MECH 421 — Lab 5

PCB Assembly and Stepper Motor Control

Gyan Edbert Zesiro (ID: 38600060)
Ryan Edric Nashota (ID: 33508219)

Lab Performed: 7 November 2025
Report Submitted: 28 November 2025

Abstract

This lab focused on building and testing a PCB designed to control stepper, and DC motors through microprocessor commands, though for this lab we are only focusing on stepper motors. The lab was completed in three main sections. First, we assembled and soldered the PCB, verifying proper functionality of the power supply, USB interface, microprocessor, and motor driver components. We configured UART communication to enable data transfer for future exercises. Second, we developed firmware to control a stepper motor using half-stepping, implementing both single-step commands and continuous velocity control through a C# graphical interface. We measured the maximum loaded and unloaded speeds of the motor and analyzed the factors limiting its performance. Finally, we expanded the system to control a two-axis gantry stage using dual stepper motors. We programmed the microcontroller to execute coordinated movements, allowing the stage to follow straight-line paths between specified coordinates. The system successfully drew predetermined shapes and a custom design, demonstrating precise synchronized control of both axes. Throughout the lab, we gained practical experience in PCB assembly, firmware development, serial communication protocols, and motion control algorithms while exploring the advantages and limitations of stepper motors for precision positioning applications.

Table of Contents

Table of Figures

List of Tables

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. This exercise also teaches us how to solder SMD components and troubleshoot common issues.

2.2 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.

Reference	Description	Source	Qty
5x1 header programming	Bergstik II 0.100" straight header	Kit	1
Motor bypass cap	1000 µF 50V 20% alum SMD	Kit	2
MCU core bypass	0.47 µF 16V 5% X7R 0603	Kit	1
0.01 µF charge pump critical	10,000 pF 100V 5% NP0 1206	Kit	1
51 pF filter encoder	51 pF 50V 5% NP0 1206	Kit	2
20-pin MCU header	20 pos 0.100" straight header	Kit	2
DC wall jack	2.1 mm PCB power jack	Kit	1
USB connector	Mini-USB receptacle, 5 pos	Kit	1
PSU diode	Schottky diode 40V 3A DO-214AC	Kit	2
Polyfuse USB	Resettable fuse 6V 0.50A 1206	Kit	1
Latch	Dual D-type latch, 14-SOIC	Kit	2
Inverter	Hex Schmitt-trigger inverter, 14-SOIC	Kit	2
3.3V PSU	LDO regulator 3.3V 1A SOT-223	Kit	1
5.0V PSU	LDO regulator 5V 1A SOT-223	Kit	1
USB chip (FT230XS)	USB serial basic UART 16-SSOP	Kit	1
Green LED	Green LED clear 1206 SMD	Kit	6
Current sense resistors	0.2 Ω 2W 1% 2512	Kit	2
1.0 kΩ (Replace B)	1.0 kΩ 1/4W 5% 1206 SMD	Kit	3
2.7 kΩ encoder pullup 1%	2.7 kΩ 1/4W 1% 1206 SMD	Kit	4
27 Ω USB terminal 1%	27 Ω 1/4W 1% 1206 SMD	Kit	2
30 kΩ driver pullups	30 kΩ 1/4W 5% 1206 SMD	Kit	2
3.0 kΩ bypass resistors	3 kΩ 1/4W 5% 1206 SMD	Kit	4
47 kΩ programming pullup	47 kΩ 1/4W 5% 1206 SMD	Kit	1
Crystal	24.00 MHz resonator with caps, SMD	Kit	1
Screw terminal	5.08 mm vertical terminal block, 2-pos	Kit	4
Bypass general	0.1 µF 100V 10% X7R 1206	Lab	18
PSU bypass (Replace A)	4.7 µF 50V 10% X7R 1206	Lab	8
Current limiting resistors	150 Ω 1/10W 5% 0603 SMD	Lab	42
4x1 header (opt)	4 pos 0.100" straight header	Optional	3
XBee socket (opt)	10 pos 2 mm vertical socket	Optional	1
Test point (opt)	Mini 0.040" OD black test point	Optional	10
PSU bypass	10 µF 50V 20% X5R 1206	Replace A	8
2.4 kΩ driver LED (24VDC max)	2.4 kΩ 1/4W 5% 1206 SMD	Replace B	2
330 Ω 3.3V LED power	330 Ω 1/4W 5% 1206 SMD	Replace B	2
510 Ω 5V LED power	510 Ω 1/4W 5% 1206 SMD	Replace B	2
MCU	16-bit MCU, 16KB FRAM, 38-TSSOP	Kit	1
Motor controller	Motor driver, 28-HTSSOP	Kit	1

Table 1. Complete components list for PCB assembly

2.4 SMD and Through-Hole Soldering Techniques

Before commencing the full assembly, we established a standardized workflow for soldering the various component types. While hot air rework is effective for SMD parts with thermal pads, we completed the build with a fine-point soldering iron to maintain manual control.

The key enabler was flux. The kit's flux pen was insufficient for fine-pitch packages, so we switched to high-quality liquid flux from a syringe, which improved solder flow and reduced bridging.

2.4.1 Passive Components (0603 Resistors and Capacitors)

For small two-terminal parts we used a tack-and-reflow method:

1. Tin a single pad with a small solder dot.
2. Hold the component with tweezers, reflow the tinned pad, and slide the part into place.
3. Confirm it is flat; then solder the second pad.

Video: Soldering 0603 Resistors - Surface Mount (YouTube)

2.4.2 Integrated Circuits (Fine-Pitch ICs)

We used a pin-by-pin transfer to reduce bridges:

1. Align the chip and tack one corner pin.
2. Flood pins with liquid flux.
3. Place a small solder blob on the iron tip and touch each pin/pad junction; flux pulls solder onto the joint without bridging.
4. Clear any excess with copper braid.

Video: Soldering SMD ICs with a Soldering Iron (YouTube)

2.4.3 Through-Hole Components

For connectors (USB, JTAG, DC jack):

1. Insert leads and secure by bending or taping.
2. Heat pad and lead together; feed solder into the joint until a small cone forms.
3. Remove solder, then iron, and let cool undisturbed.

