToString();

            if (serialPort == null) // if the previous port is not closed and null, then don't make a new
connection
            {
                serialPort = new SerialPort(portName, 9600, Parity.None, 8, StopBits.One);
                while (!serialPort.IsOpen)
                {
                    serialPort.Open();
                } // TA recommended something like this in case open doesn't actually open
                serialPort.ReadTimeout = 300;
                serialPort.DiscardInBuffer(); // remove data before reading is supposed to start
                serialPort.DataReceived += new SerialDataReceivedEventHandler(serialDataHandler);

                disconnectBtn.Enabled = true; // now that there's something to disconnect, turn disconnect
button on
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            serialPort.Close();
            serialPort = null; // close and null the serial port
            disconnectBtn.Enabled = false; // turn off disconnect button since there's nothing to disconnect
now

        }

        #endregion

        // dc motor pwm stuff
        #region

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            byte[] send = new byte[5];
            send[0] = 0xFF;
            if (checkBox1.Checked)
            {
                send[1] = 0x00; // cw
            }
            else
            {
                send[1] = 0x01; // ccw
            }
            int intValue = int.Parse(textBox1.Text);
            send[2] = (byte)(intValue >> 8);
            send[3] = (byte)intValue;
            send[4] = 0x00;
            if (send[2] == 0xFF)
            {
                send[4] = 0x02;
            }
            if (send[3] == 0xFF)
            {
                send[4] = 0x01;
                if (send[2] == 0xFF)
                {
                    send[4] = 0x03;
                }
            }
            Console.WriteLine(send.ToString());
            serialPort.Write(send, 0, 5);
        }
        #endregion

        // dc motor pos and speed and livecharts
        #region
        private void serialDataHandler(object sender, SerialDataReceivedEventArgs e)
        {
            int bytesToRead;
            bytesToRead = serialPort.BytesToRead;
            while (bytesToRead != 0) // keep reading until there's nothing left to read
            {
                serialIn.Enqueue(serialPort.ReadByte()); // enqueue byte that's read into queue
                bytesToRead = serialPort.BytesToRead; // reset bytesToRead
            }
            if (serialIn.Count >= 5)
            {
                if (InvokeRequired)
                {
                    Invoke((MethodInvoker)delegate { getNewData(); });
                }
            }
        }

        DateTime prevTime = DateTime.Now;
        double prevPos = 0;
        int velcounter = 0;
        double vel = 0;
        private void getNewData()
        {
            while (serialIn.Peek() != 255)
            {
                serialIn.Dequeue();
            }

            int discardInit255 = serialIn.Dequeue();
            int direction = serialIn.Dequeue();
            int first = serialIn.Dequeue();
            int second = serialIn.Dequeue();
            int yy = serialIn.Dequeue();

            DateTime timenow = DateTime.Now;

            if (yy == 3)
            {
                first = 255;
                second = 255;
            } else if (yy == 2)
            {
                first = 255;
            } else if (yy == 1)
            {
                second = 255;
            }
            int pos = first * 255 + second;

            //posValBox.Text = discardInit255.ToString() + "," + direction.ToString() + "," +
first.ToString() + "," + second.ToString() + "," + yy.ToString();

            if (velcounter > 15)
            {
                TimeSpan t = timenow.Subtract(prevTime);
                vel = (pos - prevPos) / t.TotalMilliseconds;
                if (t.TotalMilliseconds == 0)
                {
                    if (pos - prevPos < 0)
                    {
                        vel = -999;
                    }
                    else
                    {
                        vel = 999;
                    }
                }
                prevTime = timenow;
                prevPos = pos;

                velcounter = 0;
            }
            else
            {
                velcounter++;
            }


            setNewPos(pos % 255);
            setNewVel(vel);
        }

        private int velIndex = 0;
        private void setNewVel(double data)
        {
            velValBox.Text = data.ToString();
            ChartValues_vel.Add(new MeasureModel
            {
                index = posIndex,
                Value = data
            });
            SetAxisLimits_velChart(velIndex);
            velIndex++;

            if (ChartValues_vel.Count > numOfPointsOnChart) ChartValues_vel.RemoveAt(0);
        }

