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

namespace MotorController
{
    public partial class Form1 : Form
    {
        const int packetLength = 5;
        // Packet indices
        const int startIndex = 0, commandIndex = 1, MSBIndex = 2, LSBIndex = 3, escapeIndex = 4;
        // Command Byte command values
        const byte dcStop = 0, dcCW = 1, dcCCW = 2, stepCW = 3, stepCCW = 4, stepContCW = 5, stepContCCW = 6, stepStop = 7,
            xZero = 8, xTransmit = 9, yZero = 10, yTransmit = 11, xyTransmitY = 12, xyTransmitX = 13, velPercent = 14;

        // For scaling DC and stepper motor trackbars
        const int dcTickMax = 65535;    // obsolete
        const int dcTick0 = 0;          // obsolete
        const int dcDeadzone = 0; //500;
        const int stepTickMax = 55705; //60585;
        const int stepTick0 = 0; //30000;
        const int stepDeadzone = 0; //200;

        // Motor and gantry parameters
        const int motorCPR = 48;
        const double gearRatio = 20.4;
        const double yAxisMaxLength = 123.2;
        const int toothPitch = 2;
        const int toothNumber = 20;
        const double Kd = (double)0xFFFF / yAxisMaxLength;

        // Velocity scaling
        double vMax = 0xC800;
        double vMin = 0xA00;

        // Timing
        int samplingPeriod = 200;
        int timeCount = 0;
        int prevTimeCount = 0;
        double lastCount = 0;


        bool motorSpeedChanged = false;                                 // Flag for DC Motor and Stepper Motor Change
        byte[] output = new byte[packetLength];                          // Output packet array
        ConcurrentQueue<Int32> dataQueue = new ConcurrentQueue<Int32>();    // Queue for reading bytes from MSP
        StreamWriter outputFile;                                         // File for recording DC motor data
        int dcLSB, dcMSB, stepLSB, stepMSB, velLSB, velMSB;                 // Misc. variables

        public Form1()
        {
            InitializeComponent();
            output[startIndex] = 255;                                    // Intitialize start byte
            comboBoxCOMPorts.Items.Clear();
            comboBoxCOMPorts.Items.AddRange(System.IO.Ports.SerialPort.GetPortNames());     // Add COM ports to combo box
            if (comboBoxCOMPorts.Items.Count == 0)
                comboBoxCOMPorts.Text = "No COM ports!";
            else
            {
                comboBoxCOMPorts.SelectedIndex = comboBoxCOMPorts.Items.Count - 1;          // set combo box index to last port by default
            }
        }
        private void buttonSelectFilename_Click(object sender, EventArgs e)
        // For opening save file dialog
        {
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
                textBoxFileName.Text = saveFileDialog1.FileName;
        }

        private void checkBoxSave_CheckedChanged(object sender, EventArgs e)
        // For checking if recording data and starting new streamwriter
        {
            if (checkBoxSave.Checked)
                outputFile = new StreamWriter(textBoxFileName.Text);
            else if (!checkBoxSave.Checked)
                outputFile.Close();
        }

        private void buttonZeroStepper_Click(object sender, EventArgs e)
        // Sends packet to zero the stepper motor position
        {
            output[commandIndex] = yZero;
            output[MSBIndex] = 0;
            output[LSBIndex] = 0;
            output[escapeIndex] = 0;
            serialPort1.Write(output, startIndex, packetLength);
        }

        private void buttonZeroDC_Click(object sender, EventArgs e)
        // Sends packet to zero the DC motor position
        {
            output[commandIndex] = xZero;
            output[MSBIndex] = 0;
            output[LSBIndex] = 0;
            output[escapeIndex] = 0;
            serialPort1.Write(output, startIndex, packetLength);
        }

        private void buttonTransmitXY_Click(object sender, EventArgs e)
        // Sends packets to move both DC and stepper motors from position and velocity input
        {
            // Get position and velocity values from text boxes and convert to useful values
            double xLength = Kd * Convert.ToDouble(textBoxXPos.Text);
            double yLength = Kd * Convert.ToDouble(textBoxYPos.Text);
            double velocity = (vMax - vMin) / 100 * Convert.ToDouble(textBoxVelocity.Text) + vMin;

            // Split values into LSB and MSB
            dcMSB = (Int32)xLength >> 8;
            dcLSB = (Int32)xLength & 0xFF;
            stepMSB = (Int32)yLength >> 8;
            stepLSB = (Int32)yLength & 0xFF;
            velMSB = (Int32)velocity >> 8;
            velLSB = (Int32)velocity & 0xFF;

