

MECH 421/423 Lab 4

Op-Amp Circuits for Noisy Environments

Ryan Edric Nashota
Student ID: 33508129

November 17, 2025

Contents

1	Introduction	2
2	Exercise 1	3
2.1	LED Current Requirement	3
2.2	Low-Frequency Drive Verification	4
2.3	High-Frequency Drive and Mount Setup	4
3	Exercise 2	5
3.1	Selecting R2	5
3.2	Resistor Bias Selection	5
3.3	Cut-off Frequency Design	6
3.4	Ambient-Light Observation	6
3.5	Carrier Detection at the Photodiode	9
3.6	High-Pass Filter Measurement	9
3.7	Mechanical Alignment Guidance	11
4	Exercise 3	11
4.1	High-Pass Gain Stage Design	11
4.2	Low-Pass Noise Filter	12
4.3	Signal-Generator Verification	12
4.4	Linking to the Photodiode Stage	13
4.5	Distance-Dependent Gain Check	13
4.6	Gain Optimization	14
4.7	Final Component Values	14
5	Exercise 4	15
5.1	Second High-Pass Stage	15
5.2	Rectifier Gain	15
5.3	Low-Pass Envelope Filter	15

5.4	Full-Chain Testing	16
5.5	Documented Component Values	18
6	Exercise 5	18
6.1	Circuit Integration	18
6.2	Output Range Adjustment	18
6.3	Final Component Values	19
7	Exercise 6	19
7.1	MSP430 Firmware	19
7.2	C# Data Acquisition Application	20
8	Exercise 7	20
8.1	Distance Sweep	20
8.2	Curve Fitting	20
8.3	Position Conversion	21
8.4	Noise Measurement	23
9	Conclusion	24
A	C# Application Source Code	24
A.1	Program Entry Point	24
A.2	Main Application Form	25
A.3	Noise Analysis UI Panel	60
A.4	Serial Port Handling	78
A.5	Data Logging	82
A.6	Calibration Utilities	84
A.7	Noise Analysis Helpers	89

1 Introduction

This lab investigates the design, construction, and calibration of a modulated optical distance sensor that can operate reliably in a bright laboratory. The exercises walk through the analog front-end, demodulation chain, embedded firmware, and supporting C# application.

For reference, this is the circuit found in the lab manual, please always refer to this

