```
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;
```

```
47          int timeCount = 0;
48          int prevTimeCount = 0;
49          double lastCount = 0;
50
51
52          bool motorSpeedChanged =                                        ↵
              false;                                    // Flag for DC Motor ↵
              and Stepper Motor Change
53          byte[] output = new byte                                        ↵
              [packetLength];                            // Output packet   ↵
              array
54          ConcurrentQueue<Int32> dataQueue = new ConcurrentQueue<Int32>   ↵
              ();    // Queue for reading bytes from MSP
55          StreamWriter                                                    ↵
              outputFile;                                          // File   ↵
              for recording DC motor data
56          int dcLSB, dcMSB, stepLSB, stepMSB, velLSB,                     ↵
              velMSB;                  // Misc. variables
57
58          public Form1()
59          {
60              InitializeComponent();
61              output[startIndex] =                                        ↵
                  255;                                                  / ↵
                  / Intitialize start byte
62              comboBoxCOMPorts.Items.Clear();
63              comboBoxCOMPorts.Items.AddRange                             ↵
                  (System.IO.Ports.SerialPort.GetPortNames());     // Add COM ↵
                  ports to combo box
64              if (comboBoxCOMPorts.Items.Count == 0)
65                  comboBoxCOMPorts.Text = "No COM ports!";
66              else
67              {
68                  comboBoxCOMPorts.SelectedIndex =                        ↵
                      comboBoxCOMPorts.Items.Count - 1;          // set combo ↵
                      box index to last port by default
69              }
70          }
71          private void buttonSelectFilename_Click(object sender, EventArgs ↵
              e)
72          // For opening save file dialog
73          {
74              if (saveFileDialog1.ShowDialog() == DialogResult.OK)
75                  textBoxFileName.Text = saveFileDialog1.FileName;
76          }
77
78          private void checkBoxSave_CheckedChanged(object sender, EventArgs ↵
              e)
79          // For checking if recording data and starting new streamwriter
```

```
80          {
81              if (checkBoxSave.Checked)
82                  outputFile = new StreamWriter(textBoxFileName.Text);
83              else if (!checkBoxSave.Checked)
84                  outputFile.Close();
85          }
86
87          private void buttonZeroStepper_Click(object sender, EventArgs e)
88          // Sends packet to zero the stepper motor position
89          {
90              output[commandIndex] = yZero;
91              output[MSBIndex] = 0;
92              output[LSBIndex] = 0;
93              output[escapeIndex] = 0;
94              serialPort1.Write(output, startIndex, packetLength);
95          }
96
97          private void buttonZeroDC_Click(object sender, EventArgs e)
98          // Sends packet to zero the DC motor position
99          {
100             output[commandIndex] = xZero;
101             output[MSBIndex] = 0;
102             output[LSBIndex] = 0;
103             output[escapeIndex] = 0;
104             serialPort1.Write(output, startIndex, packetLength);
105         }
106
107         private void buttonTransmitXY_Click(object sender, EventArgs e)
108         // Sends packets to move both DC and stepper motors from position ↪
              and velocity input
109         {
110             // Get position and velocity values from text boxes and       ↪
                  convert to useful values
111             double xLength = Kd * Convert.ToDouble(textBoxXPos.Text);
112             double yLength = Kd * Convert.ToDouble(textBoxYPos.Text);
113             double velocity = (vMax - vMin) / 100 * Convert.ToDouble      ↪
                  (textBoxVelocity.Text) + vMin;
114
115             // Split values into LSB and MSB
116             dcMSB = (Int32)xLength >> 8;
117             dcLSB = (Int32)xLength & 0xFF;
118             stepMSB = (Int32)yLength >> 8;
119             stepLSB = (Int32)yLength & 0xFF;
120             velMSB = (Int32)velocity >> 8;
121             velLSB = (Int32)velocity & 0xFF;
122
123             // Assign x-y control y transmit in command byte
124             output[commandIndex] = xyTransmitY;
125
```

```
126             // Check if either byte is 255 and assign escape byte accordingly
127             output[escapeIndex] = 0;
128             if (stepLSB == 255) { output[escapeIndex] = 1; stepLSB = 0; }
129             if (stepMSB == 255) { output[escapeIndex] += 2; stepMSB = 0; }
130
131             // Assign PWM bytes in buffer
132             output[MSBIndex] = (byte)stepMSB;
133             output[LSBIndex] = (byte)stepLSB;
134
135             // Write ytransmit packet to serial port
136             serialPort1.Write(output, startIndex, packetLength);
137
138             // Assign x-y transmit x transmit in command byte
139             output[commandIndex] = xyTransmitX;
140
141             // Check if either byte is 255 and assign escape byte accordingly
142             output[escapeIndex] = 0;
143             if (dcLSB == 255) { output[escapeIndex] = 1; dcLSB = 0; }