            // Assign x-y control y transmit in command byte
            output[commandIndex] = xyTransmitY;

            // Check if either byte is 255 and assign escape byte accordingly
            output[escapeIndex] = 0;
            if (stepLSB == 255) { output[escapeIndex] = 1; stepLSB = 0; }
            if (stepMSB == 255) { output[escapeIndex] += 2; stepMSB = 0; }

            // Assign PWM bytes in buffer
            output[MSBIndex] = (byte)stepMSB;
            output[LSBIndex] = (byte)stepLSB;

            // Write ytransmit packet to serial port
            serialPort1.Write(output, startIndex, packetLength);

            // Assign x-y transmit x transmit in command byte
            output[commandIndex] = xyTransmitX;

            // Check if either byte is 255 and assign escape byte accordingly
            output[escapeIndex] = 0;
            if (dcLSB == 255) { output[escapeIndex] = 1; dcLSB = 0; }
            if (dcMSB == 255) { output[escapeIndex] += 2; dcMSB = 0; }

            // Assign PWM bytes in buffer
            output[MSBIndex] = (byte)dcMSB;
            output[LSBIndex] = (byte)dcLSB;

            // Sleep to avoid interrupting firmware
            System.Threading.Thread.Sleep(300);

            // Write xtransmit packet to serial port
            serialPort1.Write(output, startIndex, packetLength);

            // Assign x-y control velocity in command byte
            output[commandIndex] = velPercent;

            // Check if either byte is 255 and assign escape byte accordingly
            output[escapeIndex] = 0;
            if (velLSB == 255) { output[escapeIndex] = 1; velLSB = 0; }
            if (velMSB == 255) { output[escapeIndex] += 2; velMSB = 0; }

            // Assign PWM bytes in buffer
            output[MSBIndex] = (byte)velMSB;
            output[LSBIndex] = (byte)velLSB;

            // Sleep to avoid interrupting firmware
            System.Threading.Thread.Sleep(300);

            // Write velocity packet to serial port
            serialPort1.Write(output, startIndex, packetLength);
        }

        private void buttonTransmitY_Click(object sender, EventArgs e)
        {
            // Get position value from textbox and convert to LSB and MSB
            double newLength = Kd * Convert.ToDouble(textBoxYPos.Text);
            stepMSB = (Int32)newLength >> 8;
            stepLSB = (Int32)newLength & 0xFF;

            // Assign y-transmit in command byte
            output[commandIndex] = yTransmit;

            // Check if either byte is 255 and assign escape byte accordingly
            output[escapeIndex] = 0;
            if (stepLSB == 255) { output[escapeIndex] = 1; stepLSB = 0; }
            if (stepMSB == 255) { output[escapeIndex] += 2; stepMSB = 0; }

            // Assign PWM bytes in buffer
            output[MSBIndex] = (byte)stepMSB;
            output[LSBIndex] = (byte)stepLSB;

            // Write x-transmit packet to serial port
            serialPort1.Write(output, startIndex, packetLength);
        }

        private void buttonTransmitX_Click(object sender, EventArgs e)
        {
            // Get position value from textbox and convert to LSB and MSB
            double newLength = Kd * Convert.ToDouble(textBoxXPos.Text);
            dcMSB = (Int32)newLength >> 8;
            dcLSB = (Int32)newLength & 0xFF;

            // Assign x-transmit in command byte
            output[commandIndex] = xTransmit;

            // Check if either byte is 255 and assign escape byte accordingly
            output[escapeIndex] = 0;
            if (dcLSB == 255) { output[escapeIndex] = 1; dcLSB = 0; }
            if (dcMSB == 255) { output[escapeIndex] += 2; dcMSB = 0; }

            // Assign PWM bytes in buffer
            output[MSBIndex] = (byte)dcMSB;
            output[LSBIndex] = (byte)dcLSB;

            // Write y-transmit to serial port
            serialPort1.Write(output, startIndex, packetLength);
        }

        private void buttonClearChart_Click(object sender, EventArgs e)
        // Clears the plot on the chart
        {
            timeCount = 0;
            chartPosSpeed.Series["Position"].Points.Clear();
            chartPosSpeed.Series["Speed"].Points.Clear();
        }

        private void timerWrite_Tick(object sender, EventArgs e)
        // On timerWrite tick, detects if DC or Stepper motor speed has changed and writes the output packet to the serial port
        {
            if (motorSpeedChanged)
            {
                serialPort1.Write(output, startIndex, packetLength);
                motorSpeedChanged = false;
            }
        }

        private void timerRead_Tick(object sender, EventArgs e)
        // On timerRead tick, dequeues dataQueue and sends dequeued bytes to the position and speed textboxes
        {
            // Misc. variables
            int state = 0;
            int MSB = 0;
            int LSB = 0;
            int instByte = 0;
            double newCount;
            double position;
            double speed;
            int nextByte;

