

MECH 421 Lab 5: PCB Assembly and Stepper Motor Control

Student Name 1 Student Name 2

December 4, 2025

Abstract

This report details the assembly and testing of a custom Printed Circuit Board (PCB) for mechatronic applications, specifically focusing on stepper motor control. The lab is divided into three exercises: PCB assembly, single-axis stepper motor control, and dual-axis gantry control. The objective is to provide a comprehensive guide for future students to replicate the assembly and control logic without external references.

Contents

1	Introduction	3
2	Exercise 1: PCB Assembly and Soldering	3
2.1	Objective	3
2.2	Theory and Circuit Description	3
2.3	Parts List	3
2.4	Assembly Procedure	4
2.4.1	Step 1: Underside Resistors	4
2.4.2	Step 2: Power Supply	4
2.4.3	Step 3: USB Interface	4
2.4.4	Step 4: Microcontroller	5
2.4.5	Step 5: Motor Driver and Decoder	5
2.5	Results and Discussion	5
3	Exercise 2: Stepper Motor Control	6
3.1	Objective	6
3.2	Theory	6
3.2.1	Stepper Motor Operation	6
3.2.2	Current Control (PWM)	7
3.3	Firmware Implementation	7
3.3.1	Lookup Table	7
3.3.2	Step Generation Logic	7
3.4	Comparison with DC Motor Control	8

3.5	Software Interface (C#)	8
3.5.1	Communication Protocol	8
3.6	Results	9
4	Exercise 3: 2-Axis Control with Dual Stepper Motors	11
4.1	Objective	11
4.2	Procedure	11
4.2.1	Gantry Assembly	11
4.2.2	Wiring the Second Motor	11
4.2.3	Coordinated Motion Control (Firmware)	12
4.3	Software Interface Features	12
4.4	Image Processing Results & Critique	13
4.5	Results	13
5	Conclusion	15
A	Exercise 2 Code	16
A.1	Microcontroller Firmware	16
A.2	PC Interface (C#)	16
B	Exercise 3 Code	25
B.1	Microcontroller Firmware	25
B.2	PC Interface (C#)	32

1 Introduction

The purpose of this lab is to gain hands-on experience in PCB assembly, soldering surface-mount components, and implementing real-time control for stepper motors using a microcontroller. The final system is a 2-axis gantry capable of drawing complex shapes, demonstrating the integration of hardware (PCB, motors, mechanics) and software (firmware, PC interface).

2 Exercise 1: PCB Assembly and Soldering

2.1 Objective

The goal of this exercise is to populate a bare PCB with surface-mount and through-hole components to create a functional motor control board. This board includes a microcontroller (MSP430), USB interface (FTDI), power regulation, and motor drivers.

2.2 Theory and Circuit Description

The PCB consists of several key functional blocks:

- **Power Supply:** Converts external 12V DC input to 5V and 3.3V logic levels using linear regulators (NCP1117). 5V is used for the USB interface and some logic, while 3.3V powers the MSP430 microcontroller.
- **USB Interface:** Uses an FT230XS chip to convert USB signals from a PC into UART (Serial) signals compatible with the microcontroller. This allows for data logging and control commands.
- **Microcontroller:** The MSP430FR5739 is the brain of the board. It executes firmware to generate PWM signals for motors, read sensors, and communicate via UART.
- **Motor Driver:** The DRV8841 is a dual H-bridge driver capable of driving DC motors or bipolar stepper motors. It handles the high currents required by the motors, controlled by low-power logic signals from the MCU.

2.3 Parts List

The following components are required for assembly. Ensure all parts are accounted for before starting.

Component	Description	Qty
MSP430FR5739	Microcontroller (TSSOP-38)	1
FT230XS	USB-to-UART Bridge (SSOP-16)	1
DRV8841	Dual Motor Driver (HTSSOP-28)	1
NCP1117-3.3	3.3V Regulator (SOT-223)	1
NCP1117-5.0	5.0V Regulator (SOT-223)	1
Resistors	150 Ω , 330 Ω , 510 Ω , 1k Ω , 2.7k Ω , 30k Ω , 47k Ω	Kit
Capacitors	10pF, 51pF, 0.1 μ F, 0.47 μ F, 4.7 μ F, 10 μ F, 100 μ F	Kit
Connectors	USB Mini-B, DC Wall Jack, 0.1" Headers	Kit
LEDs	Green (1206)	6
Diodes	S34 Schottky (DO-214AC)	2
Crystal	24.00 MHz Resonator	1
Logic ICs	74HC14 Inverter, 74HC74 D-Latch	3

Table 1: Complete Components List for PCB Assembly

2.4 Assembly Procedure

The assembly was performed in the following sequence, testing each stage before proceeding:

2.4.1 Step 1: Underside Resistors

We began by soldering the 150 Ω current-limiting resistors on the bottom side. These 0603 components are small, so this served as good soldering practice.

- **Technique:** Tin one pad, place component with tweezers, reflow solder. Then solder the other side.

2.4.2 Step 2: Power Supply

The power supply section converts the 12V input to 5V and 3.3V rails.

- **Components:** DC Jack, S34 diodes (polarity critical), NCP1117 regulators, tantalum/ceramic capacitors.
- **Verification:** Connected 12V supply. Measured 5.01V and 3.29V at the output pins. The Power LED illuminated correctly.

2.4.3 Step 3: USB Interface

This section enables communication with the PC.

- **Components:** FT230XS (SSOP-16), Mini-USB connector, protection circuitry.
- **Challenge:** The fine pitch of the FT230XS pins led to a small solder bridge between pins 15 and 16.

- **Solution:** We used flux and desoldering braid to potential bridges.
- **Verification:** Plugged into PC; device recognized as "USB Serial Port (COM3)".

2.4.4 Step 4: Microcontroller

The core of the system.

- **Components:** MSP430FR5739 (TSSOP-38), 24MHz crystal oscillator, JTAG header.
- **Verification:** Flashed a test program to blink an LED using CCS. The successful programming confirmed the MCU and JTAG connections.

2.4.5 Step 5: Motor Driver and Decoder

Finally, the high-power motor drivers and encoder logic were added.

- **Components:** DRV8841 (HTSSOP-28) with thermal pad, 74HC14/74 connections.
- **Note:** Soldering the thermal pad through the via on the back was critical for heat dissipation.

2.5 Results and Discussion

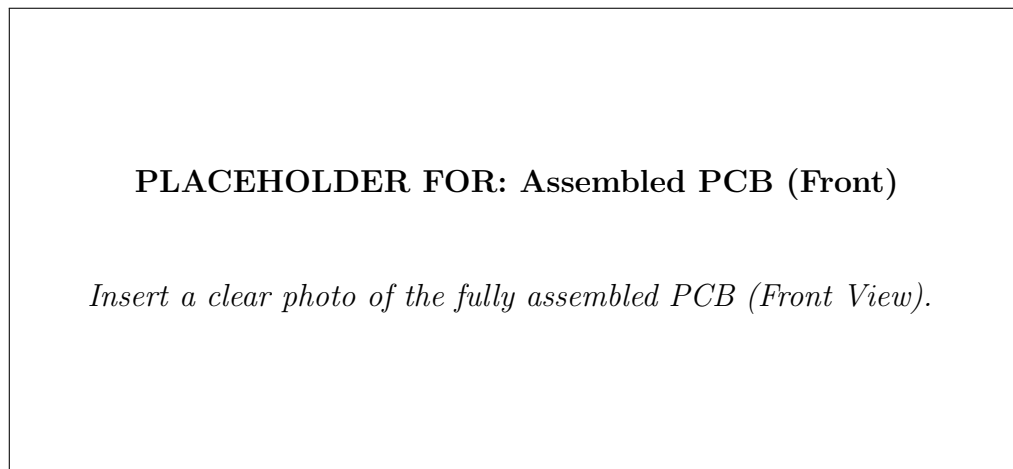


Figure 1: Assembled PCB (Front)

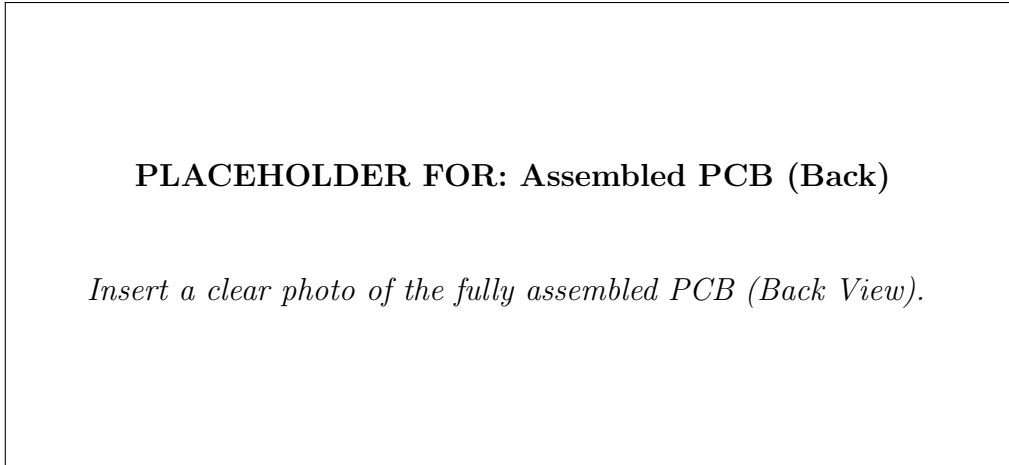


Figure 2: Assembled PCB (Back)

Challenges Encountered: Challenges Encountered: During the assembly of the USB interface, we encountered a communication failure where the PC did not recognize the board. A visual inspection using a magnifying glass revealed a solder bridge between pins 15 and 16 of the FT230XS chip. This was likely caused by using too much solder on the fine-pitch pads. We successfully removed the bridge using a desoldering braid and additional flux to reflow the joint. After cleaning the area with isopropyl alcohol, the device was correctly identified as a COM port. This experience highlighted the importance of using minimal solder and plenty of flux for SSOP packages.

3 Exercise 2: Stepper Motor Control

3.1 Objective

To control a bipolar stepper motor using the assembled PCB. This involves generating precise PWM waveforms to drive the motor phases in a specific sequence (half-stepping) and implementing a UART interface for velocity control.