Video: How to Solder Through-Hole Components (YouTube)

2.4.4 Note on Hot Air Reflow

Hot air can self-align parts via surface tension and is preferred for production or rework. For this lab the iron technique above was sufficient. **Video**: Basics of Hot Air Soldering (YouTube)

2.5 Assembly Procedure

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

2.5.1 Step 1: Underside Resistors

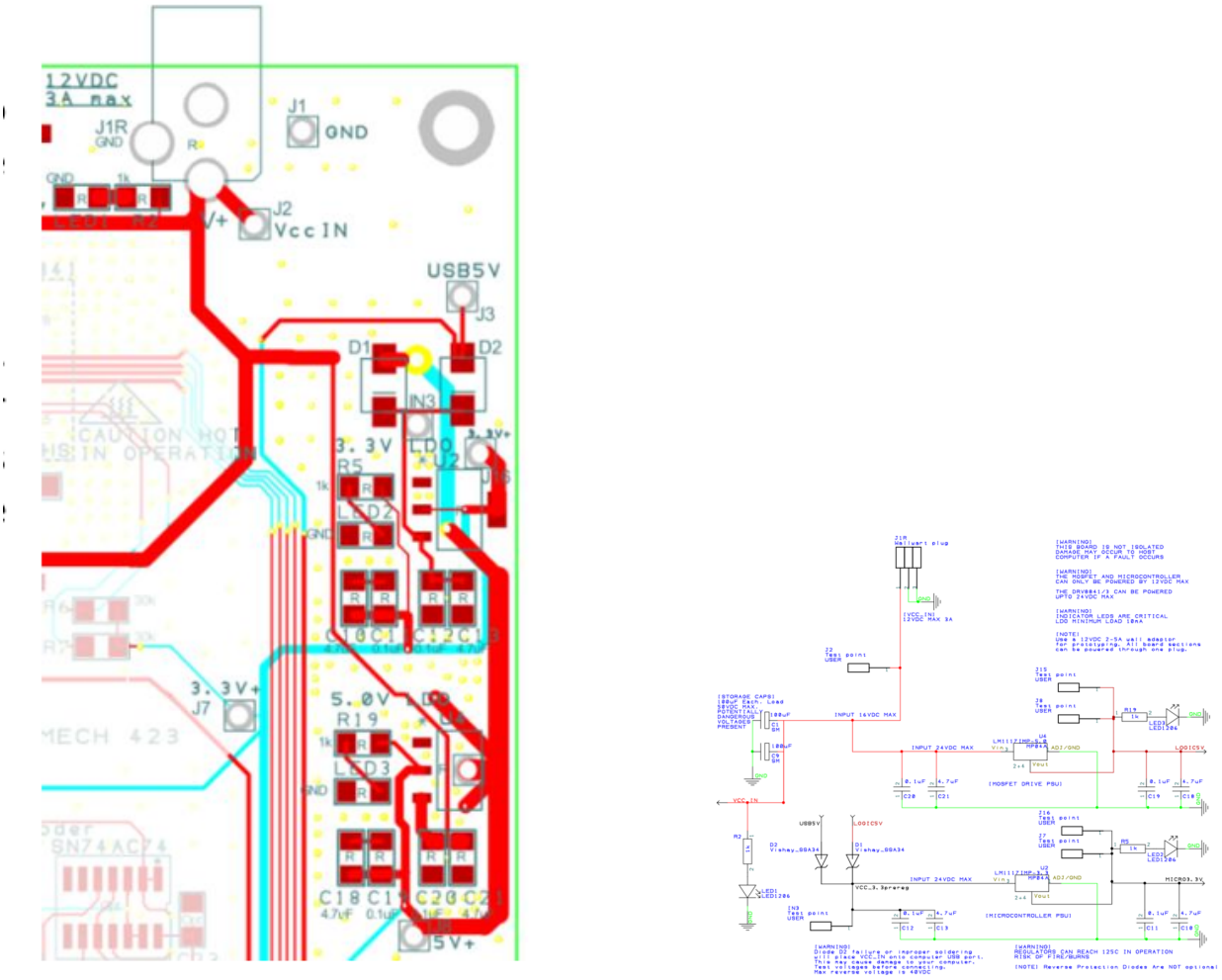
We began by soldering the 150 Ω current-limiting resistors on the bottom side. These 0603 parts also warmed us up for the finer SMD work.

- **Technique**: Tack-and-reflow (tin one pad, place with tweezers, reflow, then solder the second pad).
- **Check**: Measure across each resistor footprint with a multimeter; expect ≈150 Ω per resistor.

2.5.2 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**: Connect to a 12V bench supply. The power LED should glow brightly at 12V. Measure ≈5.0V and ≈3.3V on the regulators' outputs.



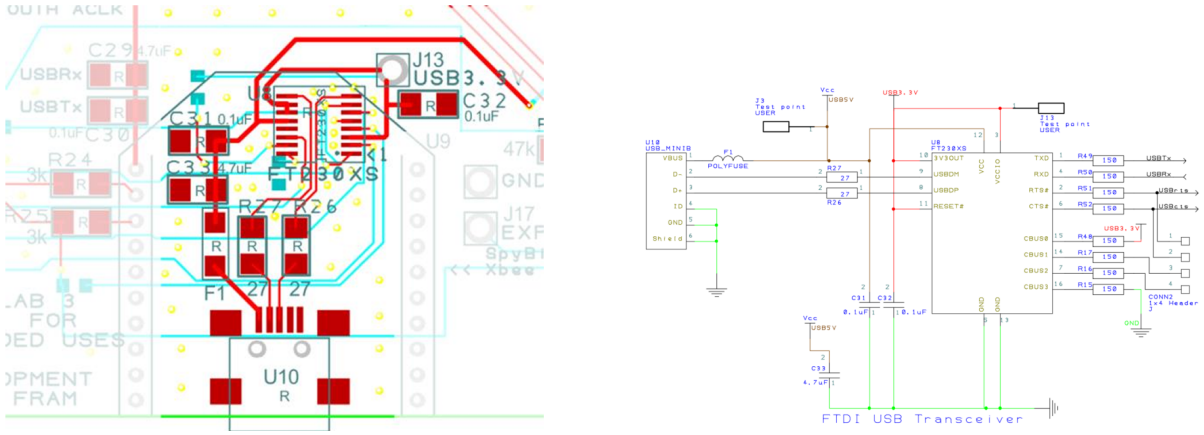
(a) Power supply PCB layout **(b)** Power supply schematic

Figure 1. Power supply reference during assembly

2.5.3 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**: Plug into a PC; listen for the USB connect chime and confirm the device enumerates as "USB Serial Port (COM3)". The FT230XS should stay cool to the touch (no noticeable heating).



(a) USB interface PCB layout **(b)** USB interface schematic

Figure 2. USB interface reference during assembly

2.5.4 Step 4: Microcontroller

The core of the system.

- **Components**: MSP430FR5739 (TSSOP-38), 24MHz crystal oscillator, JTAG header.
- **Verification**: Flash a simple LED blink via CCS; the blinking LED confirms JTAG wiring and basic MCU operation.