        private int posIndex = 0;
        private void setNewPos(double data)
        {
            posValBox.Text = data.ToString();
            ChartValues_pos.Add(new MeasureModel
            {
                index = posIndex,
                Value = data
            });
            SetAxisLimits_posChart(posIndex);
            posIndex++;

            if (ChartValues_pos.Count > numOfPointsOnChart) ChartValues_pos.RemoveAt(0);
        }

        private void SetAxisLimits_posChart(int index)
        {
            posChart.AxisX[0].MaxValue = (double)(index - numOfPointsOnChart); // lets force the axis
to be 100ms ahead
            posChart.AxisX[0].MinValue = (double)index; //we only care about the last 8 seconds
        }
        private void SetAxisLimits_velChart(int index)
        {
            velChart.AxisX[0].MaxValue = (double)(index - numOfPointsOnChart); // lets force the axis
to be 100ms ahead
            velChart.AxisX[0].MinValue = (double)index; //we only care about the last 8 seconds
        }
        #endregion
    }

    // livecharts
    public class MeasureModel
    {
        public double index { get; set; }
        public double Value { get; set; }
    }
}
```

C# code, Form1.Designer.cs:

```
namespace _1_cs
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.trackBar1 = new System.Windows.Forms.TrackBar();
            this.label1 = new System.Windows.Forms.Label();
            this.checkBox1 = new System.Windows.Forms.CheckBox();
            this.comboBox1 = new System.Windows.Forms.ComboBox();
            this.disconnectBtn = new System.Windows.Forms.Button();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.posChart = new LiveCharts.WinForms.CartesianChart();
            this.velChart = new LiveCharts.WinForms.CartesianChart();
            this.posValBox = new System.Windows.Forms.TextBox();
            this.velValBox = new System.Windows.Forms.TextBox();
            ((System.ComponentModel.ISupportInitialize)(this.trackBar1)).BeginInit();
            this.SuspendLayout();
            //
            // trackBar1
            //
            this.trackBar1.Location = new System.Drawing.Point(12, 113);
            this.trackBar1.Maximum = 65536;
            this.trackBar1.Name = "trackBar1";
            this.trackBar1.Size = new System.Drawing.Size(776, 56);
            this.trackBar1.TabIndex = 0;
            this.trackBar1.Value = 32768;
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(12, 93);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(123, 17);
            this.label1.TabIndex = 1;
            this.label1.Text = "PWM (0 to 65536)";
            //
            // checkBox1
            //
            this.checkBox1.AutoSize = true;
            this.checkBox1.Location = new System.Drawing.Point(15, 54);
            this.checkBox1.Name = "checkBox1";
            this.checkBox1.Size = new System.Drawing.Size(233, 21);
            this.checkBox1.TabIndex = 2;
            this.checkBox1.Text = "Check for CW, uncheck for CCW";
            this.checkBox1.UseVisualStyleBackColor = true;
            //
            // comboBox1
            //
            this.comboBox1.FormattingEnabled = true;
            this.comboBox1.Location = new System.Drawing.Point(18, 15);
            this.comboBox1.Name = "comboBox1";
            this.comboBox1.Size = new System.Drawing.Size(121, 24);
            this.comboBox1.TabIndex = 3;
            this.comboBox1.SelectedIndexChanged += new
System.EventHandler(this.comboBox1_SelectedIndexChanged);
            //
            // disconnectBtn
            //
            this.disconnectBtn.Location = new System.Drawing.Point(156, 17);
            this.disconnectBtn.Name = "disconnectBtn";
            this.disconnectBtn.Size = new System.Drawing.Size(92, 23);
            this.disconnectBtn.TabIndex = 4;
            this.disconnectBtn.Text = "Disconnect";
            this.disconnectBtn.UseVisualStyleBackColor = true;
            this.disconnectBtn.Click += new System.EventHandler(this.button1_Click);
            //
            // textBox1
            //
            this.textBox1.Location = new System.Drawing.Point(26, 166);
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(100, 22);
            this.textBox1.TabIndex = 5;
            this.textBox1.TextChanged += new System.EventHandler(this.textBox1_TextChanged);
            //
            // posChart
            //
            this.posChart.Location = new System.Drawing.Point(18, 214);
            this.posChart.Name = "posChart";
            this.posChart.Size = new System.Drawing.Size(359, 205);
            this.posChart.TabIndex = 6;
            this.posChart.Text = "posChart";
            //
            // velChart
            //
            this.velChart.Location = new System.Drawing.Point(414, 214);
            this.velChart.Name = "velChart";
            this.velChart.Size = new System.Drawing.Size(374, 205);
            this.velChart.TabIndex = 7;
            this.velChart.Text = "cartesianChart2";
            //
            // posValBox
            //
            this.posValBox.Location = new System.Drawing.Point(26, 416);
            this.posValBox.Name = "posValBox";
            this.posValBox.Size = new System.Drawing.Size(100, 22);
            this.posValBox.TabIndex = 8;
            //
            // velValBox
            //
            this.velValBox.Location = new System.Drawing.Point(414, 416);
            this.velValBox.Name = "velValBox";
            this.velValBox.Size = new System.Drawing.Size(100, 22);
            this.velValBox.TabIndex = 9;
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(800, 450);
            this.Controls.Add(this.velValBox);
            this.Controls.Add(this.posValBox);
            this.Controls.Add(this.velChart);
            this.Controls.Add(this.posChart);
            this.Controls.Add(this.textBox1);
            this.Controls.Add(this.disconnectBtn);
            this.Controls.Add(this.comboBox1);
            this.Controls.Add(this.checkBox1);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.trackBar1);
            this.Name = "Form1";
            this.Text = "Form1";
            ((System.ComponentModel.ISupportInitialize)(this.trackBar1)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion
        private System.Windows.Forms.TrackBar trackBar1;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.CheckBox checkBox1;
        private System.Windows.Forms.ComboBox comboBox1;
        private System.Windows.Forms.Button disconnectBtn;
        private System.Windows.Forms.TextBox textBox1;
        private LiveCharts.WinForms.CartesianChart posChart;
        private LiveCharts.WinForms.CartesianChart velChart;
        private System.Windows.Forms.TextBox posValBox;
        private System.Windows.Forms.TextBox velValBox;
    }
}
```

Appendix 4:

MCU code:

```
#include <msp430.h>

/**
 * main.c
 */

#define LEN 51
#define TRUE 1
#define FALSE 0

#define PJ_ALLON (BIT0 + BIT1 + BIT2 + BIT3) // bits 0-3
#define PJ_ALLOFF 0

volatile unsigned int read = 0;
volatile unsigned char readByte = 0;
volatile unsigned int write = 0;
volatile unsigned char buffer[51] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0 };
volatile unsigned int yeet = 0;

volatile unsigned int readEncoder = FALSE;

unsigned int plus1mod51(unsigned int i)
{
    // range: 0~51*2-1
    if (i + 1 < LEN)
    {
        return i + 1;
    }
    else
    {
        return (i + 1) - LEN;
    }
}

int readBuf()
{
    //read from buffer
    if (read == write)
    {
        return FALSE;
    }
    else
    {
        readByte = buffer[read];
        // doesn't delete old data from buffer, just wait for it to be overwritten later
        read = plus1mod51(read);
        return TRUE;
    }
}

int writeBuf(char n)
{
    if (read == plus1mod51(write))
    {
        // is full
        return FALSE;
    }
    else
    {
        buffer[write] = n;
        write = plus1mod51(write);
        return TRUE;
    }
}

int getBufCount()
{
    if (write >= read)
    {
        return write - read;
    }
    else
    {
        // read < write == write at beginning of array
        return (LEN - read) + write; // i think this is correct lol
    }
}

int parseInt16(char upperByte, char lowerByte, char escByte)
{
    if (escByte == 0x03)
    {
        upperByte = 0xFF;
        lowerByte = 0xFF;
    }
    else if (escByte == 0x02)
    {
        upperByte = 0xFF;
    }
    else if (escByte == 0x01)
    {
        lowerByte = 0xFF;
    }
    int ret = 0;
    ret |= (int) upperByte << 8;
    ret |= (int) lowerByte;
    return ret;
}

void readBufAndSetPWM()
{
    readByte = 0;
    while (!(readByte == 0xFF))
    { // find start byte and remove it
        readBuf();
    }
    char dir = 0;
    char byte1 = 0;
    char byte2 = 0;
    char escByte = 0;