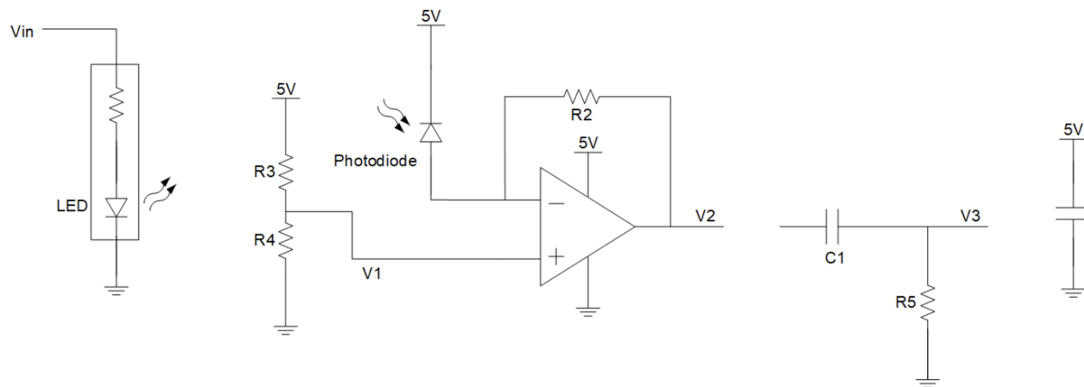


Figure 1: Exercise 2 Circuit Diagram from Lab Manual

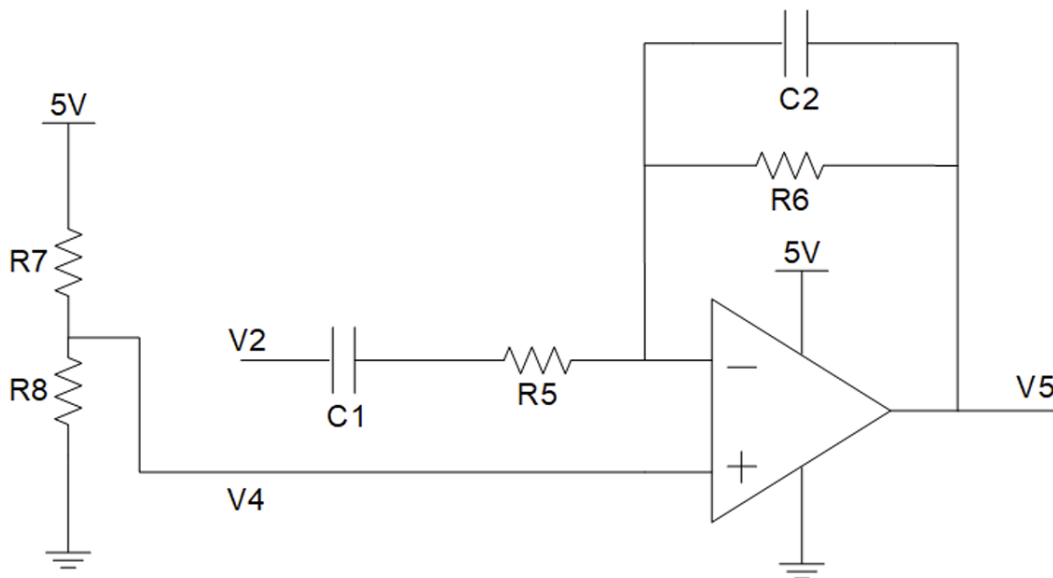


Figure 2: Exercise 3 Circuit Diagram from Lab Manual

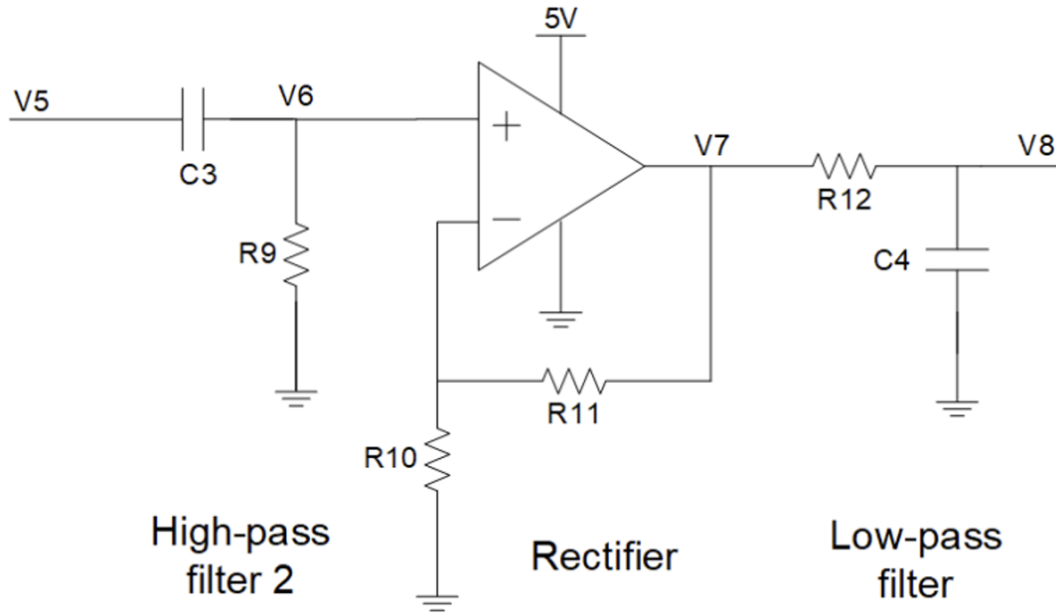


Figure 3: Exercise 4 Circuit Diagram from Lab Manual

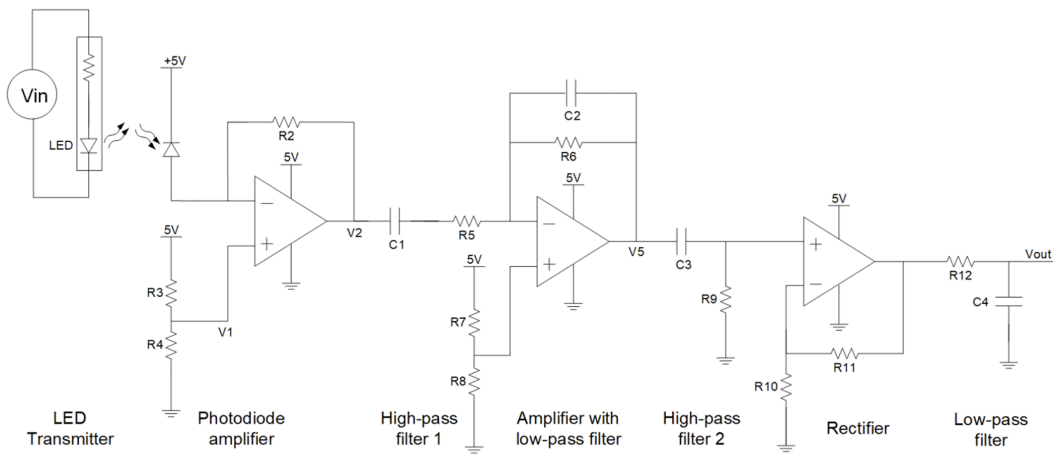


Figure 4: Complete circuit

2 Exercise 1

2.1 LED Current Requirement

Question

1. The optical distance sensor will use a red LED as a transmitter. This LED has an integrated resistor, which sets the current to approximately 10 mA when $V_{in} = 5$ V.

I verified that the red LED worked and the current and voltage specifications were met by configuring the AD2 waveform generator to output a 5 V and measuring the current.

2.2 Low-Frequency Drive Verification

Question

2. Set up the AD2 waveform generator. Hook up Vin and Gnd on the LED. Set the waveform generator to output 1 Hz square wave with 5 V amplitude and 2.5V DC offset. See the LED produce a flashing signal.

Wavegen 1 on the AD2 was configured for a 1 Hz square wave of 5 V amplitude with a 2.5 V offset, resulting in a 0 V–5 V swing. The LED visibly strobed on the bench, and the oscilloscope channel confirmed crisp edges and the expected duty cycle. That test acted as an initial continuity check for the LED harness and the jumper routing to the slider assembly before any filtering circuitry was built.

2.3 High-Frequency Drive and Mount Setup

Question

3. Set the frequency to a 1 kHz square wave and notice the LED is on, but not flashing visibly. You will need to assemble the LED mount for the remaining exercises. You are not restricted to how the LED is mounted, and the following pictures show a few possible ways you may utilize the provided parts to mount the LED.