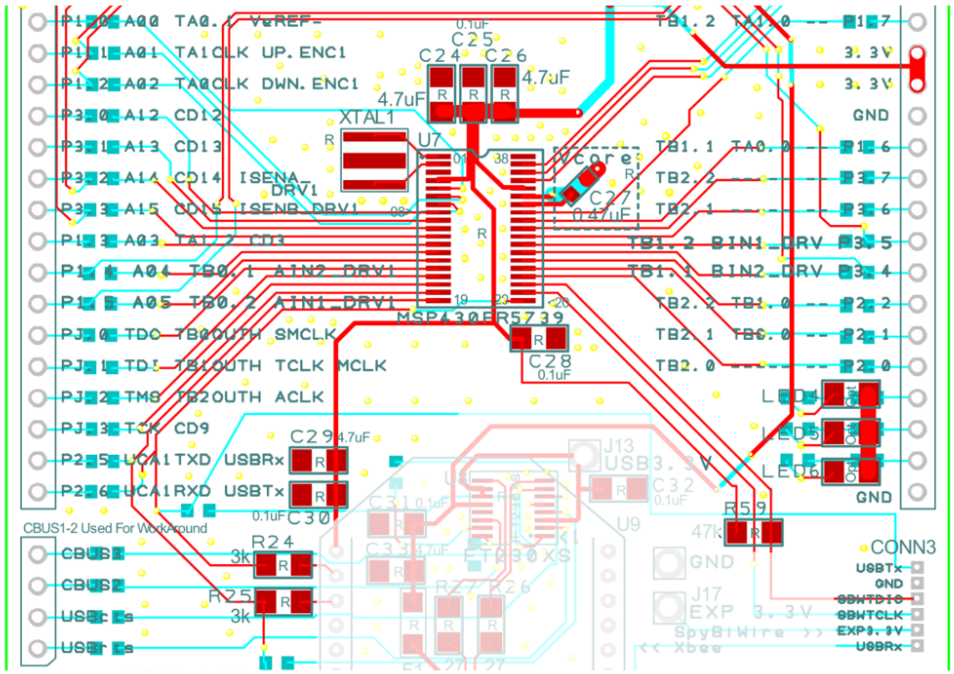


Figure 3. MSP430FR5739 layout reference

2.5.5 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.
- **Verification**: Use a UART test to communicate with firmware, then run a firmware routine to toggle the motor driver outputs and observe signals on the motor pins (scope or LED probe).

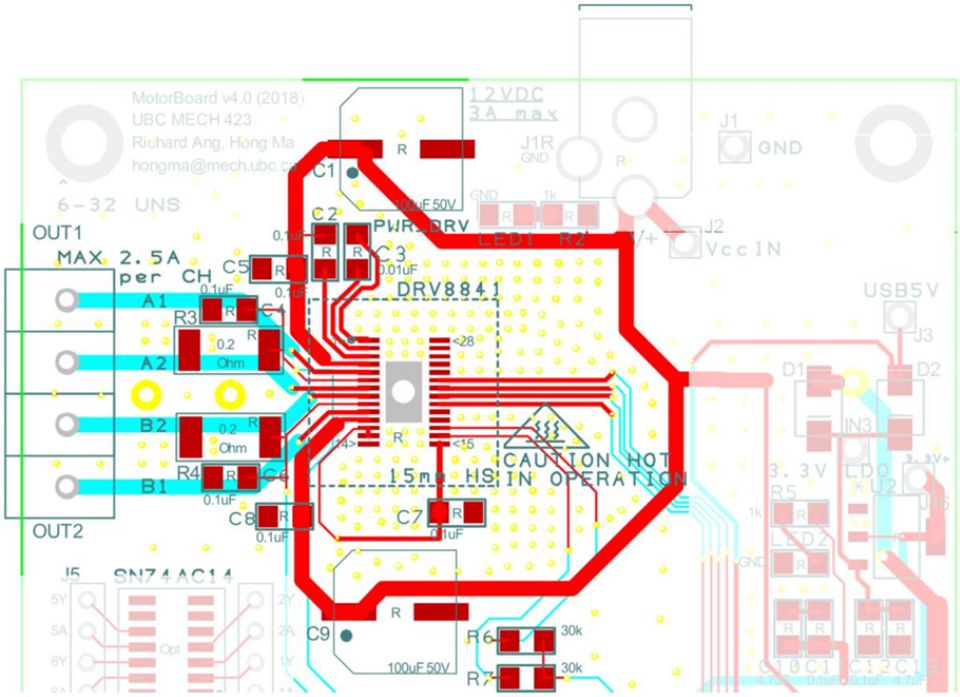


Figure 4. Motor driver layout reference

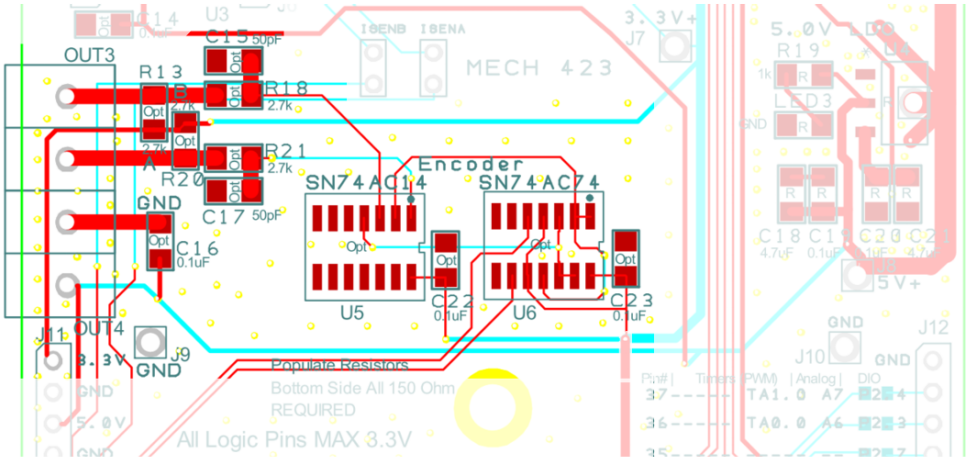


Figure 5. Shaft encoder layout reference

2.6 Results and Discussion

PLACEHOLDER FOR: Assembled PCB (Front)

Insert a clear photo of the fully assembled PCB (Front View).

Figure 6. Assembled PCB (Front)

PLACEHOLDER FOR: Assembled PCB (Back)

Insert a clear photo of the fully assembled PCB (Back View).

Figure 7. Assembled PCB (Back)

Challenges encountered:

1. Early joints on resistors/capacitors/LEDs worked electrically but looked rough; practice improved finish. One pad lifted when we overused the vacuum pump, but we repaired the net with a wire jumper.
2. Fine-pitch IC pins were hard to see; a magnifying glass was essential to spot bridges.
3. A solder bridge between pins 15 and 16 of the FT230XS blocked USB enumeration. We cleared it with braid plus liquid flux and cleaned with IPA; USB then enumerated.
4. A smaller-nozzle, temperature-controlled hot air tool would reduce collateral heat when reworking ICs; thick solder wire also made control harder, so thinner wire or solder paste would be better for future IC work.
5. The iron tip felt oversized at times; smaller tips plus paste would speed IC soldering and reduce errors.