    readBuf();
    dir = readByte;
    if (dir == 0)
    {
        // set ccw
        P3OUT |= BIT5;
        P3OUT &= ~BIT6;
    }
    else
    {
        // set cw
        P3OUT |= BIT6;
        P3OUT &= ~BIT5;
    }
    readBuf();
    byte1 = readByte;
    readBuf();
    byte2 = readByte;
    readBuf();
    escByte = readByte;
    int i = parseInt16(byte1, byte2, escByte);
    yeet = i; // for checking in debugger
    if (i > 65536)
    {
        TB1CCR1 = 65536;
    }
    else
    {
        TB1CCR1 = i;
    }

}

void readEncoderFn()
{
    unsigned int encoder0 = TA0R;
    unsigned int encoder1 = TA1R;
    unsigned int pos = 0;
    int dir = 2;
    if (encoder0 >= encoder1)
    {
        pos = encoder0 - encoder1;
        dir = 0;
    }
    else
    {
        pos = encoder1 - encoder0;
        dir = 1;
    }
    int lowerByte = pos;
    int upperByte = pos >> 8;
    int escByte = 0;
    if (lowerByte == 0xFF && upperByte == 0xFF)
    {
        escByte = 0x03;
        lowerByte = 0x00;
        upperByte = 0x00;
    }
    else if (lowerByte == 0xFF)
    {
        escByte = 0x01;
        lowerByte = 0x00;
    }
    else if (upperByte == 0xFF)
    {
        escByte = 0x02;
        upperByte = 0x00;
    }

    while ((UCA0IFG & UCTXIFG) == 0)
        ;
    UCA0TXBUF = 0xFF;
    while ((UCA0IFG & UCTXIFG) == 0)
        ;
    UCA0TXBUF = dir;
    while ((UCA0IFG & UCTXIFG) == 0)
        ;
    UCA0TXBUF = upperByte;
    while ((UCA0IFG & UCTXIFG) == 0)
        ;
    UCA0TXBUF = lowerByte;
    while ((UCA0IFG & UCTXIFG) == 0)
        ;
    UCA0TXBUF = escByte;
}

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer

    // CLOCK
    // Configure clocks
    CSCTL0 = 0xA500;                    // Write password to modify CS registers
    CSCTL1 = DCOFSEL0 + DCOFSEL1;            // DCO = 8 MHz
    CSCTL2 = SELM0 + SELM1 + SELA0 + SELA1 + SELS0 + SELS1; // MCLK = DCO, ACLK =
DCO, SMCLK = DCO

    // UART
    // Configure ports for UCA0
    P2SEL0 &= ~(BIT0 + BIT1);
    P2SEL1 |= BIT0 + BIT1;
    // Configure UCA0
    UCA0CTLW0 = UCSSEL0;
    UCA0BRW = 52;
    UCA0MCTLW = 0x4900 + UCOS16 + UCBRF0;
    UCA0IE |= UCRXIE;
    // global interrupt enable
    _EINT();

    // PWM
    P3DIR |= BIT5 + BIT6;
    P3OUT |= BIT5;
    P3OUT &= ~BIT6;
    // Set Timer B
    P3DIR |= BIT4;
    P3SEL1 &= ~(BIT4);
    P3SEL0 |= BIT4;
    // Set PWM out
    TB1CCR0 = 65536;
    TB1CCTL1 = OUTMOD_7;
    TB1CCR1 = 30000;
    //TB1CCR1 = 0;
    TB1CTL = TBSSEL_2 + MC_2 + TBCLR;
    // READ ENCODER
    // Set Timer A
    P1DIR &= ~(BIT1 + BIT2);
    P1SEL1 &= ~(BIT1 + BIT2);
    P1SEL0 |= (BIT1 + BIT2);
    // Set Timer A for capture
    TA1CTL |= TASSEL_0 + MC_2 + TACLR;
    TA0CTL |= TASSEL_0 + MC_2 + TACLR;