            // While TryDequeue from the dataQueue returns true
            while (dataQueue.TryDequeue(out nextByte))
            {
                // Check if 255 (start byte) and if so state = 1
                if (nextByte == 255)
                {
                    state = 1;
                }
                // Check if state = 1 for instruction byte
                else if (state == 1)
                {
                    instByte = nextByte;
                    if (instByte == 0) { samplingPeriod = 200; }            // If instruction byte is zero, set
```

```
               sampling period to 200ms
                else if (instByte == 1) { samplingPeriod =
               20; }        // Else if instruction byte is one, set
               sampling period to 20ms
                state =
               2;                                               // Set
               state = 2
            }
            // Check if state = 2 for MSB byte
            else if (state == 2)
            {
                MSB =
               nextByte;                                          //
               Assign MSB
                state =
               3;                                               // Set
               state = 3
            }
            // Check if state = 3 for LSB byte
            else if (state == 3)
            {
                LSB =
               nextByte;                                          //
               Assign LSB
                state =
               4;                                               // Set
               state = 4
            }
            // Check if state = 4 for final byte (escape byte)
            else if (state == 4)
            {
                if (nextByte % 2 != 0) { LSB =
               255; }                  // Check if escape byte is odd
               (1 or 3) and set LSB to 255
                if (nextByte > 1) { MSB =
               255; }                       // Check if escape byte is
                even (2) set MSB to 255

                // Combine LSB and MSB to get encoder counts and
               multiply by 4 for quadrature signal
                newCount = (4 * ((MSB << 8) | LSB));

                // Calculate position in mm and speed in Hz
                position = (double)(newCount * toothPitch *
               toothNumber) / (double)(motorCPR * gearRatio);       //
               [mm]
                speed = 1000 * (double)(newCount - lastCount) /
               (double)(samplingPeriod * motorCPR * gearRatio); // [Hz]

```

```
                    // Assign DC position and speed textboxes
                    textBoxDCPosition.Text = position.ToString();
                    textBoxDCSpeedHz.Text = speed.ToString();
                    textBoxDCSpeedRPM.Text = (60 * speed).ToString();

                    // Assign chart values
                    chartPosSpeed.Series["Position"].Points.AddXY(timeCount, position);
                    chartPosSpeed.Series["Speed"].Points.AddXY(timeCount, 60 * speed);

                    // Check if save file checkbox is checked and if so write the time and position to the outputFile
                    if (checkBoxSave.Checked == true)
                    {
                        outputFile.Write(timeCount.ToString() + ", " + position.ToString() + "\r\n");
                    }

                    // Set previous time and encoder count and set state to 0
                    prevTimeCount = timeCount;
                    lastCount = newCount;
                    state = 0;
                }
            }
        }


        private void getOutputPacketArray()
        // Takes values in packet textboxes in form and assigns them to the output packet array
        {

            output[startIndex] = Convert.ToByte(textBoxStart.Text);
            output[commandIndex] = Convert.ToByte(textBoxCommand.Text);
            output[MSBIndex] = Convert.ToByte(textBoxPWM1.Text);
            output[LSBIndex] = Convert.ToByte(textBoxPWM2.Text);
            output[escapeIndex] = Convert.ToByte(textBoxEscape.Text);
        }

        private void buttonConnect_Click(object sender, EventArgs e)
        // Connects or disconnects serial port and sets baud rate from textbox. Also starts read and write timers
        {
            if (serialPort1.IsOpen == true)
            {
                buttonConnect.Text = "Connect";
                serialPort1.Close();
            }
            else
            {
                serialPort1.PortName = comboBoxCOMPorts.Text;
                buttonConnect.Text = "Disconnect";
                serialPort1.BaudRate = Convert.ToInt16(textBoxBaud.Text);
                serialPort1.Open();
                timerRead.Enabled = true;
                timerWrite.Enabled = true;
            }
        }

        private void buttonStopDC_Click(object sender, EventArgs e)
        // Sends stop DC motor packet to serial port
        {
            output[commandIndex] = dcStop;
            serialPort1.Write(output, startIndex, packetLength);
            trackBarDCSpeed.Value = 0;
        }