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+ → B+ → A- → B-).
- **Half-Stepping**: Inserting intermediate states where both phases are energized, doubling the resolution (A+ → A+B+ → B+ ...).

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

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

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.

```
void step_motor1(uint8_t step) {
    uint8_t mask = stepper_table[step & 0x07];
    // Update PWM Duty Cycles based on mask
    TB0CCR2 = (mask & 0x01) ? PWM_DUTY : 0; // A1
    TB0CCR1 = (mask & 0x02) ? PWM_DUTY : 0; // A2
```

```
    TB1CCR2 = (mask & 0x04) ? PWM_DUTY : 0; // B1
    TB1CCR1 = (mask & 0x08) ? PWM_DUTY : 0; // B2
}
```

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:

```
// ISR triggered by Encoder Pulse
void Encoder_ISR() {
    if (ChannelA == ChannelB) position++;
    else position--;
}

// Control Loop (e.g., 100Hz Timer)
void Control_Loop() {
    error = target_pos - position;
    integral += error;
    derivative = error - prev_error;

    pwm_output = (Kp * error) + (Ki * integral) + (Kd * derivative);

    set_motor_pwm(pwm_output);
    prev_error = error;
}
```

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`):

```
public byte[] BuildPacket(byte dirByte, ushort ccr0)
{
    byte high = (byte)(ccr0 >> 8);
    byte low = (byte)(ccr0 & 0xFF);
    byte escape = 0;
    // ... escape handling logic ...
    return new[] { StartByte, dirByte, high, low, escape };
}
```

3.6 Results

PLACEHOLDER FOR: C Sharp GUI Screenshot

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

Figure 8. C Sharp GUI Screenshot

PLACEHOLDER FOR: Packet Structure

Insert a diagram of the UART packet structure (Start Byte, Command, Data, Checksum).

Figure 9. Packet Structure

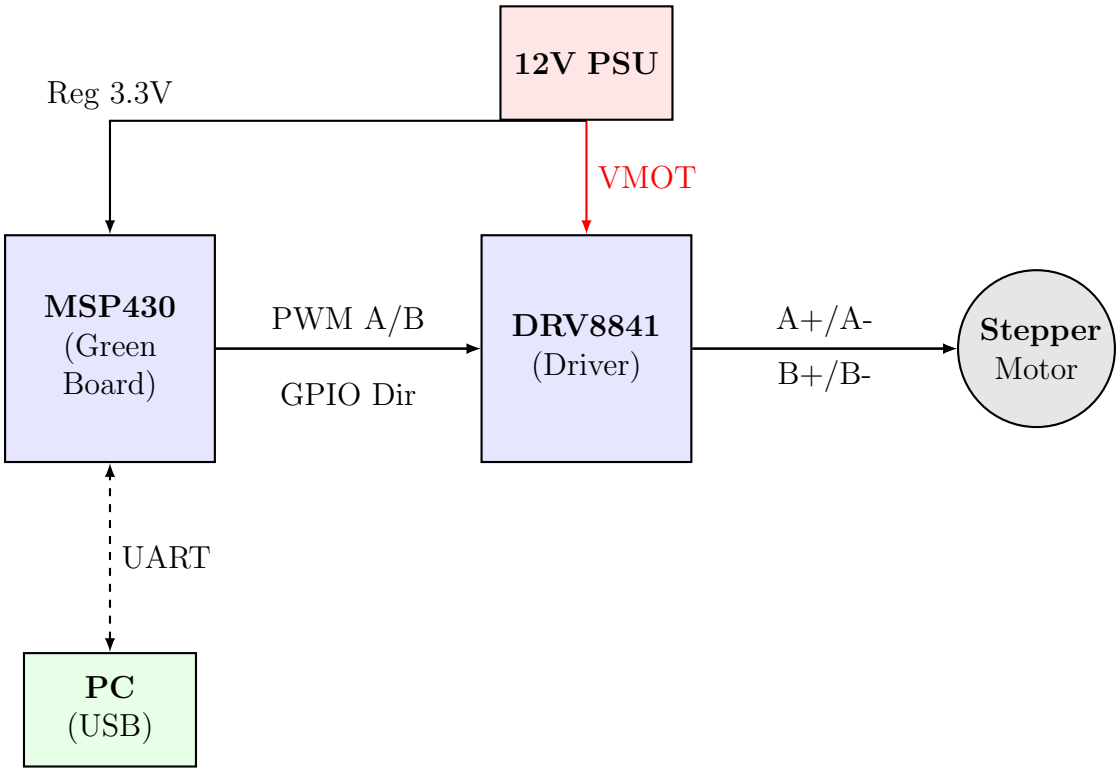


Figure 10. Electrical Schematic of Stepper Control System

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 a 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 11 illustrates the 2-axis connections.

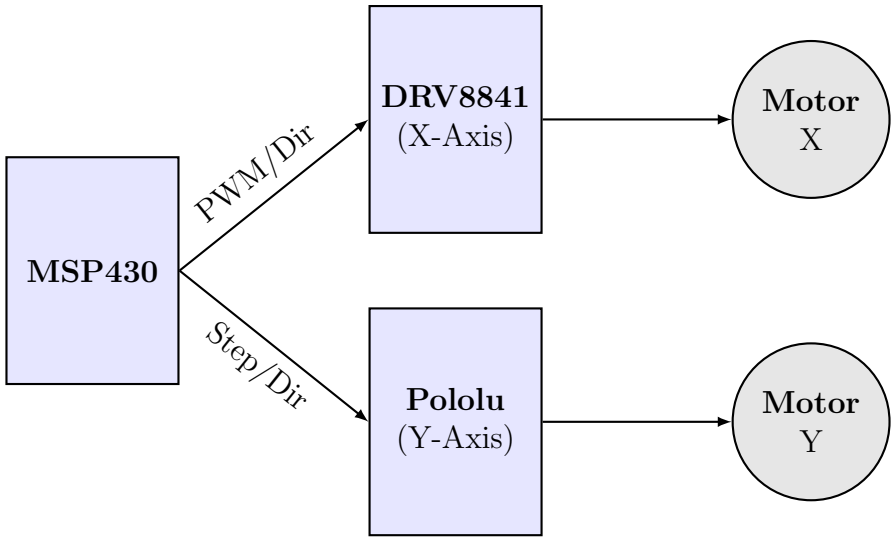


Figure 11. 2-Axis Control Wiring Diagram

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.

```
// DDA Algorithm in Timer_A1_ISR
acc_x += delta_x;
acc_y += delta_y;

if (acc_x >= total_steps_needed) {
    motor1_state = (motor1_state + step_x_inc) & 0x07;
    step_motor1(motor1_state);
    acc_x -= total_steps_needed;
}
// 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 12. Standard Shapes Plot

PLACEHOLDER FOR: Creative Shape Plot

Insert photo of the creative shape drawn by the gantry.

Figure 13. Creative Shape Plot

PLACEHOLDER FOR: Creative Shape Design Overlay

Insert photo of the drawn shape with the intended design overlaid for comparison.

Figure 14. 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#)