    // INTERRUPT
    TB2CTL |= TBSSEL_1 + MC_1;
    TB2CCR0 = 300000;     // arbitrary interval
    TB2CCTL0 = CCIE;    // enable interrupt

    while (1)
    {
        if (getBufCount() >= 5)
        {
            readBufAndSetPWM();
        }
        if (readEncoder == TRUE)
        {
            readEncoderFn();
            readEncoder = FALSE;
        }
    }

    return 0;
}

#pragma vector = USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
    unsigned char RxByte;
    RxByte = UCA0RXBUF;
    //write to buffer
    if (writeBuf(RxByte) == FALSE)
    {
        // send 0xFF 3 times as error msg if write fail
        while ((UCA0IFG & UCTXIFG) == 0)
            ;
        UCA0TXBUF = 0xFF;
        while ((UCA0IFG & UCTXIFG) == 0)
            ;
        UCA0TXBUF = 0xFF;
        while ((UCA0IFG & UCTXIFG) == 0)
            ;
        UCA0TXBUF = 0xFF;
    }
}

volatile unsigned int a = 0;

#pragma vector = TIMER2_B0_VECTOR
__interrupt void TIMER2_B0_ISR(void)
{
    readEncoder = TRUE;
}
```

Appendix 5

MCU code (stepper motor only):

```
#include <msp430.h>

/**
 * main.c
 */

#define LEN 51
#define TRUE 1
#define FALSE 0

#define PJ_ALLON (BIT0 + BIT1 + BIT2 + BIT3) // bits 0-3
#define PJ_ALLOFF 0

volatile unsigned int read = 0;
volatile unsigned char readByte = 0;
volatile unsigned int write = 0;
volatile unsigned char buffer[51] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                      0, 0, 0, 0, 0, 0, 0, 0, 0 };
volatile unsigned int yeet = 0;

unsigned int plus1mod51(unsigned int i)
{
    // range: 0~51*2-1
    if (i + 1 < LEN)
    {
        return i + 1;
    }
    else
    {
        return (i + 1) - LEN;
    }
}

int readBuf()
{
    //read from buffer
    if (read == write)
    {
        return FALSE;
    }
    else
    {
        readByte = buffer[read];
        // doesn't delete old data from buffer, just wait for it to be overwritten later
        read = plus1mod51(read);
        return TRUE;
    }
}

int writeBuf(char n)
{
    if (read == plus1mod51(write))
    {
        // is full
        return FALSE;
    }
    else
    {
        buffer[write] = n;
        write = plus1mod51(write);
        return TRUE;
    }
}

int getBufCount()
{
    if (write >= read)
    {
        return write - read;
    }
    else
    {
        // read < write == write at beginning of array
        return (LEN - read) + write; // i think this is correct lol
    }
}

int parseInt16(char upperByte, char lowerByte, char escByte)
{
    if (escByte == 0x03)
    {
        upperByte = 0xFF;
        lowerByte = 0xFF;
    }
    else if (escByte == 0x02)
    {
        upperByte = 0xFF;
    }
    else if (escByte == 0x01)
    {
        lowerByte = 0xFF;
    }
    int ret = 0;
    ret |= (int) upperByte << 8;
    ret |= (int) lowerByte;
    return ret;
}

void stepStepper(int p3, int p4, int p5, int p6)
{
    if (p3 == TRUE)
    {
        P1OUT |= BIT3;
    }
    else
    {
        P1OUT &= ~BIT3;
    }
    if (p4 == TRUE)
    {
        P1OUT |= BIT4;
    }
    else
    {
        P1OUT &= ~BIT4;
    }
    if (p5 == TRUE)
    {
        P1OUT |= BIT5;
    }
    else
    {
        P1OUT &= ~BIT5;
    }
    if (p6 == TRUE)
    {
        P1OUT |= BIT6;
    }
    else
    {
        P1OUT &= ~BIT6;
    }
}

unsigned int plus1loop8(state) {
    if (state == 7) {
        return 0;
    } else {
        return state + 1;
    }
}

unsigned int minus1loop8(state) {
    if (state == 0) {
        return 7;
    } else {
        return state - 1;
    }
}

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer

    // Configure clocks
    CSCTL0 = 0xA500;// Write password to modify CS registers
    CSCTL1 = DCOFSEL0 + DCOFSEL1;// DCO = 8 MHz
    CSCTL2 = SELM0 + SELM1 + SELA0 + SELA1 + SELS0 + SELS1;// MCLK = DCO, ACLK =
DCO, SMCLK = DCO