144             if (dcMSB == 255) { output[escapeIndex] += 2; dcMSB = 0; }
145
146             // Assign PWM bytes in buffer
147             output[MSBIndex] = (byte)dcMSB;
148             output[LSBIndex] = (byte)dcLSB;
149
150             // Sleep to avoid interrupting firmware
151             System.Threading.Thread.Sleep(300);
152
153             // Write xtransmit packet to serial port
154             serialPort1.Write(output, startIndex, packetLength);
155
156             // Assign x-y control velocity in command byte
157             output[commandIndex] = velPercent;
158
159             // Check if either byte is 255 and assign escape byte accordingly
160             output[escapeIndex] = 0;
161             if (velLSB == 255) { output[escapeIndex] = 1; velLSB = 0; }
162             if (velMSB == 255) { output[escapeIndex] += 2; velMSB = 0; }
163
164             // Assign PWM bytes in buffer
165             output[MSBIndex] = (byte)velMSB;
166             output[LSBIndex] = (byte)velLSB;
167
168             // Sleep to avoid interrupting firmware
169             System.Threading.Thread.Sleep(300);
170
171             // Write velocity packet to serial port
```

```
172             serialPort1.Write(output, startIndex, packetLength);
173         }
174
175         private void buttonTransmitY_Click(object sender, EventArgs e)
176         {
177             // Get position value from textbox and convert to LSB and MSB
178             double newLength = Kd * Convert.ToDouble(textBoxYPos.Text);
179             stepMSB = (Int32)newLength >> 8;
180             stepLSB = (Int32)newLength & 0xFF;
181
182             // Assign y-transmit in command byte
183             output[commandIndex] = yTransmit;
184
185             // Check if either byte is 255 and assign escape byte
                  accordingly
186             output[escapeIndex] = 0;
187             if (stepLSB == 255) { output[escapeIndex] = 1; stepLSB = 0; }
188             if (stepMSB == 255) { output[escapeIndex] += 2; stepMSB = 0; }
189
190             // Assign PWM bytes in buffer
191             output[MSBIndex] = (byte)stepMSB;
192             output[LSBIndex] = (byte)stepLSB;
193
194             // Write x-transmit packet to serial port
195             serialPort1.Write(output, startIndex, packetLength);
196         }
197
198         private void buttonTransmitX_Click(object sender, EventArgs e)
199         {
200             // Get position value from textbox and convert to LSB and MSB
201             double newLength = Kd * Convert.ToDouble(textBoxXPos.Text);
202             dcMSB = (Int32)newLength >> 8;
203             dcLSB = (Int32)newLength & 0xFF;
204
205             // Assign x-transmit in command byte
206             output[commandIndex] = xTransmit;
207
208             // Check if either byte is 255 and assign escape byte
                  accordingly
209             output[escapeIndex] = 0;
210             if (dcLSB == 255) { output[escapeIndex] = 1; dcLSB = 0; }
211             if (dcMSB == 255) { output[escapeIndex] += 2; dcMSB = 0; }
212
213             // Assign PWM bytes in buffer
214             output[MSBIndex] = (byte)dcMSB;
215             output[LSBIndex] = (byte)dcLSB;
216
217             // Write y-transmit to serial port
218             serialPort1.Write(output, startIndex, packetLength);
```

```
219         }
220
221         private void buttonClearChart_Click(object sender, EventArgs e)
222         // Clears the plot on the chart
223         {
224             timeCount = 0;
225             chartPosSpeed.Series["Position"].Points.Clear();
226             chartPosSpeed.Series["Speed"].Points.Clear();
227         }
228
229         private void timerWrite_Tick(object sender, EventArgs e)
230         // On timerWrite tick, detects if DC or Stepper motor speed has changed and writes the output packet to the serial port
231         {
232             if (motorSpeedChanged)
233             {
234                 serialPort1.Write(output, startIndex, packetLength);
235                 motorSpeedChanged = false;
236             }
237         }
238
239         private void timerRead_Tick(object sender, EventArgs e)
240         // On timerRead tick, dequeues dataQueue and sends dequeued bytes to the position and speed textboxes
241         {
242             // Misc. variables
243             int state = 0;
244             int MSB = 0;
245             int LSB = 0;
246             int instByte = 0;
247             double newCount;
248             double position;
249             double speed;
250             int nextByte;
251
252             // While TryDequeue from the dataQueue returns true
253             while (dataQueue.TryDequeue(out nextByte))
254             {
255                 // Check if 255 (start byte) and if so state = 1
256                 if (nextByte == 255)
257                 {
258                     state = 1;
259                 }
260                 // Check if state = 1 for instruction byte
261                 else if (state == 1)
262                 {
263                     instByte = nextByte;
264                     if (instByte == 0) { samplingPeriod = 200; }            // If instruction byte is zero, set
```

```
                    sampling period to 200ms
265                  else if (instByte == 1) { samplingPeriod =
                    20; }        // Else if instruction byte is one, set