```
using System;
using System.IO.Ports;

namespace StepperControl
{
    /// <summary>
    /// Helper for building and sending stepper control packets and tracking commanded angle.
    /// </summary>
    public class StepperCommander
    {
        private const byte StartByte = 255;
        private const int TimerClockHz = 1_000_000;
        public const int StepsPerRevolution = 400; // half-step resolution

        private readonly SerialPort _port;

        /// <summary>
        /// Current commanded angle in degrees (0-360), updated by MoveByAngle/MoveToAngle/CalibrateZero.
        /// </summary>
        public double CurrentAngleDeg { get; private set; }

        /// <summary>
        /// Create a commander that writes packets to the provided SerialPort.
        /// </summary>
        /// <param name="port">Open SerialPort to write to.</param>
        public StepperCommander(SerialPort port)
        {
            _port = port ?? throw new ArgumentNullException(nameof(port));
        }

        /// <summary>
        /// Convert a desired continuous speed in RPM (sign = direction) to a packet for dirByte 1/2.
        /// Returns the packet and outputs dirByte and TA1CCR0 used.
        /// </summary>
        public byte[] BuildContinuousPacketFromRpm(double rpm, out ushort ccr0, out byte dirByte)
        {
            dirByte = rpm >= 0 ? (byte)2 : (byte)1;
            double stepsPerSecond = Math.Abs(rpm) * StepsPerRevolution / 60.0;

```

```
            // Avoid divide-by-zero; clamp CCR0 into [1, 65535].
            if (stepsPerSecond < 1e-6)
            {
                stepsPerSecond = 1e-6;
            }

            double rawCcr0 = (TimerClockHz / stepsPerSecond) - 1.0;
            rawCcr0 = Math.Max(1.0, Math.Min(65535.0, rawCcr0));
            ccr0 = (ushort)Math.Round(rawCcr0);

            return BuildPacket(dirByte, ccr0);
        }

        /// <summary>
        /// Build a single-step packet (dirByte 3 for CCW, 4 for CW).
        /// CCR0 is zero and escape is zero for single steps.
        /// </summary>
        public byte[] BuildSingleStepPacket(bool clockwise)
        {
            byte dir = clockwise ? (byte)4 : (byte)3;
            return new byte[] { StartByte, dir, 0, 0, 0 };
        }

        /// <summary>
        /// Set the commanded zero reference angle to 0 degrees.
        /// </summary>
        public void CalibrateZero()
        {
            CurrentAngleDeg = 0;
        }

        /// <summary>
        /// Move by a signed angle (deg). Positive = CW, negative = CCW.
        /// Rounds to nearest half-step, sends that many single-step
    packets, and wraps angle to [0, 360).
        /// </summary>
        public void MoveByAngle(double deltaDeg)
        {
            int steps = (int)Math.Round(deltaDeg * StepsPerRevolution /
    360.0);
            if (steps == 0)
            {
                return;
            }

            bool clockwise = steps > 0;
            int count = Math.Abs(steps);
            byte[] packet = BuildSingleStepPacket(clockwise);

            for (int i = 0; i < count; i++)
            {
                SendPacket(packet);
            }

```

```
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
```

```
            if (high == 255)
            {
                high = 0;
                escape |= 0b10;
            }

            if (low == 255)
            {
                low = 0;
                escape |= 0b01;
            }

            return new[] { StartByte, dirByte, high, low, escape };
        }
    }
}
```

Listing 2. StepperCommander.cs

```
using System;
using System.Drawing;
using System.IO.Ports;
using System.Linq;
using System.Windows.Forms;

namespace StepperControl
{
    public partial class Form1 : Form
    {
        private const byte StartByte = 255;
        private const int StepTimerClockHz = 1_000_000; // SMCLK/8 in
    firmware
        private readonly double _stepAngleDeg = 360.0 / StepperCommander.
    StepsPerRevolution;
        private StepperCommander commander;

        public Form1()
        {
            InitializeComponent();
            DoubleBuffered = true;
            velocityTrackBar.Minimum = -100;
            velocityTrackBar.Maximum = 100;
            velocityTrackBar.TickFrequency = 10;
            velocityTrackBar.Value = 0;
            maxSpeedNumeric.Value = 1200; // steps per second at slider =
    100
            PopulatePorts();
            UpdateVelocityLabels(0, 0, 0);
            UpdateModeAndFreq("Idle", 0);
            SetTelemetry(null, null);

            // Initialize Mode
            SwitchMode(true);
        }
```

```
33
34          private void continuousModeBtn_Click(object sender, EventArgs e)
35          {
36              SwitchMode(true);
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              if (ports.Length == 0)
73              {
74                  portComboBox.Text = "No COM ports found";
75                  connectButton.Enabled = false;
76                  statusLabel.Text = "Connect board before opening the port.
    ";
77              }
78              else
79              {
80                  portComboBox.Items.AddRange(ports);
81                  portComboBox.SelectedIndex = 0;
82                  connectButton.Enabled = true;
83                  statusLabel.Text = "Select a COM port and press Connect.";
```

```
            }
        }

        private void connectButton_Click(object sender, EventArgs e)
        {
            if (serialPort.IsOpen)
            {
                serialPort.Close();
                commander = null;
                connectButton.Text = "Connect";
                statusLabel.Text = "Serial port closed.";
                return;
            }

            if (portComboBox.SelectedItem == null)
            {
                MessageBox.Show("Select a COM port first.");
                return;
            }

            serialPort.PortName = portComboBox.SelectedItem.ToString();
            serialPort.BaudRate = 9600;
            serialPort.Parity = Parity.None;
            serialPort.StopBits = StopBits.One;
            serialPort.DataBits = 8;

            try
            {
                serialPort.Open();
                commander = new StepperCommander(serialPort);
                connectButton.Text = "Disconnect";
                statusLabel.Text = $"Connected to {serialPort.PortName} at
 {serialPort.BaudRate} baud.";
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Failed to open {serialPort.PortName}: {
ex.Message}");
            }
        }

        private void refreshButton_Click(object sender, EventArgs e)
        {
            PopulatePorts();
        }

        private void ccwStepButton_Click(object sender, EventArgs e)
        {
            if (!EnsureCommander()) return;
            try
            {
                commander.MoveByAngle(-_stepAngleDeg);
                UpdateModeAndFreq("Single CCW step", 0);
                SetTelemetry(commander.CurrentAngleDeg, null);
```