    // Configure ports for UCA0
    P2SEL0 &= ~(BIT0 + BIT1);
    P2SEL1 |= BIT0 + BIT1;

    // Configure UCA0
    UCA0CTLW0 = UCSSEL0;
    UCA0BRW = 52;
    UCA0MCTLW = 0x4900 + UCOS16 + UCBRF0;
    UCA0IE |= UCRXIE;

    // global interrupt enable
    _EINT();

    // Stepper set up
    P1DIR |= BIT3 + BIT4 + BIT5 + BIT6;
    int speedCount = 0;
    int speedCountUpTo = 5000;
    int stepCount = 0;
    int stepCountUpTo = 0;
    unsigned int state = 0;
    int direction = TRUE;
    // ccw sequence
    int p3[8] =
    {   FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE};
    int p4[8] =
    {   FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE};
    int p5[8] =
    {   FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, FALSE, FALSE};
    int p6[8] =
    {   TRUE, TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE};
    stepStepper(p3[state], p4[state], p5[state], p6[state]);

    while (1)
    {
        yeet = speedCountUpTo;
        // do stepping
        if (speedCountUpTo != 0 && stepCount < stepCountUpTo) {
            if (speedCount >= speedCountUpTo)
            {
                if (direction == TRUE) // true = cw
                {
                    state = plus1loop8(state);
                }
                else
                {
                    state = minus1loop8(state);
                }
                stepStepper(p3[state], p4[state], p5[state], p6[state]);
                speedCount = 0;
                stepCount = stepCount + 1;
            }
            else
            {
                speedCount = speedCount + 1;
            }
        }

        // read uart buffer
        if (getBufCount() >= 5)
        {
            readByte = 0;
            while (!(readByte == 0xFF))
            { // find start byte and remove it
                readBuf();
            }
            int x_stepper = 0; // 0~38, 19 == 0 displace
            int y_dc = 0;
            int speed = 0;
            char escByte = 0;

            readBuf();
            x_stepper = readByte;
            readBuf();
            y_dc = readByte;
            readBuf();
```

```
            speed = readByte;
            readBuf();
            escByte = readByte;

            if (speed == 0 || x_stepper == 19)
            {
               // set cw
               direction = TRUE;
               speed = 0;
               stepCountUpTo = 0;
            }
            else
            {
               if (x_stepper > 19) {
                  // set cw
                  direction = TRUE;
                  stepCountUpTo = (x_stepper - 19)*64*16;
               } else {
                  // set ccw
                  direction = FALSE;
                  stepCountUpTo = (19 - x_stepper)*64*16;
               }
               speedCountUpTo = 718-speed*7;
            }
            speedCount = 0;
            stepCount = 0;
         }
   }

   return 0;
}

#pragma vector = USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
   unsigned char RxByte;
   RxByte = UCA0RXBUF;
   //write to buffer
   if (writeBuf(RxByte) == FALSE)
   {
      // send 0xFF 3 times as error msg if write fail
      while ((UCA0IFG & UCTXIFG) == 0)
      ;
      UCA0TXBUF = 0xFF;
      while ((UCA0IFG & UCTXIFG) == 0)
      ;
      UCA0TXBUF = 0xFF;
      while ((UCA0IFG & UCTXIFG) == 0)
```