3.2 Theory

3.2.1 Stepper Motor Operation

A bipolar stepper motor has two coils (Phase A and Phase B). By energizing these coils in a specific sequence, the rotor aligns with the magnetic field, moving in discrete steps.

- **Full-Stepping:** Energizing phases in sequence ($A+ \rightarrow B+ \rightarrow A- \rightarrow B-$).
- **Half-Stepping:** Inserting intermediate states where both phases are energized, doubling the resolution ($A+ \rightarrow A+B+ \rightarrow B+ \dots$).

3.2.2 Current Control (PWM)

To prevent overheating, the voltage applied to the coils is modulated using Pulse Width Modulation (PWM). A duty cycle of roughly 25% is sufficient for no-load operation.

3.3 Firmware Implementation

The firmware implements a lookup-based state machine to drive the stepper motor. To support variable speed, a Timer (TB0/TB1) is used to generate PWM signals for current control, while a periodic interrupt (Timer A1) advances the step state.

3.3.1 Lookup Table

The half-stepping sequence requires 8 distinct states. We used the following bitmask table where bits correspond to the phases A1, A2, B1, B2:

Stepper Step Table

```
1 // Half-step lookup table (8 steps)
2 // bit0=A1, bit1=A2, bit2=B1, bit3=B2
3 static const uint8_t stepper_table[8] = {
4     0b0001, // 1: A1
5     0b0101, // 2: A1+B1
6     0b0100, // 3: B1
7     0b0110, // 4: B1+A2
8     0b0010, // 5: A2
9     0b1010, // 6: A2+B2
10    0b1000, // 7: B2
11    0b1001 // 8: B2+A1
12 };
```

3.3.2 Step Generation Logic

The `step_motor1` function applies these masks to the Timer Capture Compare Registers (TB0CCR1/2, TB1CCR1/2) to set the PWM duty cycle for each connected pin.

```
1 void step_motor1(uint8_t step) {
2     uint8_t mask = stepper_table[step & 0x07];
3     // Update PWM Duty Cycles based on mask
4     TB0CCR2 = (mask & 0x01) ? PWM_DUTY : 0; // A1
5     TB0CCR1 = (mask & 0x02) ? PWM_DUTY : 0; // A2
6     TB1CCR2 = (mask & 0x04) ? PWM_DUTY : 0; // B1
7     TB1CCR1 = (mask & 0x08) ? PWM_DUTY : 0; // B2
8 }
```

Note: PWM duty cycle is set to 80% (approx) to manage current/heat while maintaining torque.

3.4 Comparison with DC Motor Control

While this lab focused on stepper motors, DC motors are an alternative for motion control. The key differences are:

- **Control Loop:** Stepper motors primarily operate in an open-loop configuration (command steps = assumed position). DC motors require a closed-loop system with feedback (e.g., an encoder) to control position or velocity accurately.
- **Torque:** Steppers have high holding torque at low speeds but drop off at high speeds. DC motors maintain torque better across the speed range but require continuous power to hold position against a load (unless geared).
- **Resolution:** Stepper resolution is fixed by the step angle (1.8°). DC motor resolution depends on the encoder PPR (Pulses Per Revolution).

To control a DC motor's position, a PID loop reading an encoder is required. The pseudocode for such a system would be:

```
1 // ISR triggered by Encoder Pulse
2 void Encoder_ISR() {
3     if (ChannelA == ChannelB) position++;
4     else position--;
5 }
6
7 // Control Loop (e.g., 100Hz Timer)
8 void Control_Loop() {
9     error = target_pos - position;
10    integral += error;
11    derivative = error - prev_error;
12
13    pwm_output = (Kp * error) + (Ki * integral) + (Kd * derivative);
14
15    set_motor_pwm(pwm_output);
16    prev_error = error;
17 }
```

Listing 1: DC Motor PID Pseudocode

3.5 Software Interface (C#)

The C# application acts as the commander. It sends packets to the MCU to set velocity or trigger single steps.

3.5.1 Communication Protocol

The packet structure is designed for reliability. Table 2 details the byte layout.

Byte Name	Pos	Description
Start Byte	1	Fixed at 255. synchronization marker.
Instruction Byte	2	1: CCW Continuous 2: CW Continuous 3/4: Single Step CCW/CW
Data H	3	High byte of Timer CCR0 value (Speed Control)
Data L	4	Low byte of Timer CCR0 value
Escape Byte	5	Bitmask for handling 255 in data: Bit 0: Data L was 255 Bit 1: Data H was 255

Table 2: UART Packet Structure

Code snippet for packet building (from `StepperCommander.cs`):

```

1 public byte[] BuildPacket(byte dirByte, ushort ccr0)
2 {
3     byte high = (byte)(ccr0 >> 8);
4     byte low = (byte)(ccr0 & 0xFF);
5     byte escape = 0;
6     // ... escape handling logic ...
7     return new[] { StartByte, dirByte, high, low, escape };
8 }

```

3.6 Results

PLACEHOLDER FOR: C Sharp GUI Screenshot

Insert a screenshot of the C# application controlling the motor.

Figure 3: C Sharp GUI Screenshot

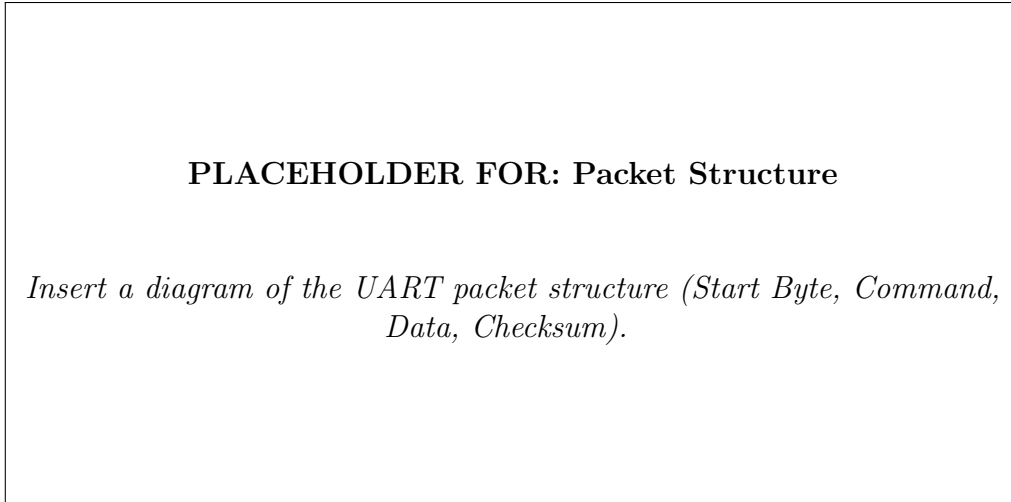


Figure 4: Packet Structure

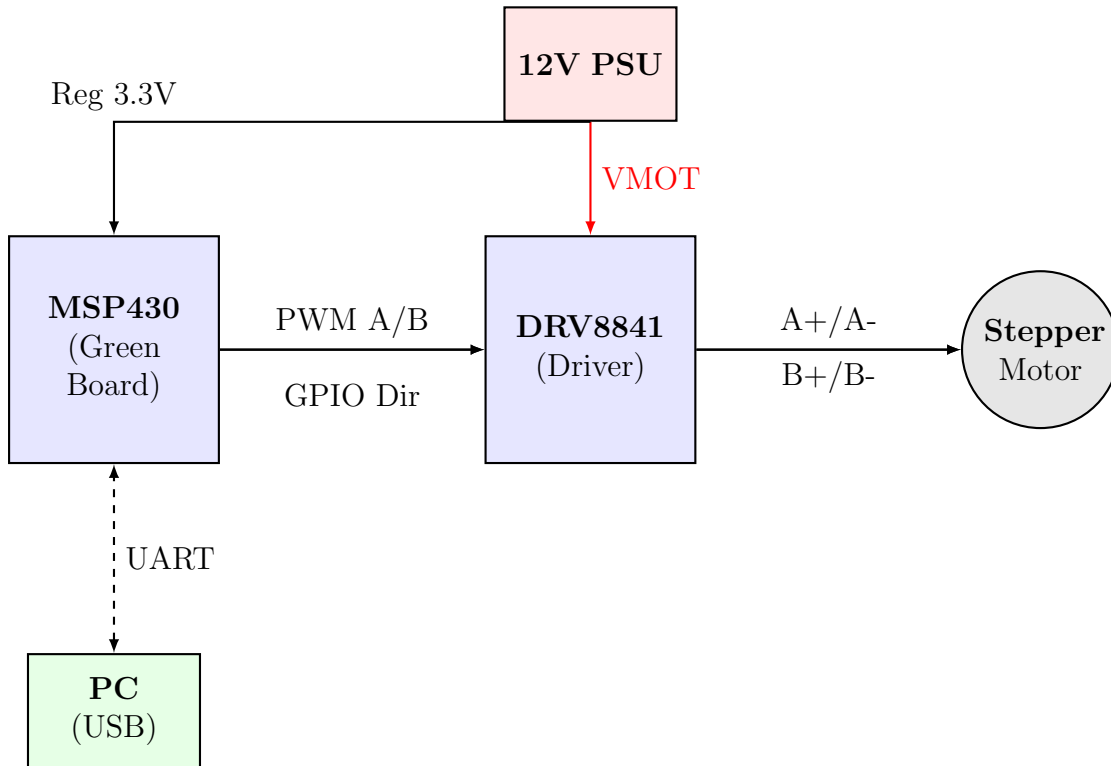


Figure 5: Electrical Schematic of Stepper Control System

Speed Measurements:

- **Max No-Load Speed:** [Value] steps/s
- **Max Loaded Speed:** [Value] steps/s

Discussion: The maximum speed is limited by the inductance of the motor coils, which opposes rapid current changes. As speed increases, the current doesn't have enough time to reach the target level within a step period, reducing torque until the motor stalls.

4 Exercise 3: 2-Axis Control with Dual Stepper Motors

4.1 Objective

To extend the system to 2 axes (X and Y) for a gantry stage. This requires driving a second stepper motor using an external H-bridge driver wired to the PCB.