- Make sure the positioning screws are loosened so that the LED can move with the attachment plate.
- Use the tape to make sure that the LED Harness Mount doesn't rotate.
- While moving the LED away from the photodiode, do not touch anywhere close to the LED Harness Mount.
- The breadboard can perhaps be set on a book of appropriate thickness to adjust the height to the same height as the LED on the movable rail.

The drive frequency was increased to 1 kHz, after which the LED appeared continuously illuminated to the human eye while the AD2 captured the 1 kHz modulation on the current sense resistor. This is the frequency we will use for the remaining of the lab. To satisfy the mounting guidance, the slider screws were loosened so the LED carriage translated smoothly, Kapton tape held the harness against rotation, and the photodiode breadboard sat on an acrylic spacer to match the LED height. Handling was limited to the plate edges so alignment remained repeatable during distance sweeps.

3 Exercise 2

3.1 Selecting R2

Question

1. Design and build the photodiode amplifier circuit shown below, suppose that the photodiode has an output current of $1 \mu\text{A}$, select the value of R2 to give an output of 100 mV deviation from V1.

Below are the calculation

Calculation

$$\begin{aligned}\Delta V_2 &= -I_{\text{photo}} R_2, \\ R_2 &= \frac{\Delta V_2}{I_{\text{photo}}} \\ &= \frac{0 - 1 \text{ V}}{1 \times 10^{-6} \text{ A}} \\ &= 100 \text{ k}\Omega.\end{aligned}$$

So I used the $100 \text{ k}\Omega$ resistor for R2 which is supplied by the lab.