```
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Failed to step CCW: {ex.Message}");
            }
        }

        private void cwStepButton_Click(object sender, EventArgs e)
        {
            if (!EnsureCommander()) return;
            try
            {
                commander.MoveByAngle(_stepAngleDeg);
                UpdateModeAndFreq("Single CW step", 0);
                SetTelemetry(commander.CurrentAngleDeg, null);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Failed to step CW: {ex.Message}");
            }
        }

        private void stopButton_Click(object sender, EventArgs e)
        {
            StopMotor("Stop (DirnByte 3 halts timer)");
        }

        private void velocityTrackBar_Scroll(object sender, EventArgs e)
        {
            HandleVelocityChange();
        }

        private void maxSpeedNumeric_ValueChanged(object sender, EventArgs
    e)
        {
            HandleVelocityChange();
        }

        private void HandleVelocityChange()
        {
            int sliderValue = velocityTrackBar.Value;
            double maxStepsPerSecond = (double)maxSpeedNumeric.Value;

            if (sliderValue == 0)
            {
                UpdateVelocityLabels(0, 0, 0);
                StopMotor("Slider at zero -> stop timer");
                UpdateModeAndFreq("Stopped", 0);
                SetTelemetry(commander?.CurrentAngleDeg, 0);
                return;
            }

            double fraction = Math.Abs(sliderValue) / 100.0;
            double commandedStepsPerSecond = Math.Max(1.0, fraction *
```

```
maxStepsPerSecond);
            bool clockwise = sliderValue > 0;
            double rpm = commandedStepsPerSecond * 60.0 / StepperCommander
.StepsPerRevolution;
            if (!clockwise) rpm *= -1;

            if (!EnsureCommander()) return;

            try
            {
                byte dir;
                ushort ccr0;
                byte[] packet = commander.BuildContinuousPacketFromRpm(rpm
, out ccr0, out dir);
                commander.SendPacket(packet);

                UpdateVelocityLabels(sliderValue, commandedStepsPerSecond,
 ccr0);
                UpdateModeAndFreq($"Continuous {(clockwise ? "CW" : "CCW")
}", ccr0);
                SetTelemetry(commander.CurrentAngleDeg,
commandedStepsPerSecond);
                statusLabel.Text = $"Continuous {(clockwise ? "CW" : "CCW"
)} | {commandedStepsPerSecond:F1} steps/s | CCR0 {ccr0}";
                lastPacketLabel.Text = $"Last Packet [{string.Join(", ",
packet.Select(b => b.ToString()))}]";
            }
            catch (Exception ex)
            {
                statusLabel.Text = "Write failed.";
                MessageBox.Show($"Failed to send packet: {ex.Message}");
            }
        }

        private void StopMotor(string reason)
        {
            if (velocityTrackBar.Value != 0)
            {
                velocityTrackBar.Value = 0;
            }

            if (commander != null && serialPort.IsOpen)
            {
                byte[] packet = { StartByte, 3, 0, 0, 0 };
                try
                {
                    commander.SendPacket(packet);
                    lastPacketLabel.Text = $"Last Packet [{string.Join(",
", packet.Select(b => b.ToString()))}]";
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Failed to stop motor: {ex.Message}")
;
```

```
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
```

```
// 20kHz: Period=400 (Silent)
#define PWM_PERIOD 400
#define PWM_DUTY   (PWM_PERIOD * 80 / 100)

// Stepper Constants
#define STEPS_PER_REV 200 // Standard 1.8 deg stepper
#define MICROSTEPS 1      // Full stepping for now
// Assuming some calibration: Steps per CM.
// Let's assume 1 rev = 4 cm travel (example). -> 50 steps/cm.
// User can adjust this calibration in C# or here.
// For now, we will receive "Steps" directly from C# to keep firmware simple,
// OR we receive CM and convert. The prompt says "takes a relative [X,Y] distance".
// It's better to do the math in C# and send raw steps to MCU.
// So Packet will contain STEPS.

// ==================== GLOBALS ====================
volatile unsigned char queue[QUEUE_SIZE];
volatile unsigned int front = 0;
volatile unsigned int numItems = 0;

// Packet Buffer
volatile unsigned char startByte;
volatile unsigned char cmdByte; // 1=Move, 2=Stop
volatile unsigned char xH, xL, yH, yL;
volatile unsigned char velByte;
volatile unsigned char escapeByte;

// Motion Control Globals
volatile int32_t target_x_steps = 0;
volatile int32_t target_y_steps = 0;
volatile int32_t current_x_steps = 0;
volatile int32_t current_y_steps = 0;

volatile int32_t delta_x = 0;
volatile int32_t delta_y = 0;
volatile int32_t step_x_inc = 0;
volatile int32_t step_y_inc = 0;
volatile int32_t error_term = 0;
volatile uint32_t total_steps_needed = 0;
volatile uint32_t steps_taken = 0;

volatile uint8_t motor1_state = 0;
volatile uint8_t motor2_state = 0;
volatile uint8_t is_moving = 0;

// Half-step lookup table (8 steps)
// bit0=A1, bit1=A2, bit2=B1, bit3=B2
static const uint8_t stepper_table[8] = {
    0b0001, // 1: A1
    0b0101, // 2: A1+B1
    0b0100, // 3: B1
    0b0110, // 4: B1+A2
```

```
    0b0010, // 5: A2
    0b1010, // 6: A2+B2
    0b1000, // 7: B2
    0b1001  // 8: B2+A1
};

// ==================== PROTOTYPES ====================
void clockSetup(void);
void gpioSetup(void);
void timerSetup(void);
void uartSetup(void);
void step_motor1(uint8_t step);
void step_motor2(uint8_t step);
void process_packet(void);
void start_move(int16_t dx, int16_t dy, uint8_t speed);

// ==================== SETUP FUNCTIONS ====================
void clockSetup(void) {
    CSCTL0_H = CSKEY_H;
    CSCTL1 = DCOFSEL_3; // 8 MHz
    CSCTL2 = SELS__DCOCLK | SELA__DCOCLK | SELM__DCOCLK;
    CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;
    CSCTL0_H = 0;
}

void gpioSetup(void) {
    // Motor 1 (Existing) - PWM pins handled in Timer Setup, but we need to ensure directions
    // P1.4, P1.5, P3.4, P3.5 are used by Timer B0/B1 in the example.
    // We will stick to the example's method of using TBxCCRx for Motor 1.