                    sampling period to 20ms
266                  state =
                    2;                                          // Set
                    state = 2
267                 }
268                 // Check if state = 2 for MSB byte
269                 else if (state == 2)
270                 {
271                  MSB =
                    nextByte;                                    //
                    Assign MSB
272                  state =
                    3;                                          // Set
                    state = 3
273                 }
274                 // Check if state = 3 for LSB byte
275                 else if (state == 3)
276                 {
277                  LSB =
                    nextByte;                                    //
                    Assign LSB
278                  state =
                    4;                                          // Set
                    state = 4
279                 }
280                 // Check if state = 4 for final byte (escape byte)
281                 else if (state == 4)
282                 {
283                  if (nextByte % 2 != 0) { LSB =
                    255; }                    // Check if escape byte is odd
                    (1 or 3) and set LSB to 255
284                  if (nextByte > 1) { MSB =
                    255; }                          // Check if escape byte is
                     even (2) set MSB to 255
285
286                  // Combine LSB and MSB to get encoder counts and
                    multiply by 4 for quadrature signal
287                  newCount = (4 * ((MSB << 8) | LSB));
288
289                  // Calculate position in mm and speed in Hz
290                  position = (double)(newCount * toothPitch *
                    toothNumber) / (double)(motorCPR * gearRatio);       //
                    [mm]
291                  speed = 1000 * (double)(newCount - lastCount) /
                    (double)(samplingPeriod * motorCPR * gearRatio); // [Hz]
292
```

```
293                     // Assign DC position and speed textboxes
294                     textBoxDCPosition.Text = position.ToString();
295                     textBoxDCSpeedHz.Text = speed.ToString();
296                     textBoxDCSpeedRPM.Text = (60 * speed).ToString();
297
298                     // Assign chart values
299                     chartPosSpeed.Series["Position"].Points.AddXY
                    (timeCount, position);
300                     chartPosSpeed.Series["Speed"].Points.AddXY(timeCount,
                    60 * speed);
301
302                     // Check if save file checkbox is checked and if so
                    write the time and position to the outputFile
303                     if (checkBoxSave.Checked == true)
304                     {
305                         outputFile.Write(timeCount.ToString() + ", " +
                    position.ToString() + "\r\n");
306                     }
307
308                     // Set previous time and encoder count and set state
                    to 0
309                     prevTimeCount = timeCount;
310                     lastCount = newCount;
311                     state = 0;
312                 }
313             }
314         }
315
316
317         private void getOutputPacketArray()
318         // Takes values in packet textboxes in form and assigns them to
              the output packet array
319         {
320
321             output[startIndex] = Convert.ToByte(textBoxStart.Text);
322             output[commandIndex] = Convert.ToByte(textBoxCommand.Text);
323             output[MSBIndex] = Convert.ToByte(textBoxPWM1.Text);
324             output[LSBIndex] = Convert.ToByte(textBoxPWM2.Text);
325             output[escapeIndex] = Convert.ToByte(textBoxEscape.Text);
326         }
327
328         private void buttonConnect_Click(object sender, EventArgs e)
329         // Connects or disconnects serial port and sets baud rate from
              textbox. Also starts read and write timers
330         {
331             if (serialPort1.IsOpen == true)
332             {
333                 buttonConnect.Text = "Connect";
334                 serialPort1.Close();
```

```
335             }
336             else
337             {
338                 serialPort1.PortName = comboBoxCOMPorts.Text;
339                 buttonConnect.Text = "Disconnect";
340                 serialPort1.BaudRate = Convert.ToInt16(textBoxBaud.Text);
341                 serialPort1.Open();
342                 timerRead.Enabled = true;
343                 timerWrite.Enabled = true;
344             }
345         }
346 
347         private void buttonStopDC_Click(object sender, EventArgs e)
348         // Sends stop DC motor packet to serial port
349         {
350             output[commandIndex] = dcStop;
351             serialPort1.Write(output, startIndex, packetLength);
352             trackBarDCSpeed.Value = 0;
353         }
354 
355         private void buttonStopStepper_Click(object sender, EventArgs e)
356         // Sends stop stepper motor packet to serial port
357         {
358             output[commandIndex] = stepStop;
359             serialPort1.Write(output, startIndex, packetLength);
360             trackBarStepperSpeed.Value = 0;
361         }
362 
363         private void buttonStepCW_Click(object sender, EventArgs e)