        private void buttonStopStepper_Click(object sender, EventArgs e)
        // Sends stop stepper motor packet to serial port
        {
            output[commandIndex] = stepStop;
            serialPort1.Write(output, startIndex, packetLength);
            trackBarStepperSpeed.Value = 0;
        }

        private void buttonStepCW_Click(object sender, EventArgs e)
        // Sends CW step packet to serial port
        {
            output[commandIndex] = stepCW;
            serialPort1.Write(output, startIndex, packetLength);
        }
        private void buttonStepCCW_Click(object sender, EventArgs e)
        // Sends CCW step packet to serial port
        {
            output[commandIndex] = stepCCW;
            serialPort1.Write(output, startIndex, packetLength);
        }

        private void buttonTransmit_Click(object sender, EventArgs e)
        // Assigns output array from packet textboxes and writes packet to serial prot
        {
            if (serialPort1.IsOpen == true)
            {
                getOutputPacketArray();
                serialPort1.Write(output, startIndex, packetLength);
            }
            else
            {
                textBoxUserConsole.AppendText("Serial port is closed\r\n");
            }
        }

        private void serialPort1_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
        // On data receive, gets new bytes from serial port and queues them in dataQueue
        {
            int newByte = 0;
            int bytesToRead;
            bytesToRead = serialPort1.BytesToRead;

            while (bytesToRead != 0)
            {
                newByte = serialPort1.ReadByte();           // Gets new byte from serial port
                dataQueue.Enqueue(newByte);                 // Queues it in dataQueue
                bytesToRead = serialPort1.BytesToRead;      // Checks for more bytes
            }
        }

        private void trackBarDCSpeed_ValueChanged(object sender, EventArgs e)
        // Assigns command byte, PWM bytes, and escape byte from DC motor track bar when the track bar value changes
        {
            // Check direction
            if (trackBarDCSpeed.Value > 0) { output[commandIndex] = dcCW; }
            else { output[commandIndex] = dcCCW; }

            // Display speed
            DCSpeed.Text = (100 * (double)trackBarDCSpeed.Value / (double)trackBarDCSpeed.Maximum).ToString();

            // Deadzone
            if (Math.Abs(trackBarDCSpeed.Value) < dcDeadzone)
            {
                dcLSB = 0;
                dcMSB = 0;
            }
            else
            {
                // Take abs value and scale
                dcLSB = Math.Abs(trackBarDCSpeed.Value) & 0xFF;
                dcMSB = Math.Abs(trackBarDCSpeed.Value) >> 8;
            }

            // Check if either byte is 255 and assign escape byte accordingly
            output[escapeIndex] = 0;
            if (dcLSB == 255) { output[escapeIndex] = 1; dcLSB = 0; }
            if (dcMSB == 255) { output[escapeIndex] += 2; dcMSB = 0; }

            // Assign PWM bytes in buffer
            output[MSBIndex] = Convert.ToByte(dcMSB);
            output[LSBIndex] = Convert.ToByte(dcLSB);

            // Flag motor speed changed
            motorSpeedChanged = true;
        }

        private void trackBarStepperSpeed_ValueChanged(object sender, EventArgs e)
        // Assigns command byte, PWM bytes, and escape byte from stepper motor track bar when the track bar value changes
        {
            // Check direction
            if (trackBarStepperSpeed.Value > 0) { output[commandIndex] = stepContCW; }
            else { output[commandIndex] = stepContCCW; }

            // Display speed
            StepperSpeed.Text = (100 * (double) trackBarStepperSpeed.Value / (double) trackBarStepperSpeed.Maximum).ToString();

            // Deadzone
            if (Math.Abs(trackBarStepperSpeed.Value) < stepDeadzone)
            {
                stepLSB = 0;
                stepMSB = 0;
            }
            else
            {
                // Take abs value and scale
                stepLSB = Math.Abs(trackBarStepperSpeed.Value * (stepTickMax - stepTick0) / trackBarStepperSpeed.Maximum + stepTick0) & 0xFF;
                stepMSB = Math.Abs(trackBarStepperSpeed.Value * (stepTickMax - stepTick0) / trackBarStepperSpeed.Maximum
                   + stepTick0) >> 8;
            }

            // Check if either byte is 255 and assign escape byte accordingly
            output[escapeIndex] = 0;
            if (stepLSB == 255) { output[escapeIndex] = 1; }
            if (stepMSB == 255) { output[escapeIndex] += 2; }

            // Assign PWM bytes in buffer
            output[MSBIndex] = Convert.ToByte(stepMSB);
            output[LSBIndex] = Convert.ToByte(stepLSB);

            // Flag motor speed changed
            motorSpeedChanged = true;
        }
    }
}
```