    // Motor 2 (New) - GPIO Control
    // P1.3 (PWM/Enable) -> Output, High
    P1DIR |= M2_PWM_PIN;
    P1OUT |= M2_PWM_PIN; // Enable driver

    // Port J (PJ.0 - PJ.3) for Motor 2 Coils
    // Note: PJ is often shared with JTAG. Ensure JTAG is not interfering if debugging.
    PJDIR |= M2_AIN2 | M2_AIN1 | M2_BIN1 | M2_BIN2;
    PJOUT &= ~(M2_AIN2 | M2_AIN1 | M2_BIN1 | M2_BIN2);
    // Clear SEL bits to ensure GPIO mode if necessary (though usually default is GPIO for PJ on some devices, check datasheet)
    // On FR5739, PJ.0-3 are JTAG. To use as GPIO, we might need to be careful.
    // Assuming user has this working or knows the setup.
    PJSEL0 &= ~(M2_AIN2 | M2_AIN1 | M2_BIN1 | M2_BIN2);
    PJSEL1 &= ~(M2_AIN2 | M2_AIN1 | M2_BIN1 | M2_BIN2);

    // Unlock GPIO
    PM5CTL0 &= ~LOCKLPM5;
}

void timerSetup(void) {
```

```
    // --- Timer A1: Step Clock ---
    // We will use TA1 for the motion tick.
    // Frequency will be set dynamically based on speed.
    TA1CTL = TASSEL__SMCLK | MC__STOP | TACLR;
    TA1CCTL0 = CCIE;

    // --- Timer B0/B1: Motor 1 PWM Generation (from example) ---
    // TB0: P1.4(A2), P1.5(A1)
    TB0CCR0 = PWM_PERIOD - 1;
    TB0CCTL1 = OUTMOD_7; // Reset/Set
    TB0CCTL2 = OUTMOD_7;
    TB0CTL = TBSSEL__SMCLK | MC__UP | TBCLR;

    // TB1: P3.4(B2), P3.5(B1)
    TB1CCR0 = PWM_PERIOD - 1;
    TB1CCTL1 = OUTMOD_7;
    TB1CCTL2 = OUTMOD_7;
    TB1CTL = TBSSEL__SMCLK | MC__UP | TBCLR;

    // Set Pins for TB0/TB1
    P1DIR |= BIT4 | BIT5;
    P1SEL0 |= BIT4 | BIT5;
    P1SEL1 &= ~(BIT4 | BIT5);

    P3DIR |= BIT4 | BIT5;
    P3SEL0 |= BIT4 | BIT5;
    P3SEL1 &= ~(BIT4 | BIT5);
}

void uartSetup(void) {
    // P2.5=RX, P2.6=TX
    P2SEL1 |= BIT5 | BIT6;
    P2SEL0 &= ~(BIT5 | BIT6);

    UCA1CTLW0 |= UCSWRST;
    UCA1CTLW0 |= UCSSEL__SMCLK; // Use SMCLK (8MHz)
    // 9600 Baud from 8MHz
    // N = 8000000/9600 = 833.33
    // UCBRx = 52, UCBRFx = 1, UCBRSx = 0x49 (from example)
    // Wait, example used ACLK=8MHz. We set SMCLK=8MHz. Same difference.
    UCA1MCTLW = UCOS16 | 0x4900 | 0x0010; // UCBRF=1
    UCA1BRW = 52;

    UCA1CTLW0 &= ~UCSWRST;
    UCA1IE |= UCRXIE;
}

// ==================== MOTOR CONTROL ====================
void step_motor1(uint8_t step) {
    // Motor 1 uses Timer PWMs (TB0, TB1)
    uint8_t mask = stepper_table[step & 0x07];

    // A1 (TB0.2)
    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
    just standard Bresenham.
231     // Let's use a counter approach for both.
232     // We want to complete 'total_steps_needed' ticks.
233     // On each tick:
234     //   AccumulatorX += delta_x
235     //   if AccumulatorX >= total_steps_needed: step X, AccumulatorX -=
```

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

```
        // Invalid start, consume one byte
        front = (front + 1) % QUEUE_SIZE;
        numItems--;
        return;
    }

    // Extract bytes
    unsigned char pkt[8];
    int i;
    for (i = 0; i < 8; i++) {
        pkt[i] = queue[(front + i) % QUEUE_SIZE];
    }

    // Handle Escape
    unsigned char esc = pkt[7];
    if (esc & 0x01) pkt[6] = 255; // Vel
    if (esc & 0x02) pkt[5] = 255; // YL
    if (esc & 0x04) pkt[4] = 255; // YH
    if (esc & 0x08) pkt[3] = 255; // XL
    if (esc & 0x10) pkt[2] = 255; // XH
    // CMD usually doesn't need escape if it's small enum

    int16_t dx = (int16_t)((pkt[2] << 8) | pkt[3]);
    int16_t dy = (int16_t)((pkt[4] << 8) | pkt[5]);
    uint8_t vel = pkt[6];
    uint8_t cmd = pkt[1];

    // Consume queue
    front = (front + 8) % QUEUE_SIZE;
    numItems -= 8;

    if (cmd == 1) { // MOVE
        start_move(dx, dy, vel);
    } else if (cmd == 2) { // STOP
        TA1CTL &= ~MC_3;
        is_moving = 0;
    }
}

// ==================== ISRs ====================
#pragma vector = USCI_A1_VECTOR
__interrupt void USCI_A1_ISR(void) {
    if (UCA1IFG & UCRXIFG) {
        unsigned char rx = UCA1RXBUF;
        if (numItems < QUEUE_SIZE) {
            queue[(front + numItems) % QUEUE_SIZE] = rx;
            numItems++;
        }
    }
}

#pragma vector = TIMER1_A0_VECTOR
__interrupt void Timer_A1_ISR(void) {
    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
  .ToString(), 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             _stopRequested = true;
182             StatusText.Text = "Stopping...";
183             ProcStatusText.Text = "Stopping...";
184 
185             // Also send hardware stop
186             if (_serialPort != null && _serialPort.IsOpen)
187             {
188                 byte[] packet = new byte[8];
189                 packet[0] = 255;
190                 packet[1] = 2; // Stop
191                 _serialPort.Write(packet, 0, 8);
192             }
193         }
194 
195         private void ClearHistoryBtn_Click(object sender, RoutedEventArgs
    e)
196         {
197             HistoryList.Items.Clear();
198             StatusText.Text = "History Cleared";
199         }
200 
201         private void AddToHistory(string message)
202         {
203             string time = DateTime.Now.ToString("HH:mm:ss");
204             HistoryList.Items.Insert(0, $"[{time}] {message}");
205         }
206 
207         // --- New Drawing & Tracing Logic ---
208 
209         private void SetCenterBtn_Click(object sender, RoutedEventArgs e)
210         {
211             _currentX = 0;
212             _currentY = 0;
213             StatusText.Text = "Center Set to Current Position (0,0)";
214         }
215 
216         private async void GoToCenterBtn_Click(object sender,
    RoutedEventArgs e)
217         {
218             double moveX = -_currentX;
219             double moveY = -_currentY;
```

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