364         // Sends CW step packet to serial port
365         {
366             output[commandIndex] = stepCW;
367             serialPort1.Write(output, startIndex, packetLength);
368         }
369         private void buttonStepCCW_Click(object sender, EventArgs e)
370         // Sends CCW step packet to serial port
371         {
372             output[commandIndex] = stepCCW;
373             serialPort1.Write(output, startIndex, packetLength);
374         }
375 
376         private void buttonTransmit_Click(object sender, EventArgs e)
377         // Assigns output array from packet textboxes and writes packet to ↵
               serial prot
378         {
379             if (serialPort1.IsOpen == true)
380             {
381                 getOutputPacketArray();
382                 serialPort1.Write(output, startIndex, packetLength);
```

```
383             }
384             else
385             {
386                 textBoxUserConsole.AppendText("Serial port is closed\r
                      \n");
387             }
388         }
389
390         private void serialPort1_DataReceived(object sender,
              System.IO.Ports.SerialDataReceivedEventArgs e)
391         // On data receive, gets new bytes from serial port and queues
              them in dataQueue
392         {
393             int newByte = 0;
394             int bytesToRead;
395             bytesToRead = serialPort1.BytesToRead;
396
397             while (bytesToRead != 0)
398             {
399                 newByte = serialPort1.ReadByte();           // Gets new
                      byte from serial port
400                 dataQueue.Enqueue(newByte);                 // Queues it
                      in dataQueue
401                 bytesToRead = serialPort1.BytesToRead;      // Checks for
                      more bytes
402             }
403         }
404
405         private void trackBarDCSpeed_ValueChanged(object sender, EventArgs
                e)
406         // Assigns command byte, PWM bytes, and escape byte from DC motor
              track bar when the track bar value changes
407         {
408             // Check direction
409             if (trackBarDCSpeed.Value > 0) { output[commandIndex] =
                  dcCW; }
410             else { output[commandIndex] = dcCCW; }
411
412             // Display speed
413             DCSpeed.Text = (100 * (double)trackBarDCSpeed.Value / (double)
                  trackBarDCSpeed.Maximum).ToString();
414
415             // Deadzone
416             if (Math.Abs(trackBarDCSpeed.Value) < dcDeadzone)
417             {
418                 dcLSB = 0;
419                 dcMSB = 0;
420             }
421             else
```

```
422             {
423                 // Take abs value and scale
424                 dcLSB = Math.Abs(trackBarDCSpeed.Value) & 0xFF;
425                 dcMSB = Math.Abs(trackBarDCSpeed.Value) >> 8;
426             }
427 
428             // Check if either byte is 255 and assign escape byte
                  accordingly
429             output[escapeIndex] = 0;
430             if (dcLSB == 255) { output[escapeIndex] = 1; dcLSB = 0; }
431             if (dcMSB == 255) { output[escapeIndex] += 2; dcMSB = 0; }
432 
433             // Assign PWM bytes in buffer
434             output[MSBIndex] = Convert.ToByte(dcMSB);
435             output[LSBIndex] = Convert.ToByte(dcLSB);
436 
437             // Flag motor speed changed
438             motorSpeedChanged = true;
439         }
440 
441         private void trackBarStepperSpeed_ValueChanged(object sender,
              EventArs e)
442         // Assigns command byte, PWM bytes, and escape byte from stepper
              motor track bar when the track bar value changes
443         {
444             // Check direction
445             if (trackBarStepperSpeed.Value > 0) { output[commandIndex] =
                  stepContCW; }
446             else { output[commandIndex] = stepContCCW; }
447 
448             // Display speed
449             StepperSpeed.Text = (100 * (double)
                  trackBarStepperSpeed.Value / (double)
                  trackBarStepperSpeed.Maximum).ToString();
450 
451             // Deadzone
452             if (Math.Abs(trackBarStepperSpeed.Value) < stepDeadzone)
453             {
454                 stepLSB = 0;
455                 stepMSB = 0;
456             }
457             else
458             {
459                 // Take abs value and scale
460                 stepLSB = Math.Abs(trackBarStepperSpeed.Value *
                      (stepTickMax - stepTick0) / trackBarStepperSpeed.Maximum
                       + stepTick0) & 0xFF;
461                 stepMSB = Math.Abs(trackBarStepperSpeed.Value *
                      (stepTickMax - stepTick0) / trackBarStepperSpeed.Maximum
```

```
                        + stepTick0) >> 8;
462             }
463
464             // Check if either byte is 255 and assign escape byte
                  accordingly
465             output[escapeIndex] = 0;
466             if (stepLSB == 255) { output[escapeIndex] = 1; }
467             if (stepMSB == 255) { output[escapeIndex] += 2; }
468
469             // Assign PWM bytes in buffer
470             output[MSBIndex] = Convert.ToByte(stepMSB);
471             output[LSBIndex] = Convert.ToByte(stepLSB);
472
473             // Flag motor speed changed
474             motorSpeedChanged = true;
475         }
476     }
477 }
478
```