4.2 Procedure

4.2.1 Gantry Assembly

The 2-axis gantry was assembled using V-slot aluminum extrusions and timing belts. The procedure followed the standard build provided in the lab manual, with the following key steps:

1. **Frame:** Assembled the base frame using corner brackets and T-nuts.
2. **X-Axis:** Mounted the X-axis carriage and belt drive.
3. **Y-Axis:** Mounted the Y-axis extrusion on top of the X-carriage.
4. **Pen Holder:** Uniquely, we designed and 3D printed a custom pen holder that attaches to the Y-axis slider. This holder uses an interference fit to secure a Sharpie marker, ensuring rigid contact with the paper during drawing operations.

4.2.2 Wiring the Second Motor

To enable Y-axis control, a second stepper driver (Pololu DRV8825) was wired to the system. Figure 6 illustrates the 2-axis connections.

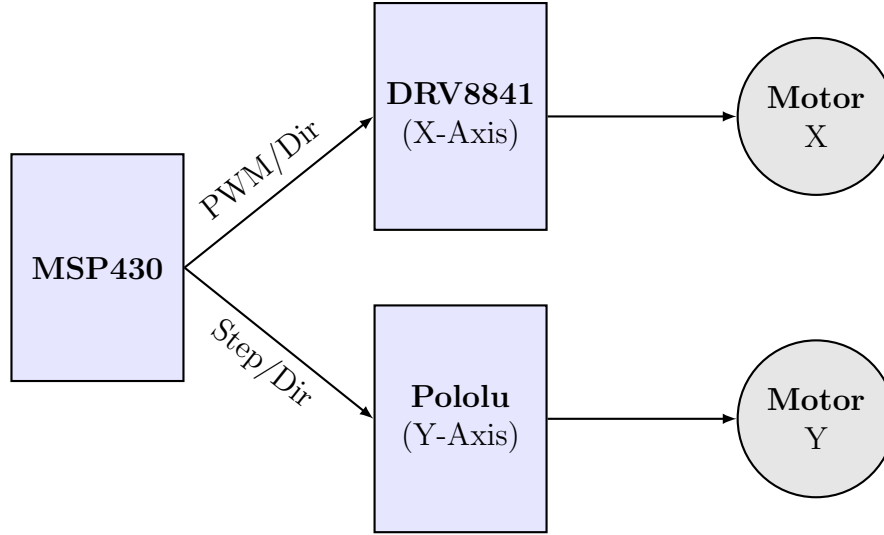


Figure 6: 2-Axis Control Wiring Diagram

4.2.3 Coordinated Motion Control (Firmware)

To draw straight lines and complex shapes, the X and Y axes must move in sync. We implemented a simplified Digital Differential Analyzer (DDA) algorithm in the firmware.

Firmware Logic: The ‘Timer_A1_ISR’ function serves as the central tick for motion.

1. **Accumulators:** We maintain ‘acc_x’ and ‘acc_y’.
2. **Step Decision:** At every interrupt, we add the target delta ($\Delta X, \Delta Y$) to the accumulators.
3. **Threshold:** If an accumulator exceeds ‘total_steps’, a step is issued to that motor, and ‘total_steps’ is subtracted.

This ensures that the ratio of X steps to Y steps is constant, producing a straight line.

```

1 // DDA Algorithm in Timer_A1_ISR
2 acc_x += delta_x;
3 acc_y += delta_y;
4
5 if (acc_x >= total_steps_needed) {
6     motor1_state = (motor1_state + step_x_inc) & 0x07;
7     step_motor1(motor1_state);
8     acc_x -= total_steps_needed;
9 }
10 // Repeat for Y...

```

4.3 Software Interface Features

The C# application provides a comprehensive dashboard:

- **Connection Panel:** Serial port selection and connect/disconnect.

- **Manual Control:** Slider for variable speed (1-100%), Buttons for single stepping.
- **Gantry Control:** Input fields for X/Y coordinates (cm) and "Move" button.
- **Image Processing:**
 - **Canvas:** Displays the loaded image and computed path.
 - **Import:** Loads JPG/PNG.
 - **Process:** Runs Sobel edge detection and Nearest Neighbor sorting.
 - **Draw:** Sends the point stream to the robot.

4.4 Image Processing Results & Critique

The "Image to Drawing" feature successfully identified edges and generated a path, but the physical result was mixed. The bitmaps were recognizable, but the drawing quality suffered due to several factors:

- **Z-Axis:** We lacked a Z-axis servo to lift the pen between strokes. This resulted in "travel lines" connecting distinct parts of the drawing, cluttering the image.
- **Precision:** The nearest-neighbor path optimization is greedy and often chose sub-optimal routes, increasing drawing time and error accumulation.
- **Mechanical:** The pen holder, while rigid, had no suspension. Variations in table height caused the pen to skip or dig into the paper.

Future Improvements:

1. Add a servo to lift the pen.
2. Implement microstepping (1/16 or 1/32) to smooth out the lines.
3. Use a vector-based approach (SVG) instead of raster edge detection for cleaner lines.

4.5 Results

We tested the gantry by drawing standard shapes (Square, Diamond, Triangle) and a complex "Creative Shape" (imported image).

PLACEHOLDER FOR: Standard Shapes Plot

Insert photo of the paper with the 6 test points/lines drawn.

Figure 7: Standard Shapes Plot

PLACEHOLDER FOR: Creative Shape Plot

Insert photo of the creative shape drawn by the gantry.

Figure 8: Creative Shape Plot

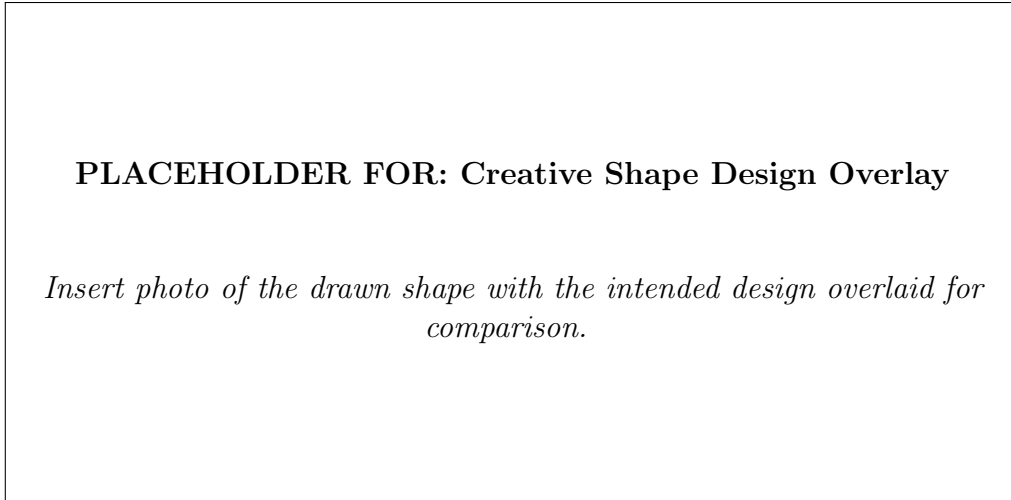


Figure 9: Creative Shape Design Overlay

Discussion on Accuracy: Deviations between the design and the result can be attributed to:

- **Backlash:** Play in the belt/pulley system caused circle endpoints to not perfectly meet.
- **Vibration:** At high speeds, the gantry frame resonance caused some line waviness.
- **Resolution:** The standard 200 step/rev motor + DDA rounding errors limit the finest detail to approx 0.2mm.

5 Conclusion

This lab successfully demonstrated the complete process of building a mechatronic controller, from soldering the PCB to implementing low-level motor control firmware and high-level PC software. The final 2-axis gantry system was able to draw complex shapes, validating the integration of all subsystems.

A Exercise 2 Code

A.1 Microcontroller Firmware

(Note: Exercise 2 functionality was integrated into the Exercise 3 firmware below.)

A.2 PC Interface (C#)

```
1 using System;
2 using System.IO.Ports;
3
4 namespace StepperControl
5 {
6     /// <summary>
7     /// Helper for building and sending stepper control packets and
8     tracking commanded angle.
9     /// </summary>
10    public class StepperCommander
11    {
12        private const byte StartByte = 255;
13        private const int TimerClockHz = 1_000_000;
14        public const int StepsPerRevolution = 400; // half-step resolution
15
16        private readonly SerialPort _port;
17
18        /// <summary>
19        /// Current commanded angle in degrees (0-360), updated by
20        MoveByAngle/MoveToAngle/CalibrateZero.
21        /// </summary>
22        public double CurrentAngleDeg { get; private set; }
23
24        /// <summary>
25        /// Create a commander that writes packets to the provided
26        SerialPort.
27        /// </summary>
28        /// <param name="port">Open SerialPort to write to.</param>
29        public StepperCommander(SerialPort port)
30        {
31            _port = port ?? throw new ArgumentNullException(nameof(port));
32        }
33
34        /// <summary>
35        /// Convert a desired continuous speed in RPM (sign = direction)
36        to a packet for dirByte 1/2.
37        /// Returns the packet and outputs dirByte and TA1CCR0 used.
38        /// </summary>
39        public byte[] BuildContinuousPacketFromRpm(double rpm, out ushort
40        ccr0, out byte dirByte)
41        {
42            dirByte = rpm >= 0 ? (byte)2 : (byte)1;
43            double stepsPerSecond = Math.Abs(rpm) * StepsPerRevolution /
44            60.0;
```

```

40      // Avoid divide-by-zero; clamp CCR0 into [1, 65535].
41      if (stepsPerSecond < 1e-6)
42      {
43          stepsPerSecond = 1e-6;
44      }
45
46      double rawCcr0 = (TimerClockHz / stepsPerSecond) - 1.0;
47      rawCcr0 = Math.Max(1.0, Math.Min(65535.0, rawCcr0));
48      ccr0 = (ushort)Math.Round(rawCcr0);
49
50      return BuildPacket(dirByte, ccr0);
51  }
52
53  /// <summary>
54  /// Build a single-step packet (dirByte 3 for CCW, 4 for CW).
55  /// CCR0 is zero and escape is zero for single steps.
56  /// </summary>
57  public byte[] BuildSingleStepPacket(bool clockwise)
58  {
59      byte dir = clockwise ? (byte)4 : (byte)3;
60      return new byte[] { StartByte, dir, 0, 0, 0 };
61  }
62
63  /// <summary>
64  /// Set the commanded zero reference angle to 0 degrees.
65  /// </summary>
66  public void CalibrateZero()
67  {
68      CurrentAngleDeg = 0;
69  }
70
71  /// <summary>
72  /// Move by a signed angle (deg). Positive = CW, negative = CCW.
73  /// Rounds to nearest half-step, sends that many single-step
74  packets, and wraps angle to [0, 360).
75  /// </summary>
76  public void MoveByAngle(double deltaDeg)
77  {
78      int steps = (int)Math.Round(deltaDeg * StepsPerRevolution /
360.0);
79      if (steps == 0)
80      {
81          return;
82      }
83
84      bool clockwise = steps > 0;
85      int count = Math.Abs(steps);
86      byte[] packet = BuildSingleStepPacket(clockwise);
87
88      for (int i = 0; i < count; i++)
89      {
90          SendPacket(packet);
91      }