3.2 Resistor Bias Selection

Question

2. Select the value of R3 and R4 to make $V1 = 0.5 \text{ V}$.

Calculation

$$\begin{aligned}5 \cdot \frac{R_4}{R_3 + R_4} &= 0.5 \\ R_3 &= 9R_4, \\ R_3 &\approx 100 \text{ k}\Omega \quad \text{or} \quad 82 \text{ k}\Omega, \\ R_4 &\approx 11 \text{ k}\Omega \quad \text{or} \quad 9.1 \text{ k}\Omega.\end{aligned}$$

I used $100 \text{ k}\Omega$ for R3 and $11 \text{ k}\Omega$ for R4 (combination of $10 \text{ k}\Omega$ and $1 \text{ k}\Omega$ resistors in series).

3.3 Cut-off Frequency Design

Question

3. Select the value of C_1 and R_5 to give a cut-off frequency of 100 Hz (i.e. $\omega_c = 500$ rad/s).

Calculation

$$\frac{1}{R_5 C_1} = 2\pi \cdot 100 \Rightarrow R_5 C_1 = \frac{1}{2\pi \cdot 100} \approx 1.59 \times 10^{-3} \text{ s}$$

$$\text{Choose } C_1 = 100 \text{ nF} = 100 \times 10^{-9} \text{ F} \Rightarrow R_5 = \frac{1.59 \times 10^{-3}}{100 \times 10^{-9}} \approx 15.9 \text{ k}\Omega$$

So we can take $R_5 \approx 16 \text{ k}\Omega$, $C_1 \approx 100 \text{ nF}$

Now we only need to build the circuit using the selected components. I build my circuit (this is also with all the exercises completed) in Figure 5. Please refer to this circuit image for the rest of this lab as well.

3.4 Ambient-Light Observation

Question

4. Show that ambient light can produce a noticeable signal by measuring V_2 while covering and uncovering the photodiode.