```
        ;
        UCA0TXBUF = 0xFF;
    }

}
```

C# code, Form1.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO.Ports;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        private SerialPort serialPort = null;
        bool clicked1 = false;
        bool clicked2 = false;

        public Form1()
        {
            InitializeComponent();

            comboBox1.Items.AddRange(SerialPort.GetPortNames()); //set up combo box

            this.FormClosing += CloseSerial; // set up auto serial port close, in case user didn't click
disconnect

            disconnectBtn.Enabled = false; // turn disconnect button off since there's nothing to disconnect
now

            this.trackBar1.Scroll += (s, e) =>
            {
                if (clicked1)
                    return;
            };

            this.trackBar1.MouseDown += (s,
                                e) =>
            {
                clicked1 = true;
        };
        trackBar1.MouseUp += (s,
                        e) =>
        {
            if (!clicked1)
                return;

            clicked1 = false;
            stepperPosBox.Text = trackBar1.Value.ToString();
        };

        this.trackBar2.Scroll += (s, e) =>
        {
            if (clicked2)
                return;
        };

        this.trackBar2.MouseDown += (s,
                        e) =>
        {
            clicked2 = true;
        };
        trackBar2.MouseUp += (s,
                        e) =>
        {
            if (!clicked2)
                return;

            clicked2 = false;
            dcPosBox.Text = trackBar2.Value.ToString();
        };
    }


    private void CloseSerial(object sender, FormClosingEventArgs e)
    {
        if (serialPort != null && serialPort.IsOpen)
        {
            serialPort.Close(); // close port if not closed before app is closed
        }
    }

    private void serialDataHandler(object sender, SerialDataReceivedEventArgs e)
    {
        // do nothing
    }

    private void sendData(byte x, byte y, byte speed)
```

```
        {
            byte[] send = new byte[5];
            send[0] = 0xFF;
            send[1] = x;
            send[2] = y;
            send[3] = speed;
            send[4] = 0;
            if (serialPort != null)
            {
                serialPort.Write(send, 0, 5);
            }
        }

        private void disconnectBtn_Click(object sender, EventArgs e)
        {
            serialPort.Close();
            serialPort = null; // close and null the serial port

            disconnectBtn.Enabled = false; // turn off disconnect button since there's nothing to disconnect
now

        }

        private void comboBox1_SelectedIndexChanged_1(object sender, EventArgs e)
        {
            string portName = comboBox1.SelectedItem.ToString();

            if (serialPort == null) // if the previous port is not closed and null, then don't make a new
connection
            {
                serialPort = new SerialPort(portName, 9600, Parity.None, 8, StopBits.One);
                while (!serialPort.IsOpen)
                {
                    serialPort.Open();
                } // TA recommended something like this in case open doesn't actually open
                serialPort.ReadTimeout = 300;
                serialPort.DiscardInBuffer(); // remove data before reading is supposed to start
                serialPort.DataReceived += new SerialDataReceivedEventHandler(serialDataHandler);

                disconnectBtn.Enabled = true; // now that there's something to disconnect, turn disconnect
button on
            }
        }

        private void textBox1_TextChanged_1(object sender, EventArgs e)
        {
            // do nothing lol
        }
        private void button1_Click(object sender, EventArgs e)
        {
            int stepperDisplace = int.Parse(stepperPosBox.Text) + 19; // scale is -19 to 19
            int dcDisplace = int.Parse(dcPosBox.Text) + 19; // scale is -19 to 19
            int speed = int.Parse(speedBox.Text);
            sendData((byte)stepperDisplace, (byte)dcDisplace, (byte)speed);
        }
    }
}
```