```

```

92         double executedDeg = steps * (360.0 / StepsPerRevolution);
93         CurrentAngleDeg = NormalizeAngle(CurrentAngleDeg + executedDeg
94     );
95     }
96     /// <summary>
97     /// Move to an absolute target angle (deg). Uses the smallest
signed delta (-180, 180] then calls MoveByAngle.
98     /// </summary>
99     public void MoveToAngle(double targetDeg)
100     {
101         double targetNorm = NormalizeAngle(targetDeg);
102         double diff = targetNorm - CurrentAngleDeg;
103         diff = NormalizeSigned180(diff);
104         MoveByAngle(diff);
105     }
106
107     /// <summary>
108     /// Send a packet over the configured SerialPort. Throws if the
port is not open.
109     /// </summary>
110     public void SendPacket(byte[] packet)
111     {
112         if (packet == null) throw new ArgumentNullException(nameof(
packet));
113         if (_port == null || !_port.IsOpen)
114         {
115             throw new InvalidOperationException("Serial port is not
open.");
116         }
117         _port.Write(packet, 0, packet.Length);
118     }
119
120     private static double NormalizeAngle(double deg)
121     {
122         double wrapped = deg % 360.0;
123         if (wrapped < 0) wrapped += 360.0;
124         return wrapped;
125     }
126
127     private static double NormalizeSigned180(double deg)
128     {
129         double wrapped = deg % 360.0;
130         if (wrapped <= -180.0) wrapped += 360.0;
131         if (wrapped > 180.0) wrapped -= 360.0;
132         return wrapped;
133     }
134
135     private static byte[] BuildPacket(byte dirByte, ushort ccr0)
136     {
137         byte high = (byte)(ccr0 >> 8);
138         byte low = (byte)(ccr0 & 0xFF);
139         byte escape = 0;
140

```

```

141         if (high == 255)
142         {
143             high = 0;
144             escape |= 0b10;
145         }
146
147         if (low == 255)
148         {
149             low = 0;
150             escape |= 0b01;
151         }
152
153         return new[] { StartByte, dirByte, high, low, escape };
154     }
155 }
156 }

```

Listing 2: StepperCommander.cs

```

1 using System;
2 using System.Drawing;
3 using System.IO.Ports;
4 using System.Linq;
5 using System.Windows.Forms;
6
7 namespace StepperControl
8 {
9     public partial class Form1 : Form
10     {
11         private const byte StartByte = 255;
12         private const int StepTimerClockHz = 1_000_000; // SMCLK/8 in
13         firmware
14         private readonly double _stepAngleDeg = 360.0 / StepperCommander.
15         StepsPerRevolution;
16         private StepperCommander commander;
17
18         public Form1()
19         {
20             InitializeComponent();
21             DoubleBuffered = true;
22             velocityTrackBar.Minimum = -100;
23             velocityTrackBar.Maximum = 100;
24             velocityTrackBar.TickFrequency = 10;
25             velocityTrackBar.Value = 0;
26             maxSpeedNumeric.Value = 1200; // steps per second at slider =
27             100
28
29             PopulatePorts();
30             UpdateVelocityLabels(0, 0, 0);
31             UpdateModeAndFreq("Idle", 0);
32             SetTelemetry(null, null);
33
34             // Initialize Mode
35             SwitchMode(true);
36         }
37     }
38 }

```

```

33     private void continuousModeBtn_Click(object sender, EventArgs e)
34     {
35         SwitchMode(true);
36     }
37
38
39     private void singleStepModeBtn_Click(object sender, EventArgs e)
40     {
41         SwitchMode(false);
42         // Optional: Stop motor when switching to single step to
prevent confusion
43         StopMotor("Switching to Single Step Mode");
44     }
45
46     private void SwitchMode(bool continuous)
47     {
48         continuousPanel.Visible = continuous;
49         singleStepPanel.Visible = !continuous;
50
51         // Update button styles
52         if (continuous)
53         {
54             continuousModeBtn.BackColor = Color.FromArgb(50, 55, 70);
55             continuousModeBtn.ForeColor = Color.White;
56             singleStepModeBtn.BackColor = Color.Transparent;
57             singleStepModeBtn.ForeColor = Color.Gray;
58         }
59         else
60         {
61             singleStepModeBtn.BackColor = Color.FromArgb(50, 55, 70);
62             singleStepModeBtn.ForeColor = Color.White;
63             continuousModeBtn.BackColor = Color.Transparent;
64             continuousModeBtn.ForeColor = Color.Gray;
65         }
66     }
67
68     private void PopulatePorts()
69     {
70         portComboBox.Items.Clear();
71         var ports = SerialPort.GetPortNames().OrderBy(p => p).ToArray
72     );
73         if (ports.Length == 0)
74         {
75             portComboBox.Text = "No COM ports found";
76             connectButton.Enabled = false;
77             statusLabel.Text = "Connect board before opening the port.
";
78         }
79         else
80         {
81             portComboBox.Items.AddRange(ports);
82             portComboBox.SelectedIndex = 0;
83             connectButton.Enabled = true;
84             statusLabel.Text = "Select a COM port and press Connect.";

```

```

84     }
85 }
86
87 private void connectButton_Click(object sender, EventArgs e)
88 {
89     if (serialPort.IsOpen)
90     {
91         serialPort.Close();
92         commander = null;
93         connectButton.Text = "Connect";
94         statusLabel.Text = "Serial port closed.";
95         return;
96     }
97
98     if (portComboBox.SelectedItem == null)
99     {
100         MessageBox.Show("Select a COM port first.");
101         return;
102     }
103
104     serialPort.PortName = portComboBox.SelectedItem.ToString();
105     serialPort.BaudRate = 9600;
106     serialPort.Parity = Parity.None;
107     serialPort.StopBits = StopBits.One;
108     serialPort.DataBits = 8;
109
110     try
111     {
112         serialPort.Open();
113         commander = new StepperCommander(serialPort);
114         connectButton.Text = "Disconnect";
115         statusLabel.Text = $"Connected to {serialPort.PortName} at
116 {serialPort.BaudRate} baud.";
117     }
118     catch (Exception ex)
119     {
120         MessageBox.Show($"Failed to open {serialPort.PortName}: {
121 ex.Message}");
122     }
123 }
124
125 private void refreshButton_Click(object sender, EventArgs e)
126 {
127     PopulatePorts();
128 }
129
130 private void ccwStepButton_Click(object sender, EventArgs e)
131 {
132     if (!EnsureCommander()) return;
133     try
134     {
135         commander.MoveByAngle(-_stepAngleDeg);
136         UpdateModeAndFreq("Single CCW step", 0);
137         SetTelemetry(commander.CurrentAngleDeg, null);

```

```

136     }
137     catch (Exception ex)
138     {
139         MessageBox.Show($"Failed to step CCW: {ex.Message}");
140     }
141 }
142
143 private void cwStepButton_Click(object sender, EventArgs e)
144 {
145     if (!EnsureCommander()) return;
146     try
147     {
148         commander.MoveByAngle(_stepAngleDeg);
149         UpdateModeAndFreq("Single CW step", 0);
150         SetTelemetry(commander.CurrentAngleDeg, null);
151     }
152     catch (Exception ex)
153     {
154         MessageBox.Show($"Failed to step CW: {ex.Message}");
155     }
156 }
157
158 private void stopButton_Click(object sender, EventArgs e)
159 {
160     StopMotor("Stop (DirnByte 3 halts timer)");
161 }
162
163 private void velocityTrackBar_Scroll(object sender, EventArgs e)
164 {
165     HandleVelocityChange();
166 }
167
168 private void maxSpeedNumeric_ValueChanged(object sender, EventArgs
e)
169 {
170     HandleVelocityChange();
171 }
172
173 private void HandleVelocityChange()
174 {
175     int sliderValue = velocityTrackBar.Value;
176     double maxStepsPerSecond = (double)maxSpeedNumeric.Value;
177
178     if (sliderValue == 0)
179     {
180         UpdateVelocityLabels(0, 0, 0);
181         StopMotor("Slider at zero -> stop timer");
182         UpdateModeAndFreq("Stopped", 0);
183         SetTelemetry(commander?.CurrentAngleDeg, 0);
184         return;
185     }
186
187     double fraction = Math.Abs(sliderValue) / 100.0;
188     double commandedStepsPerSecond = Math.Max(1.0, fraction *