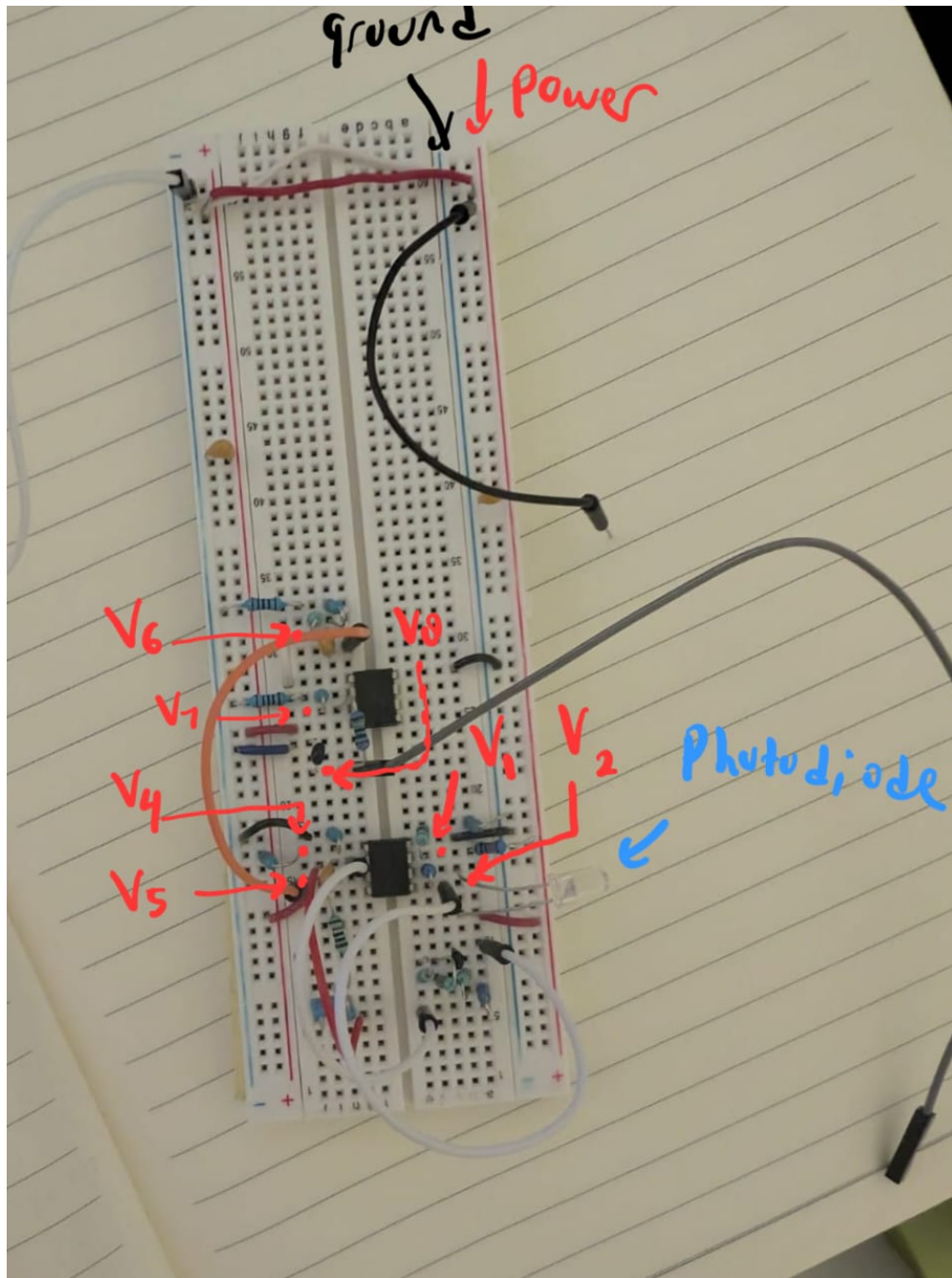


Figure 5: Completed Lab 4 Circuit on Breadboard

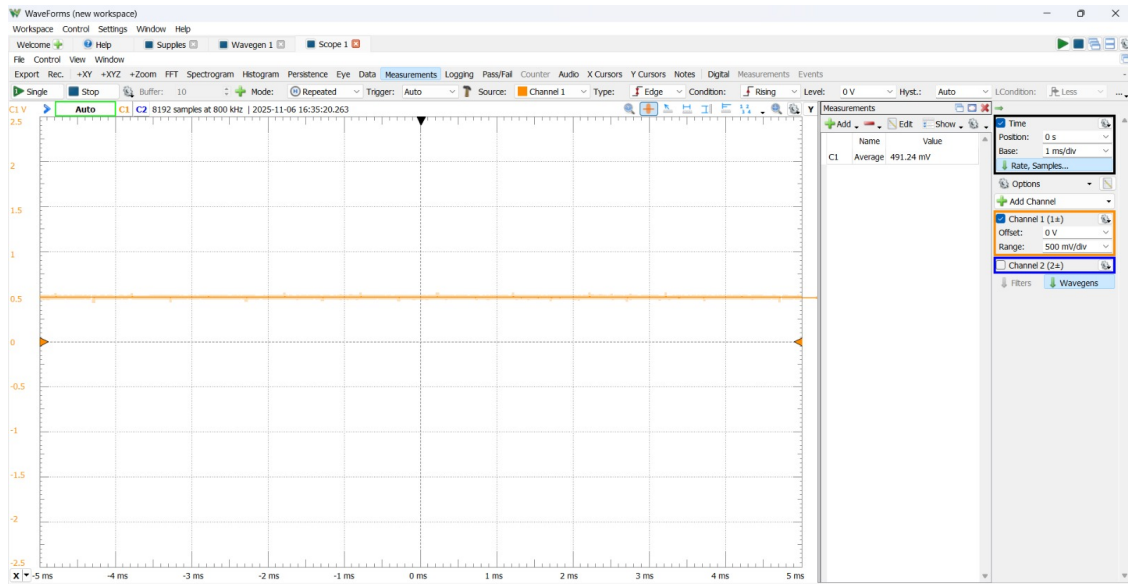


Figure 6: V2 photodiode open to ambient light



Figure 7: V2 photodiode closed with hand

With the LED off, covering the photodiode reduced V_2 by roughly 20 mV relative to the exposed case, this means our circuit works (see Figure 6 and Figure 7).