```
    96, 96, PixelFormats.Pbgra32);
267             rtb.Render(parentGrid);
268 
269             // 2. Extract Points (Downsampling)
270             // Canvas is 300x300. Physical is 12x12cm.
271             // 1 pixel = 0.04 cm = 0.4 mm.
272 
273             int stride = width * 4;
274             byte[] pixels = new byte[height * stride];
275             rtb.CopyPixels(pixels, stride, 0);
276 
277             List<Point> points = new List<Point>();
278 
279             for (int y = 0; y < height; y += 2)
280             {
281                 for (int x = 0; x < width; x += 2)
282                 {
283                     int index = y * stride + x * 4;
284                     // BGRA format
285                     byte b = pixels[index];
286                     byte g = pixels[index + 1];
287                     byte r = pixels[index + 2];
288                     byte a = pixels[index + 3];
289 
290                     // Simple threshold: if dark enough AND not
    transparent
291                     if (a > 50 && r < 128 && g < 128 && b < 128)
292                     {
293                         // Map pixel (0-300) to cm (-6 to 6)
294                         // x=0 -> -6, x=300 -> 6
295                         double cmX = (x / (double)width) * DRAWING_SIZE_CM
     - (DRAWING_SIZE_CM / 2.0);
296                         // y=0 -> 6 (Top), y=300 -> -6 (Bottom)
297                         double cmY = -((y / (double)height) *
    DRAWING_SIZE_CM - (DRAWING_SIZE_CM / 2.0));
298 
299                         points.Add(new Point(cmX, cmY));
300                     }
301                 }
302             }
303 
304             if (points.Count == 0)
305             {
306                 StatusText.Text = "No drawing found!";
307                 return;
308             }
309 
310             StatusText.Text = $"Found {points.Count} points. Sorting...";
311             await Task.Delay(100);
312 
313             // 3. Sort Points (Nearest Neighbor)
314             List<Point> sortedPoints = SortPointsNearestNeighbor(points);
315 
316             // 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)
    continue;
336 
337                 int dx = (int)(moveY * STEPS_PER_CM_M1);
338                 int dy = (int)(moveX * STEPS_PER_CM_M2);
339 
340                 // Low speed for tracing
341                 SendPacket(dx, dy, TRACING_SPEED);
342 
343                 currentPos = target;
344 
345                 // Wait based on distance
346                 double dist = Math.Sqrt(moveX*moveX + moveY*moveY);
347                 int waitTime = (int)(dist * 100 + 50); // Heuristic
348                 await Task.Delay(waitTime);
349             }
350 
351             StatusText.Text = "Tracing Complete";
352         }
353 
354         private List<Point> SortPointsNearestNeighbor(List<Point> points)
355         {
356             List<Point> sorted = new List<Point>();
357             HashSet<int> visited = new HashSet<int>();
358 
359             // Start with the point closest to current position
360             Point current = new Point(_currentX, _currentY);
361 
362             while (sorted.Count < points.Count)
363             {
364                 int nearestIndex = -1;
365                 double minDistSq = double.MaxValue;
366 
367                 for (int i = 0; i < points.Count; i++)
368                 {
369                     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 
```

```
                    int mag = (int)Math.Sqrt(gx * gx + gy * gy);

                    // Threshold
                    resultPixels[i] = (byte)(mag > 128 ? 255 : 0);
                }
            }

            // 4. Create Result Bitmap
            WriteableBitmap result = new WriteableBitmap(width, height, 96, 96, PixelFormats.Gray8, null);
            result.WritePixels(new Int32Rect(0, 0, width, height), resultPixels, stride, 0);

            ProcImage.Source = result;
            ProcStatusText.Text = "Edge Detection Complete";
        }

        private async void ProcDrawBtn_Click(object sender, RoutedEventArgs e)
        {
            if (ProcImage.Source == null) return;

            ProcStatusText.Text = "Analyzing Edges...";
            await Task.Delay(10);

            BitmapSource source = (BitmapSource)ProcImage.Source;

            // Ensure it's Gray8 or convert
            if (source.Format != PixelFormats.Gray8)
            {
                FormatConvertedBitmap gray = new FormatConvertedBitmap();
                gray.BeginInit();
                gray.Source = source;
                gray.DestinationFormat = PixelFormats.Gray8;
                gray.EndInit();
                source = gray;
            }

            int width = source.PixelWidth;
            int height = source.PixelHeight;
            int stride = width; // 1 byte per pixel for Gray8
            byte[] pixels = new byte[height * stride];
            source.CopyPixels(pixels, stride, 0);

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

            // Collect points (White pixels are edges)
            // Downsample for drawing if high res? Maybe.
            // Let's sample every N pixels if it's huge, but the user asked for high res processing.
            // However, the robot has physical limits.
            // Let's just take every pixel that is an edge.

            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                         // If aspect ratio is not 1:1, we should preserve it.
527                         // Let's fit the largest dimension to DRAWING_SIZE_CM.
528 
529                         double maxDim = Math.Max(width, height);
530                         double cmX = (x - width / 2.0) / maxDim * DRAWING_SIZE_CM;
531                         double cmY = -(y - height / 2.0) / maxDim * DRAWING_SIZE_CM;
532 
533                         points.Add(new Point(cmX, cmY));
534                     }
535                 }
536             }
537 
538             if (points.Count == 0)
539             {
540                 ProcStatusText.Text = "No edges found!";
541                 return;
542             }
543 
544             // Optimization: If too many points, maybe simplify?
545             // For now, let's just run it.
546 
547             ProcStatusText.Text = $"Found {points.Count} points. Sorting...";
548             await Task.Delay(100);
549 
550             List<Point> sortedPoints = SortPointsNearestNeighbor(points);
551 
552             ProcStatusText.Text = "Drawing...";
553             _stopRequested = false;
554 
555             Point currentPos = new Point(_currentX, _currentY);
556 
557             foreach (var target in sortedPoints)
558             {
559                 if (_stopRequested)
560                 {
561                     ProcStatusText.Text = "Drawing Stopped by User";
562                     return;
563                 }
564 
```

```
                double moveX = target.X - currentPos.X;
                double moveY = target.Y - currentPos.Y;

                if (Math.Abs(moveX) < 0.05 && Math.Abs(moveY) < 0.05)
    continue;

                int dx = (int)(moveY * STEPS_PER_CM_M1);
                int dy = (int)(moveX * STEPS_PER_CM_M2);

                SendPacket(dx, dy, TRACING_SPEED);

                currentPos = target;

                double dist = Math.Sqrt(moveX*moveX + moveY*moveY);
                int waitTime = (int)(dist * 100 + 50);
                await Task.Delay(waitTime);
            }

            ProcStatusText.Text = "Drawing Complete";
        }
    }
}
```

Listing 5. MainWindow.xaml.cs