```

```

maxStepsPerSecond);
189         bool clockwise = sliderValue > 0;
190         double rpm = commandedStepsPerSecond * 60.0 / StepperCommander
.StepsPerRevolution;
191         if (!clockwise) rpm *= -1;
192
193         if (!EnsureCommander()) return;
194
195         try
196         {
197             byte dir;
198             ushort ccr0;
199             byte[] packet = commander.BuildContinuousPacketFromRpm(rpm
, out ccr0, out dir);
200             commander.SendPacket(packet);
201
202             UpdateVelocityLabels(sliderValue, commandedStepsPerSecond,
ccr0);
203             UpdateModeAndFreq($"Continuous {(clockwise ? "CW" : "CCW")
}", ccr0);
204             SetTelemetry(commander.CurrentAngleDeg,
commandedStepsPerSecond);
205             statusLabel.Text = $"Continuous {(clockwise ? "CW" : "CCW")
}} | {commandedStepsPerSecond:F1} steps/s | CCR0 {ccr0}";
206             lastPacketLabel.Text = $"Last Packet [{string.Join(", ",
packet.Select(b => b.ToString()))}]";
207         }
208         catch (Exception ex)
209         {
210             statusLabel.Text = "Write failed.";
211             MessageBox.Show($"Failed to send packet: {ex.Message}");
212         }
213     }
214
215     private void StopMotor(string reason)
216     {
217         if (velocityTrackBar.Value != 0)
218         {
219             velocityTrackBar.Value = 0;
220         }
221
222         if (commander != null && serialPort.IsOpen)
223         {
224             byte[] packet = { StartByte, 3, 0, 0, 0 };
225             try
226             {
227                 commander.SendPacket(packet);
228                 lastPacketLabel.Text = $"Last Packet [{string.Join(",
", packet.Select(b => b.ToString()))}]";
229             }
230             catch (Exception ex)
231             {
232                 MessageBox.Show($"Failed to stop motor: {ex.Message}");
233
;

```

```

233         }
234     }
235
236     UpdateVelocityLabels(0, 0, 0);
237     UpdateModeAndFreq("Stopped", 0);
238     SetTelemetry(commander?.CurrentAngleDeg, 0);
239 }
240
241 private void UpdateVelocityLabels(int sliderValue, double
stepsPerSecond, ushort ccr0)
242 {
243     string direction = sliderValue > 0 ? "CW" : sliderValue < 0 ?
"CCW" : "Stopped";
244     velocitySummaryLabel.Text =
245         $"Slider {sliderValue} -> {direction} | {stepsPerSecond:F1
} steps/s | TA1CCR0 {ccr0}";
246 }
247
248 private void UpdateModeAndFreq(string modeText, ushort ccr0)
249 {
250     modeLabel.Text = $"Mode: {modeText}";
251     double freq = ccr0 == 0 ? 0 : StepTimerClockHz / (ccr0 + 1.0);
252     freqLabel.Text = $"Step freq: {freq:F1} Hz (TA1CCR0 {ccr0})";
253 }
254
255 private void SetTelemetry(double? positionDeg, double?
velocityStepsPerSecond)
256 {
257     positionValueTextBox.Text = positionDeg.HasValue ? $"{{
NormalizeDeg(positionDeg.Value):F2} deg" : "N/A";
258
259     if (velocityStepsPerSecond.HasValue)
260     {
261         double hz = velocityStepsPerSecond.Value;
262         double rpm = hz * 60.0 / StepperCommander.
StepsPerRevolution;
263         velocityHzTextBox.Text = $"{{hz:F1}}";
264         velocityRpmTextBox.Text = $"{{rpm:F2}}";
265     }
266     else
267     {
268         velocityHzTextBox.Text = "N/A";
269         velocityRpmTextBox.Text = "N/A";
270     }
271 }
272
273 private void Form1_FormClosing(object sender, FormClosingEventArgs
e)
274 {
275     if (serialPort.IsOpen)
276     {
277         serialPort.Close();
278     }
279 }

```

```

280
281     private bool EnsureCommander()
282     {
283         if (commander == null || !serialPort.IsOpen)
284         {
285             statusLabel.Text = "Connect to the board first.";
286             return false;
287         }
288
289         return true;
290     }
291
292     private static double NormalizeDeg(double deg)
293     {
294         double wrapped = deg % 360.0;
295         if (wrapped < 0) wrapped += 360.0;
296         return wrapped;
297     }
298 }
299 }

```

Listing 3: Form1.cs

B Exercise 3 Code

B.1 Microcontroller Firmware

```

1 #include <msp430.h>
2 #include <stdint.h>
3 #include <stdlib.h>
4
5 // ===== DEFINES =====
6 #define QUEUE_SIZE 50
7
8 // Motor 1 (X-Axis) Pins - Using Timer B0/B1 PWM logic from example
9 // A1: P1.5 (TB0.2), A2: P1.4 (TB0.1)
10 // B1: P3.5 (TB1.2), B2: P3.4 (TB1.1)
11 // Note: The example used PWM CCRs to drive these. We will replicate that.
12
13 // Motor 2 (Y-Axis) Pins - New Driver
14 // PWMA, PWMB: P1.3 (Shared Enable/PWM)
15 // AIN2: P2.0
16 // AIN1: P2.1
17 // BIN1: P3.2
18 // BIN2: P3.3
19 #define M2_PWM_PIN BIT3 // P1.3
20 #define M2_AIN2 BIT0 // PJ.0
21 #define M2_AIN1 BIT1 // PJ.1
22 #define M2_BIN1 BIT2 // PJ.2
23 #define M2_BIN2 BIT3 // PJ.3
24
25 // PWM Settings (8MHz Clock)
26 // 8kHz: Period=1000

```

```

27 // 20kHz: Period=400 (Silent)
28 #define PWM_PERIOD 400
29 #define PWM_DUTY    (PWM_PERIOD * 80 / 100)
30
31 // Stepper Constants
32 #define STEPS_PER_REV 200 // Standard 1.8 deg stepper
33 #define MICROSTEPS 1      // Full stepping for now
34 // Assuming some calibration: Steps per CM.
35 // Let's assume 1 rev = 4 cm travel (example). -> 50 steps/cm.
36 // User can adjust this calibration in C# or here.
37 // For now, we will receive "Steps" directly from C# to keep firmware
    simple,
38 // OR we receive CM and convert. The prompt says "takes a relative [X,Y]
    distance".
39 // It's better to do the math in C# and send raw steps to MCU.
40 // So Packet will contain STEPS.
41
42 // ===== GLOBALS =====
43 volatile unsigned char queue[QUEUE_SIZE];
44 volatile unsigned int front = 0;
45 volatile unsigned int numItems = 0;
46
47 // Packet Buffer
48 volatile unsigned char startByte;
49 volatile unsigned char cmdByte; // 1=Move, 2=Stop
50 volatile unsigned char xH, xL, yH, yL;
51 volatile unsigned char velByte;
52 volatile unsigned char escapeByte;
53
54 // Motion Control Globals
55 volatile int32_t target_x_steps = 0;
56 volatile int32_t target_y_steps = 0;
57 volatile int32_t current_x_steps = 0;
58 volatile int32_t current_y_steps = 0;
59
60 volatile int32_t delta_x = 0;
61 volatile int32_t delta_y = 0;
62 volatile int32_t step_x_inc = 0;
63 volatile int32_t step_y_inc = 0;
64 volatile int32_t error_term = 0;
65 volatile uint32_t total_steps_needed = 0;
66 volatile uint32_t steps_taken = 0;
67
68 volatile uint8_t motor1_state = 0;
69 volatile uint8_t motor2_state = 0;
70 volatile uint8_t is_moving = 0;
71
72 // Half-step lookup table (8 steps)
73 // bit0=A1, bit1=A2, bit2=B1, bit3=B2
74 static const uint8_t stepper_table[8] = {
75     0b0001, // 1: A1
76     0b0101, // 2: A1+B1
77     0b0100, // 3: B1
78     0b0110, // 4: B1+A2

```

```

79     0b0010, // 5: A2
80     0b1010, // 6: A2+B2
81     0b1000, // 7: B2
82     0b1001  // 8: B2+A1
83 };
84
85 // ===== PROTOTYPES =====
86 void clockSetup(void);
87 void gpioSetup(void);
88 void timerSetup(void);
89 void uartSetup(void);
90 void step_motor1(uint8_t step);
91 void step_motor2(uint8_t step);
92 void process_packet(void);
93 void start_move(int16_t dx, int16_t dy, uint8_t speed);
94
95 // ===== SETUP FUNCTIONS =====
96 void clockSetup(void) {
97     CSCTL0_H = CSKEY_H;
98     CSCTL1 = DCOFSEL_3; // 8 MHz
99     CSCTL2 = SELS__DCOCLK | SELA__DCOCLK | SELM__DCOCLK;
100    CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;
101    CSCTL0_H = 0;
102 }
103
104 void gpioSetup(void) {
105     // Motor 1 (Existing) - PWM pins handled in Timer Setup, but we need
    to ensure directions
106     // P1.4, P1.5, P3.4, P3.5 are used by Timer B0/B1 in the example.
107     // We will stick to the example's method of using TBxCCRx for Motor 1.
108
109     // Motor 2 (New) - GPIO Control
110     // P1.3 (PWM/Enable) -> Output, High
111     P1DIR |= M2_PWM_PIN;
112     P1OUT |= M2_PWM_PIN; // Enable driver
113
114     // Port J (PJ.0 - PJ.3) for Motor 2 Coils
115     // Note: PJ is often shared with JTAG. Ensure JTAG is not interfering
    if debugging.
116     PJDIR |= M2_AIN2 | M2_AIN1 | M2_BIN1 | M2_BIN2;
117     PJOUT &= ~(M2_AIN2 | M2_AIN1 | M2_BIN1 | M2_BIN2);
118     // Clear SEL bits to ensure GPIO mode if necessary (though usually
    default is GPIO for PJ on some devices, check datasheet)
119     // On FR5739, PJ.0-3 are JTAG. To use as GPIO, we might need to be
    careful.
120     // Assuming user has this working or knows the setup.
121     PJSEL0 &= ~(M2_AIN2 | M2_AIN1 | M2_BIN1 | M2_BIN2);
122     PJSEL1 &= ~(M2_AIN2 | M2_AIN1 | M2_BIN1 | M2_BIN2);
123
124     // Unlock GPIO
125     PM5CTL0 &= ~LOCKLPM5;
126 }
127
128 void timerSetup(void) {