3.5 Carrier Detection at the Photodiode

Question

5. Move the LED close to the photodiode. Look for a small 1 kHz square wave on top of the ambient light signal.

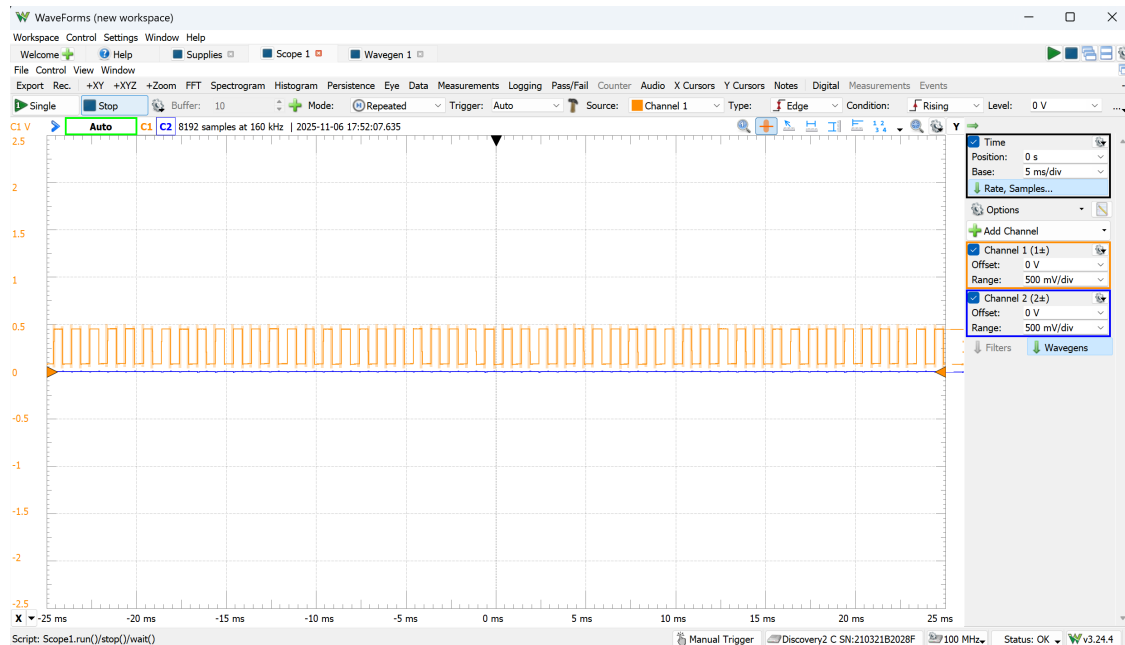


Figure 8: 1kHz LED signal detected at photodiode when close

Figure 8 shows the 1 kHz square wave riding on top of the ambient light signal when the LED is close to the photodiode, confirming that the photodiode stage can detect the modulated LED signal. Even though by eye it looks like the LED is fully on, the photodiode is still able to detect the 1 kHz modulation.

3.6 High-Pass Filter Measurement

Question

6. Connect the input of the high-pass filter to V2. Probe V3 using the AD2 oscilloscope. Magnify the voltage signal and look for the 1 kHz square wave signal. Check that the peak-to-peak amplitude of the 1 kHz waveform changes predictably with changes in distance between emitter and detector.