C# code, Form1.Designer.cs:

```
namespace WindowsFormsApp1
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.trackBar1 = new System.Windows.Forms.TrackBar();
            this.stepperPosBox = new System.Windows.Forms.TextBox();
            this.label2 = new System.Windows.Forms.Label();
            this.comboBox1 = new System.Windows.Forms.ComboBox();
            this.disconnectBtn = new System.Windows.Forms.Button();
            this.button1 = new System.Windows.Forms.Button();
            this.label3 = new System.Windows.Forms.Label();
            this.label4 = new System.Windows.Forms.Label();
            this.dcPosBox = new System.Windows.Forms.TextBox();
            this.trackBar2 = new System.Windows.Forms.TrackBar();
            this.label5 = new System.Windows.Forms.Label();
            this.speedBox = new System.Windows.Forms.TextBox();
            ((System.ComponentModel.ISupportInitialize)(this.trackBar1)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.trackBar2)).BeginInit();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(12, 58);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(126, 17);
            this.label1.TabIndex = 0;
            this.label1.Text = "X / Stepper control";
            //
            // trackBar1
            //
            this.trackBar1.Location = new System.Drawing.Point(5, 120);
            this.trackBar1.Maximum = 19;
            this.trackBar1.Minimum = -19;
            this.trackBar1.Name = "trackBar1";
            this.trackBar1.Size = new System.Drawing.Size(776, 56);
            this.trackBar1.TabIndex = 1;
            //
            // stepperPosBox
            //
            this.stepperPosBox.Location = new System.Drawing.Point(109, 92);
            this.stepperPosBox.Name = "stepperPosBox";
            this.stepperPosBox.Size = new System.Drawing.Size(100, 22);
            this.stepperPosBox.TabIndex = 2;
            this.stepperPosBox.Text = "0";
            this.stepperPosBox.TextChanged += new
System.EventHandler(this.textBox1_TextChanged_1);
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Location = new System.Drawing.Point(13, 89);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(90, 17);
            this.label2.TabIndex = 3;
            this.label2.Text = "Position (cm)";
            //
            // comboBox1
            //
            this.comboBox1.FormattingEnabled = true;
            this.comboBox1.Location = new System.Drawing.Point(16, 12);
            this.comboBox1.Name = "comboBox1";
            this.comboBox1.Size = new System.Drawing.Size(121, 24);
            this.comboBox1.TabIndex = 7;
            this.comboBox1.SelectedIndexChanged += new
System.EventHandler(this.comboBox1_SelectedIndexChanged_1);
            //
            // disconnectBtn
            //
            this.disconnectBtn.Location = new System.Drawing.Point(163, 12);
            this.disconnectBtn.Name = "disconnectBtn";
            this.disconnectBtn.Size = new System.Drawing.Size(90, 23);
            this.disconnectBtn.TabIndex = 8;
            this.disconnectBtn.Text = "Disconnect";
            this.disconnectBtn.UseVisualStyleBackColor = true;
            this.disconnectBtn.Click += new System.EventHandler(this.disconnectBtn_Click);
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(702, 335);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(75, 23);
            this.button1.TabIndex = 9;
            this.button1.Text = "Send";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // label3
            //
            this.label3.AutoSize = true;
            this.label3.Location = new System.Drawing.Point(13, 179);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(99, 17);
            this.label3.TabIndex = 10;
            this.label3.Text = "Y /  DC control";
            //
            // label4
            //
            this.label4.AutoSize = true;
            this.label4.Location = new System.Drawing.Point(13, 208);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(90, 17);
            this.label4.TabIndex = 11;
            this.label4.Text = "Position (cm)";
```

```
//
// dcPosBox
//
this.dcPosBox.Location = new System.Drawing.Point(109, 208);
this.dcPosBox.Name = "dcPosBox";
this.dcPosBox.Size = new System.Drawing.Size(100, 22);
this.dcPosBox.TabIndex = 12;
this.dcPosBox.Text = "0";
//
// trackBar2
//
this.trackBar2.Location = new System.Drawing.Point(12, 236);
this.trackBar2.Maximum = 19;
this.trackBar2.Minimum = -19;
this.trackBar2.Name = "trackBar2";
this.trackBar2.Size = new System.Drawing.Size(765, 56);
this.trackBar2.TabIndex = 13;
//
// label5
//
this.label5.AutoSize = true;
this.label5.Location = new System.Drawing.Point(13, 295);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(75, 17);
this.label5.TabIndex = 14;
this.label5.Text = "Speed (%)";
//
// speedBox
//
this.speedBox.Location = new System.Drawing.Point(109, 295);
this.speedBox.Name = "speedBox";
this.speedBox.Size = new System.Drawing.Size(100, 22);
this.speedBox.TabIndex = 15;
this.speedBox.Text = "100";
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(800, 376);
this.Controls.Add(this.speedBox);
this.Controls.Add(this.label5);
this.Controls.Add(this.trackBar2);
this.Controls.Add(this.dcPosBox);
this.Controls.Add(this.label4);
this.Controls.Add(this.label3);
this.Controls.Add(this.button1);
this.Controls.Add(this.disconnectBtn);
```

```
            this.Controls.Add(this.comboBox1);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.stepperPosBox);
            this.Controls.Add(this.trackBar1);
            this.Controls.Add(this.label1);
            this.Name = "Form1";
            this.Text = "Form1";
            ((System.ComponentModel.ISupportInitialize)(this.trackBar1)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.trackBar2)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TrackBar trackBar1;
        private System.Windows.Forms.TextBox stepperPosBox;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.ComboBox comboBox1;
        private System.Windows.Forms.Button disconnectBtn;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.TextBox dcPosBox;
        private System.Windows.Forms.TrackBar trackBar2;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.TextBox speedBox;
    }
}
```