```

```

129 // --- Timer A1: Step Clock ---
130 // We will use TA1 for the motion tick.
131 // Frequency will be set dynamically based on speed.
132 TA1CTL = TASSEL__SMCLK | MC__STOP | TACLRL;
133 TA1CCTL0 = CCIE;
134
135 // --- Timer B0/B1: Motor 1 PWM Generation (from example) ---
136 // TB0: P1.4(A2), P1.5(A1)
137 TB0CCR0 = PWM_PERIOD - 1;
138 TB0CCTL1 = OUTMOD_7; // Reset/Set
139 TB0CCTL2 = OUTMOD_7;
140 TB0CTL = TBSSEL__SMCLK | MC__UP | TBCLR;
141
142 // TB1: P3.4(B2), P3.5(B1)
143 TB1CCR0 = PWM_PERIOD - 1;
144 TB1CCTL1 = OUTMOD_7;
145 TB1CCTL2 = OUTMOD_7;
146 TB1CTL = TBSSEL__SMCLK | MC__UP | TBCLR;
147
148 // Set Pins for TB0/TB1
149 P1DIR |= BIT4 | BIT5;
150 P1SEL0 |= BIT4 | BIT5;
151 P1SEL1 &= ~(BIT4 | BIT5);
152
153 P3DIR |= BIT4 | BIT5;
154 P3SEL0 |= BIT4 | BIT5;
155 P3SEL1 &= ~(BIT4 | BIT5);
156 }
157
158 void uartSetup(void) {
159 // P2.5=RX, P2.6=TX
160 P2SEL1 |= BIT5 | BIT6;
161 P2SEL0 &= ~(BIT5 | BIT6);
162
163 UCA1CTLW0 |= UCSWRST;
164 UCA1CTLW0 |= UCSSEL__SMCLK; // Use SMCLK (8MHz)
165 // 9600 Baud from 8MHz
166 // N = 8000000/9600 = 833.33
167 // UCBRx = 52, UCBRFx = 1, UCBRSx = 0x49 (from example)
168 // Wait, example used ACLK=8MHz. We set SMCLK=8MHz. Same difference.
169 UCA1MCTLW = UCOS16 | 0x4900 | 0x0010; // UCBRF=1
170 UCA1BRW = 52;
171
172 UCA1CTLW0 &= ~UCSWRST;
173 UCA1IE |= UCRXIE;
174 }
175
176 // ===== MOTOR CONTROL =====
177 void step_motor1(uint8_t step) {
178 // Motor 1 uses Timer PWMs (TB0, TB1)
179 uint8_t mask = stepper_table[step & 0x07];
180
181 // A1 (TB0.2)
182 TB0CCR2 = (mask & 0x01) ? PWM_DUTY : 0;

```

```

183 // A2 (TB0.1)
184 TB0CCR1 = (mask & 0x02) ? PWM_DUTY : 0;
185 // B1 (TB1.2)
186 TB1CCR2 = (mask & 0x04) ? PWM_DUTY : 0;
187 // B2 (TB1.1)
188 TB1CCR1 = (mask & 0x08) ? PWM_DUTY : 0;
189 }
190
191 void step_motor2(uint8_t step) {
192 // Motor 2 uses GPIOs directly
193 uint8_t mask = stepper_table[step & 0x07];
194
195 // A1 (AIN2?) - Let's map bit0->AIN2, bit1->AIN1
196 // Actually, let's just map logically.
197 // Coil A: AIN1, AIN2. Coil B: BIN1, BIN2.
198 // Table: bit0=A1, bit1=A2, bit2=B1, bit3=B2
199
200 // A1 (AIN1)
201 if (mask & 0x01) PJOUT |= M2_AIN1; else PJOUT &= ~M2_AIN1;
202 // A2 (AIN2)
203 if (mask & 0x02) PJOUT |= M2_AIN2; else PJOUT &= ~M2_AIN2;
204
205 // B1 (BIN1)
206 if (mask & 0x04) PJOUT |= M2_BIN1; else PJOUT &= ~M2_BIN1;
207 // B2 (BIN2)
208 if (mask & 0x08) PJOUT |= M2_BIN2; else PJOUT &= ~M2_BIN2;
209 }
210
211 void start_move(int16_t dx, int16_t dy, uint8_t speed) {
212 // Disable interrupt to setup
213 TA1CTL &= ~MC_3;
214
215 delta_x = dx;
216 delta_y = dy;
217
218 step_x_inc = (dx > 0) ? 1 : -1;
219 step_y_inc = (dy > 0) ? 1 : -1;
220
221 delta_x = abs(delta_x);
222 delta_y = abs(delta_y);
223
224 total_steps_needed = (delta_x > delta_y) ? delta_x : delta_y;
225 steps_taken = 0;
226
227 // Initialize error term for Bresenham's
228 // We will use a simplified DDA:
229 // We have a major axis and a minor axis.
230 // But actually, we can just use floating point logic scaled up, or
231 // just standard Bresenham.
232 // Let's use a counter approach for both.
233 // We want to complete 'total_steps_needed' ticks.
234 // On each tick:
235 // AccumulatorX += delta_x
236 // if AccumulatorX >= total_steps_needed: step X, AccumulatorX -=

```

```

236     total_steps_needed
237     // AccumulatorY += delta_y
238     // if AccumulatorY >= total_steps_needed: step Y, AccumulatorY -=
239     total_steps_needed
240
241     // Reset accumulators
242     // We'll use static vars in ISR or globals.
243     // Let's use globals 'current_x_steps' as accumulator for this move?
244     No, that tracks position.
245     // Let's add new globals for accumulators.
246
247     // Speed: 100% = Max Speed.
248     // Timer CCR0 = Base / Speed.
249     // Base 8MHz / 8 = 1MHz.
250     // Max speed (100%) -> 1kHz stepping?
251     // CCR0 = 1000 -> 1kHz.
252     // If speed is 100, CCR0 = 1000.
253     // If speed is 10, CCR0 = 10000.
254     // Formula: CCR0 = 100000 / speed (if speed 1..100)
255
256     // Speed is now pre-scaled in C# (1-12% range)
257     if (speed == 0) speed = 1;
258     TA1CCR0 = 40000 / speed; // Adjust constant to tune max speed
259
260     is_moving = 1;
261     TA1CTL |= MC_UP; // Start Timer
262 }
263
264 // Globals for DDA
265 volatile int32_t acc_x = 0;
266 volatile int32_t acc_y = 0;
267
268 // ===== MAIN =====
269 int main(void) {
270     WDTCTL = WDTPW | WDTHOLD;
271     clockSetup();
272     gpioSetup();
273     timerSetup();
274     uartSetup();
275
276     __enable_interrupt();
277
278     while (1) {
279         if (numItems >= 8) {
280             process_packet();
281         }
282     }
283 }
284
285 // ===== PACKET PROCESSING =====
286 void process_packet(void) {
287     // Packet: [255][CMD][XH][XL][YH][YL][VEL][ESC]
288     // Check start byte
289     if (queue[front] != 255) {

```

```

287     // Invalid start, consume one byte
288     front = (front + 1) % QUEUE_SIZE;
289     numItems--;
290     return;
291 }
292
293 // Extract bytes
294 unsigned char pkt[8];
295 int i;
296 for (i = 0; i < 8; i++) {
297     pkt[i] = queue[(front + i) % QUEUE_SIZE];
298 }
299
300 // Handle Escape
301 unsigned char esc = pkt[7];
302 if (esc & 0x01) pkt[6] = 255; // Vel
303 if (esc & 0x02) pkt[5] = 255; // YL
304 if (esc & 0x04) pkt[4] = 255; // YH
305 if (esc & 0x08) pkt[3] = 255; // XL
306 if (esc & 0x10) pkt[2] = 255; // XH
307 // CMD usually doesn't need escape if it's small enum
308
309 int16_t dx = (int16_t)((pkt[2] << 8) | pkt[3]);
310 int16_t dy = (int16_t)((pkt[4] << 8) | pkt[5]);
311 uint8_t vel = pkt[6];
312 uint8_t cmd = pkt[1];
313
314 // Consume queue
315 front = (front + 8) % QUEUE_SIZE;
316 numItems -= 8;
317
318 if (cmd == 1) { // MOVE
319     start_move(dx, dy, vel);
320 } else if (cmd == 2) { // STOP
321     TA1CTL &= ~MC_3;
322     is_moving = 0;
323 }
324 }
325
326 // ===== ISRs =====
327 #pragma vector = USCI_A1_VECTOR
328 __interrupt void USCI_A1_ISR(void) {
329     if (UCA1IFG & UCRXIFG) {
330         unsigned char rx = UCA1RXBUF;
331         if (numItems < QUEUE_SIZE) {
332             queue[(front + numItems) % QUEUE_SIZE] = rx;
333             numItems++;
334         }
335     }
336 }
337
338 #pragma vector = TIMER1_A0_VECTOR
339 __interrupt void Timer_A1_ISR(void) {
340     if (!is_moving) return;

```

```

341
342 // DDA Algorithm
343 // We step the dominant axis? No, we step based on accumulators.
344 // Actually, to ensure straight line, we should use the 'total_steps'
    approach.
345
346 // Add delta to accumulators
347 acc_x += delta_x;
348 acc_y += delta_y;
349
350 if (acc_x >= total_steps_needed) {
351     motor1_state = (motor1_state + step_x_inc) & 0x07;
352     step_motor1(motor1_state);
353     acc_x -= total_steps_needed;
354 }
355
356 if (acc_y >= total_steps_needed) {
357     motor2_state = (motor2_state + step_y_inc) & 0x07;
358     step_motor2(motor2_state);
359     acc_y -= total_steps_needed;
360 }
361
362 steps_taken++;
363 if (steps_taken >= total_steps_needed) {
364     is_moving = 0;
365     TA1CTL &= ~MC_3; // Stop timer
366     // Reset accumulators for next time (though start_move does this)
367     acc_x = 0;
368     acc_y = 0;
369 }
370 }

```

Listing 4: main.c

B.2 PC Interface (C#)

```

1 using System;
2 using System.IO.Ports;
3 using System.Threading;
4 using System.Threading.Tasks;
5 using System.Windows;
6 using System.Windows.Controls;
7 using System.Windows.Media;
8 using System.Windows.Media.Imaging;
9 using System.Collections.Generic;
10 using System.Linq;
11 using Microsoft.Win32;
12
13 namespace GantryControl
14 {
15     public partial class MainWindow : Window
16     {
17         SerialPort _serialPort;
18         // Calibration: Steps per CM

```

```

19 // Motor 1 (Physical X, now UI Y): 100 steps/cm
20 // Motor 2 (Physical Y, now UI X): 125 steps/cm
21 const double STEPS_PER_CM_M1 = 100.0;
22 const double STEPS_PER_CM_M2 = 125.0;
23
24 // Position Tracking (Relative to "Center")
25 double _currentX = 0;
26 double _currentY = 0;
27
28 // Speeds
29 const int TRACING_SPEED = 1;
30 const int RETURN_SPEED = 1;
31
32 private bool _isUpdatingVelocity = false;
33 private volatile bool _stopRequested = false;
34
35 public MainWindow()
36 {
37     InitializeComponent();
38     LoadPorts();
39 }
40
41 private void VelSlider_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
42 {
43     if (_isUpdatingVelocity) return;
44     _isUpdatingVelocity = true;
45     if (VelInput != null)
46     {
47         VelInput.Text = ((int)VelSlider.Value).ToString();
48     }
49     _isUpdatingVelocity = false;
50 }
51
52 private void VelInput_TextChanged(object sender,
TextChangedEventArgs e)
53 {
54     if (_isUpdatingVelocity) return;
55     _isUpdatingVelocity = true;
56     if (int.TryParse(VelInput.Text, out int value))
57     {
58         if (value < 1) value = 1;
59         if (value > 100) value = 100;
60         VelSlider.Value = value;
61     }
62     _isUpdatingVelocity = false;
63 }
64
65
66
67 private void LoadPorts()
68 {
69     PortSelector.ItemsSource = SerialPort.GetPortNames();
70     if (PortSelector.Items.Count > 0) PortSelector.SelectedIndex =

```

```

0;
71     }
72
73     private void ConnectBtn_Click(object sender, RoutedEventArgs e)
74     {
75         if (_serialPort != null && _serialPort.IsOpen)
76         {
77             _serialPort.Close();
78             ConnectBtn.Content = "Connect";
79             StatusText.Text = "Disconnected";
80         }
81         else
82         {
83             try
84             {
85                 _serialPort = new SerialPort(PortSelector.SelectedItem
.ToToString(), 9600, Parity.None, 8, StopBits.One);
86                 _serialPort.Open();
87                 ConnectBtn.Content = "Disconnect";
88                 StatusText.Text = "Connected";
89             }
90             catch (Exception ex)
91             {
92                 MessageBox.Show("Error: " + ex.Message);
93             }
94         }
95     }
96
97     private void SendPacket(int dxSteps, int dySteps, int velocity)
98     {
99         if (_serialPort == null || !_serialPort.IsOpen) return;
100
101         // Packet Structure: [255][CMD][XH][XL][YH][YL][VEL][ESC]
102         byte[] packet = new byte[8];
103         packet[0] = 255;
104         packet[1] = 1; // Move Command
105
106         // Convert 16-bit signed to bytes
107         byte xh = (byte)((dxSteps >> 8) & 0xFF);
108         byte xl = (byte)(dxSteps & 0xFF);
109         byte yh = (byte)((dySteps >> 8) & 0xFF);
110         byte yl = (byte)(dySteps & 0xFF);
111         byte vel = (byte)velocity;
112
113         // Escape byte logic
114         byte esc = 0;
115         if (vel == 255) { esc |= 0x01; vel = 0; }
116         if (yl == 255) { esc |= 0x02; yl = 0; }
117         if (yh == 255) { esc |= 0x04; yh = 0; }
118         if (xl == 255) { esc |= 0x08; xl = 0; }
119         if (xh == 255) { esc |= 0x10; xh = 0; }
120
121         packet[2] = xh;
122         packet[3] = xl;

```

```

123         packet[4] = yh;
124         packet[5] = yl;
125         packet[6] = vel;
126         packet[7] = esc;
127
128         _serialPort.Write(packet, 0, 8);
129
130         // Update Position Tracking
131         // dxSteps was sent to Motor 1 (Physical X / UI Y) -> Wait,
let's check MoveBtn_Click mapping
132         // In MoveBtn_Click:
133         // dx (Motor 1) = yCm * STEPS_PER_CM_M1
134         // dy (Motor 2) = xCm * STEPS_PER_CM_M2
135         // So dxSteps corresponds to UI Y, dySteps corresponds to UI X
.
136
137         double movedY = dxSteps / STEPS_PER_CM_M1;
138         double movedX = dySteps / STEPS_PER_CM_M2;
139
140         _currentX += movedX;
141         _currentY += movedY;
142     }
143
144     private void MoveBtn_Click(object sender, RoutedEventArgs e)
145     {
146         if (double.TryParse(XInput.Text, out double xCm) && double.
TryParse(YInput.Text, out double yCm))
147         {
148             // Axis Flip:
149             // UI X -> Motor 2 (Physical Y)
150             // UI Y -> Motor 1 (Physical X)
151
152             int dx = (int)(yCm * STEPS_PER_CM_M1); // Motor 1 is now Y
input
153             int dy = (int)(xCm * STEPS_PER_CM_M2); // Motor 2 is now X
input
154
155             // Remap speed: UI 1-100% -> Actual 1-6.4 (Old 50%)
156             double uiSpeed = VelSlider.Value;
157             // Old max was 12 (range 11). New max is value at 50%: 1 +
(49/99)*11 = ~6.44
158             double targetMax = 1.0 + (50.0 - 1.0) * 11.0 / 99.0;
159             int actualSpeed = (int)Math.Round(1.0 + (uiSpeed - 1.0) *
(targetMax - 1.0) / 99.0);
160
161             SendPacket(dx, dy, actualSpeed);
162             string msg = $"Sent Move: X={xCm}cm, Y={yCm}cm @ {uiSpeed:
F0}%";
163             StatusText.Text = msg;
164             AddToHistory(msg);
165         }
166     }
167
168     private void StopBtn_Click(object sender, RoutedEventArgs e)

```

```

169     {
170         _stopRequested = true;
171         if (_serialPort == null || !_serialPort.IsOpen) return;
172         byte[] packet = new byte[8];
173         packet[0] = 255;
174         packet[1] = 2; // Stop
175         _serialPort.Write(packet, 0, 8);
176         StatusText.Text = "Sent Stop";
177     }
178
179     private void StopDrawingBtn_Click(object sender, RoutedEventArgs e
180     )
181     {
182         _stopRequested = true;
183         StatusText.Text = "Stopping...";
184         ProcStatusText.Text = "Stopping...";
185
186         // Also send hardware stop
187         if (_serialPort != null && _serialPort.IsOpen)
188         {
189             byte[] packet = new byte[8];
190             packet[0] = 255;
191             packet[1] = 2; // Stop
192             _serialPort.Write(packet, 0, 8);
193         }
194
195     private void ClearHistoryBtn_Click(object sender, RoutedEventArgs
196     e)
197     {
198         HistoryList.Items.Clear();
199         StatusText.Text = "History Cleared";
200     }
201
202     private void AddToHistory(string message)
203     {
204         string time = DateTime.Now.ToString("HH:mm:ss");
205         HistoryList.Items.Insert(0, $"[{time}] {message}");
206     }
207
208     // --- New Drawing & Tracing Logic ---
209
210     private void SetCenterBtn_Click(object sender, RoutedEventArgs e)
211     {
212         _currentX = 0;
213         _currentY = 0;
214         StatusText.Text = "Center Set to Current Position (0,0)";
215     }
216
217     private async void GoToCenterBtn_Click(object sender,
218     RoutedEventArgs e)
219     {
220         double moveX = -_currentX;
221         double moveY = -_currentY;

```

```

220         int dx = (int)(moveY * STEPS_PER_CM_M1);
221         int dy = (int)(moveX * STEPS_PER_CM_M2);
222
223         SendPacket(dx, dy, RETURN_SPEED); // Moderate speed for return
224         StatusText.Text = $"Returning to Center: {moveX:F2}, {moveY:F2}
225     }";
226
227     // Wait for move to complete (estimated)
228     int delay = (int)(Math.Max(Math.Abs(moveX), Math.Abs(moveY)) *
229         200 + 500);
230     await Task.Delay(delay);
231 }
232
233 private void ImportImageBtn_Click(object sender, RoutedEventArgs e
234 )
235 {
236     OpenFileDialog openFileDialog = new OpenFileDialog();
237     openFileDialog.Filter = "Image files (*.png;*.jpg;*.jpeg)|*.
238     png;*.jpg;*.jpeg|All files (*.*)|*.*";
239     if (openFileDialog.ShowDialog() == true)
240     {
241         BitmapImage bitmap = new BitmapImage(new Uri(
242             openFileDialog.FileName));
243         TraceImage.Source = bitmap;
244         StatusText.Text = "Image Imported";
245     }
246 }
247
248 private void ClearBtn_Click(object sender, RoutedEventArgs e)
249 {
250     DrawingCanvas.Strokes.Clear();
251     TraceImage.Source = null;
252     StatusText.Text = "Canvas Cleared";
253 }
254
255 // Drawing Limits
256 const double DRAWING_SIZE_CM = 8.0;
257
258 private async void TraceBtn_Click(object sender, RoutedEventArgs e
259 )
260 {
261     StatusText.Text = "Processing Image...";
262     await Task.Delay(100); // UI Refresh
263
264     // 1. Render Canvas to Bitmap
265     int width = (int)DrawingCanvas.ActualWidth;
266     int height = (int)DrawingCanvas.ActualHeight;
267
268     // We need to render the parent Grid to capture both Image and
269     InkCanvas
270     Grid parentGrid = (Grid)DrawingCanvas.Parent;
271
272     RenderTargetBitmap rtb = new RenderTargetBitmap(width, height,

```

```

96, 96, PixelFormats.Pbgra32);
rtb.Render(parentGrid);

// 2. Extract Points (Downsampling)
// Canvas is 300x300. Physical is 12x12cm.
// 1 pixel = 0.04 cm = 0.4 mm.

int stride = width * 4;
byte[] pixels = new byte[height * stride];
rtb.CopyPixels(pixels, stride, 0);

List<Point> points = new List<Point>();

for (int y = 0; y < height; y += 2)
{
    for (int x = 0; x < width; x += 2)
    {
        int index = y * stride + x * 4;
        // BGRA format
        byte b = pixels[index];
        byte g = pixels[index + 1];
        byte r = pixels[index + 2];
        byte a = pixels[index + 3];

        // Simple threshold: if dark enough AND not
transparent
        if (a > 50 && r < 128 && g < 128 && b < 128)
        {
            // Map pixel (0-300) to cm (-6 to 6)
            // x=0 -> -6, x=300 -> 6
            double cmX = (x / (double)width) * DRAWING_SIZE_CM
- (DRAWING_SIZE_CM / 2.0);
            // y=0 -> 6 (Top), y=300 -> -6 (Bottom)
            double cmY = -((y / (double)height) *
DRAWING_SIZE_CM - (DRAWING_SIZE_CM / 2.0));

            points.Add(new Point(cmX, cmY));
        }
    }

    if (points.Count == 0)
    {
        StatusText.Text = "No drawing found!";
        return;
    }

    StatusText.Text = $"Found {points.Count} points. Sorting...";
    await Task.Delay(100);

    // 3. Sort Points (Nearest Neighbor)
    List<Point> sortedPoints = SortPointsNearestNeighbor(points);

    // 4. Execute

```

```

317     StatusText.Text = "Tracing...";
318     _stopRequested = false;
319
320     // Move to first point
321     Point currentPos = new Point(_currentX, _currentY);
322
323     foreach (var target in sortedPoints)
324     {
325         if (_stopRequested)
326         {
327             StatusText.Text = "Tracing Stopped by User";
328             return;
329         }
330
331         double moveX = target.X - currentPos.X;
332         double moveY = target.Y - currentPos.Y;
333
334         // Skip tiny moves
335         if (Math.Abs(moveX) < 0.05 && Math.Abs(moveY) < 0.05)
336             continue;
337
338         int dx = (int)(moveY * STEPS_PER_CM_M1);
339         int dy = (int)(moveX * STEPS_PER_CM_M2);
340
341         // Low speed for tracing
342         SendPacket(dx, dy, TRACING_SPEED);
343
344         currentPos = target;
345
346         // Wait based on distance
347         double dist = Math.Sqrt(moveX*moveX + moveY*moveY);
348         int waitTime = (int)(dist * 100 + 50); // Heuristic
349         await Task.Delay(waitTime);
350     }
351
352     StatusText.Text = "Tracing Complete";
353 }
354
355 private List<Point> SortPointsNearestNeighbor(List<Point> points)
356 {
357     List<Point> sorted = new List<Point>();
358     HashSet<int> visited = new HashSet<int>();
359
360     // Start with the point closest to current position
361     Point current = new Point(_currentX, _currentY);
362
363     while (sorted.Count < points.Count)
364     {
365         int nearestIndex = -1;
366         double minDistSq = double.MaxValue;
367
368         for (int i = 0; i < points.Count; i++)
369         {
370             if (visited.Contains(i)) continue;

```

```

370
371         double dSq = (points[i].X - current.X) * (points[i].X
- current.X) +
372                     (points[i].Y - current.Y) * (points[i].Y
- current.Y);
373
374         if (dSq < minDistSq)
375         {
376             minDistSq = dSq;
377             nearestIndex = i;
378         }
379     }
380
381     if (nearestIndex != -1)
382     {
383         visited.Add(nearestIndex);
384         current = points[nearestIndex];
385         sorted.Add(current);
386     }
387     else
388     {
389         break;
390     }
391 }
392 return sorted;
393 }
394
395 // --- Image Processing Tab Logic ---
396
397 private void ProcImportBtn_Click(object sender, RoutedEventArgs e)
398 {
399     OpenFileDialog openFileDialog = new OpenFileDialog();
400     openFileDialog.Filter = "Image files (*.png;*.jpg;*.jpeg)|*.
png;*.jpg;*.jpeg|All files (*.*)|*.*";
401     if (openFileDialog.ShowDialog() == true)
402     {
403         BitmapImage bitmap = new BitmapImage(new Uri(
openFileDialog.FileName));
404         ProcImage.Source = bitmap;
405         ProcStatusText.Text = "Image Imported";
406     }
407 }
408
409 private async void ProcConvertBtn_Click(object sender,
RoutedEventArgs e)
410 {
411     if (ProcImage.Source == null) return;
412
413     ProcStatusText.Text = "Processing...";
414     await Task.Delay(10); // UI Refresh
415
416     BitmapSource source = (BitmapSource)ProcImage.Source;
417
418     // 1. Resize if needed

```

```

419     bool highRes = HighResCheck.IsChecked == true;
420     int targetWidth = highRes ? source.PixelWidth : 300;
421
422     if (source.PixelWidth > targetWidth)
423     {
424         double scale = (double)targetWidth / source.PixelWidth;
425         source = new TransformedBitmap(source, new ScaleTransform(
scale, scale));
426     }
427
428     // 2. Convert to Gray8 for simpler processing
429     FormatConvertedBitmap grayBitmap = new FormatConvertedBitmap()
;
430     grayBitmap.BeginInit();
431     grayBitmap.Source = source;
432     grayBitmap.DestinationFormat = PixelFormats.Gray8;
433     grayBitmap.EndInit();
434
435     int width = grayBitmap.PixelWidth;
436     int height = grayBitmap.PixelHeight;
437     int stride = width; // 1 byte per pixel
438     byte[] pixels = new byte[height * stride];
439     grayBitmap.CopyPixels(pixels, stride, 0);
440
441     byte[] resultPixels = new byte[height * stride];
442
443     // 3. Sobel Edge Detection
444     // Kernels
445     // Gx: -1 0 1
446     //      -2 0 2
447     //      -1 0 1
448     // Gy: -1 -2 -1
449     //      0 0 0
450     //      1 2 1
451
452     for (int y = 1; y < height - 1; y++)
453     {
454         for (int x = 1; x < width - 1; x++)
455         {
456             int i = y * stride + x;
457
458             // Gx
459             int gx = -pixels[i - stride - 1] + pixels[i - stride +
1]
460                     - 2 * pixels[i - 1] + 2 * pixels[i + 1]
461                     - pixels[i + stride - 1] + pixels[i + stride
+ 1];
462
463             // Gy
464             int gy = -pixels[i - stride - 1] - 2 * pixels[i -
stride] - pixels[i - stride + 1]
465                     + pixels[i + stride - 1] + 2 * pixels[i +
stride] + pixels[i + stride + 1];
466

```

```

467         int mag = (int)Math.Sqrt(gx * gx + gy * gy);
468
469         // Threshold
470         resultPixels[i] = (byte)(mag > 128 ? 255 : 0);
471     }
472 }
473
474 // 4. Create Result Bitmap
475 WriteableBitmap result = new WriteableBitmap(width, height,
96, 96, PixelFormats.Gray8, null);
476 result.WritePixels(new Int32Rect(0, 0, width, height),
resultPixels, stride, 0);
477
478 ProcImage.Source = result;
479 ProcStatusText.Text = "Edge Detection Complete";
480 }
481
482 private async void ProcDrawBtn_Click(object sender,
RoutedEventArgs e)
483 {
484     if (ProcImage.Source == null) return;
485
486     ProcStatusText.Text = "Analyzing Edges...";
487     await Task.Delay(10);
488
489     BitmapSource source = (BitmapSource)ProcImage.Source;
490
491     // Ensure it's Gray8 or convert
492     if (source.Format != PixelFormats.Gray8)
493     {
494         FormatConvertedBitmap gray = new FormatConvertedBitmap();
495         gray.BeginInit();
496         gray.Source = source;
497         gray.DestinationFormat = PixelFormats.Gray8;
498         gray.EndInit();
499         source = gray;
500     }
501
502     int width = source.PixelWidth;
503     int height = source.PixelHeight;
504     int stride = width; // 1 byte per pixel for Gray8
505     byte[] pixels = new byte[height * stride];
506     source.CopyPixels(pixels, stride, 0);
507
508     List<Point> points = new List<Point>();
509
510     // Collect points (White pixels are edges)
511     // Downsample for drawing if high res? Maybe.
512     // Let's sample every N pixels if it's huge, but the user
asked for high res processing.
513     // However, the robot has physical limits.
514     // Let's just take every pixel that is an edge.
515
516     for (int y = 0; y < height; y++)

```

```

517     {
518         for (int x = 0; x < width; x++)
519         {
520             if (pixels[y * stride + x] > 128) // Edge
521             {
522                 // Map to Physical Space (12x12 cm)
523                 // Center (0,0) is middle of image
524
525                 // Scale to fit? The above maps full width to 12cm
526                 .
527                 // If aspect ratio is not 1:1, we should preserve
528                 it.
529                 // Let's fit the largest dimension to
530                 DRAWING_SIZE_CM.
531
532                 double maxDim = Math.Max(width, height);
533                 double cmX = (x - width / 2.0) / maxDim *
534                 DRAWING_SIZE_CM;
535                 double cmY = -(y - height / 2.0) / maxDim *
536                 DRAWING_SIZE_CM;
537
538                 points.Add(new Point(cmX, cmY));
539             }
540         }
541
542         if (points.Count == 0)
543         {
544             ProcStatusText.Text = "No edges found!";
545             return;
546         }
547
548         // Optimization: If too many points, maybe simplify?
549         // For now, let's just run it.
550
551         ProcStatusText.Text = $"Found {points.Count} points. Sorting
552         ...";
553         await Task.Delay(100);
554
555         List<Point> sortedPoints = SortPointsNearestNeighbor(points);
556
557         ProcStatusText.Text = "Drawing...";
558         _stopRequested = false;
559
560         Point currentPos = new Point(_currentX, _currentY);
561
562         foreach (var target in sortedPoints)
563         {
564             if (_stopRequested)
565             {
566                 ProcStatusText.Text = "Drawing Stopped by User";
567                 return;
568             }
569         }

```

```

565         double moveX = target.X - currentPos.X;
566         double moveY = target.Y - currentPos.Y;
567
568         if (Math.Abs(moveX) < 0.05 && Math.Abs(moveY) < 0.05)
569             continue;
570
571         int dx = (int)(moveY * STEPS_PER_CM_M1);
572         int dy = (int)(moveX * STEPS_PER_CM_M2);
573
574         SendPacket(dx, dy, TRACING_SPEED);
575
576         currentPos = target;
577
578         double dist = Math.Sqrt(moveX*moveX + moveY*moveY);
579         int waitTime = (int)(dist * 100 + 50);
580         await Task.Delay(waitTime);
581     }
582     ProcStatusText.Text = "Drawing Complete";
583 }
584 }
585 }

```

Listing 5: MainWindow.xaml.cs