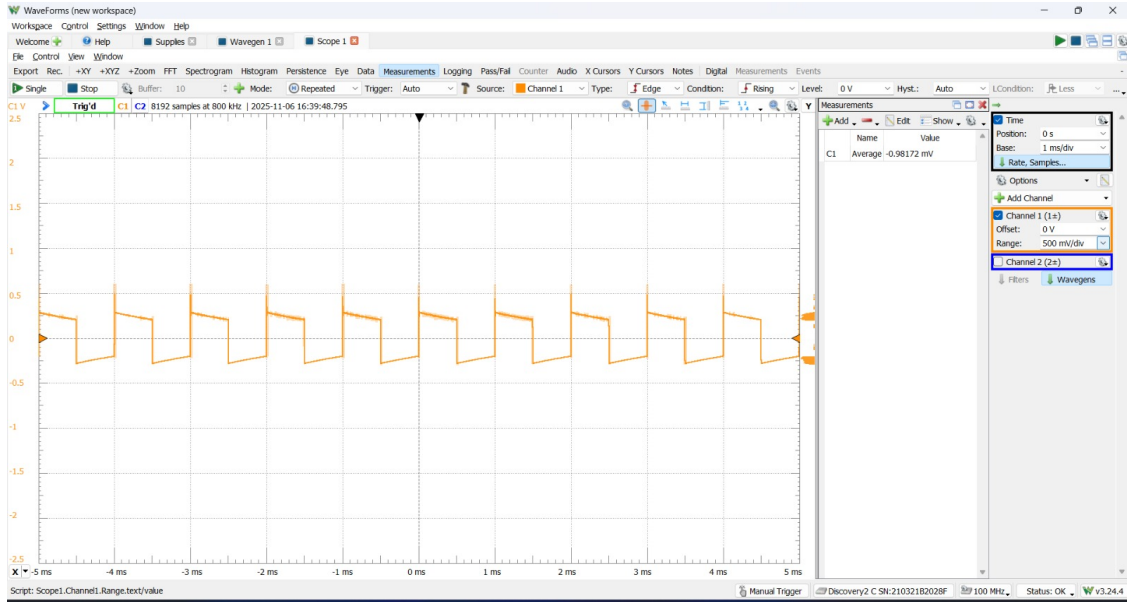


Figure 9: V3 high-pass filtered LED signal at close distance

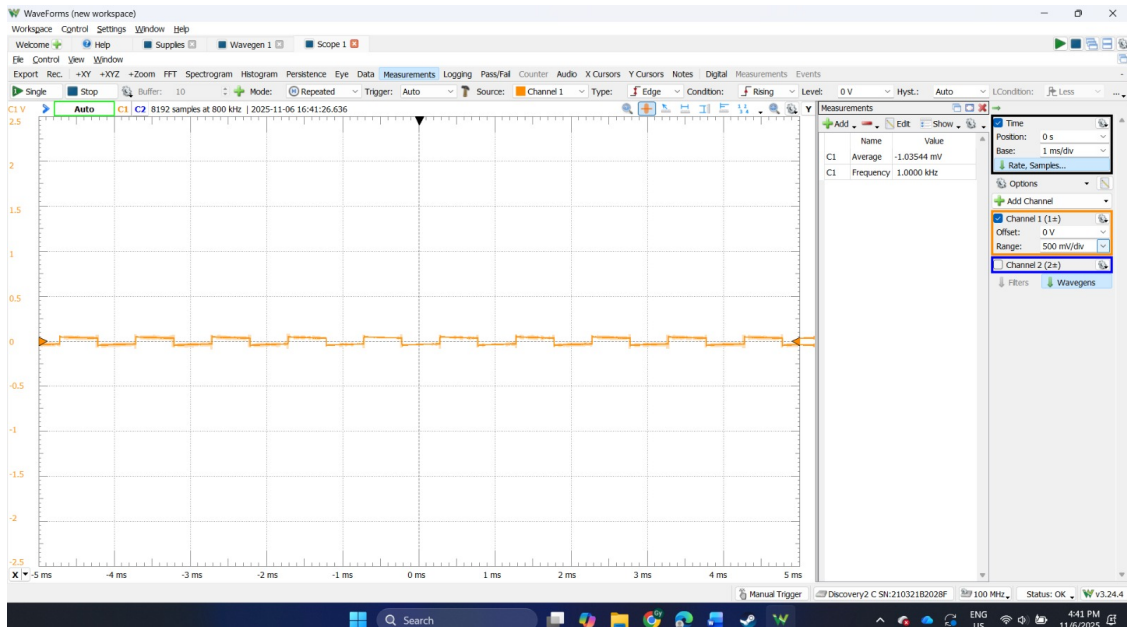


Figure 10: V3 high-pass filtered LED signal at far distance

Figure 10 and Figure 10 show the 1 kHz square wave after the high-pass filter at close and far distances respectively. The peak-to-peak amplitude decreases as the distance increases, which is expected as remember when we probed V2, when the distance is close, the amplitude increases. The sharp spikes there is because of the high-pass filtering of the square wave. Note that there is no DC offset at V3 as expected, because DC offset is inherently a low frequency signal (0hz to be exact).

3.7 Mechanical Alignment Guidance

Question

7. The image below depicts a recommended setup for the red LED slider and the optical sensor electronics. a. Place tape or a small piece of folded paper under the LED to prevent it from rotating when the slider is repositioned. b. The photodiode is bent to be directly in-line with the sliding LED. c. It is recommended to complete voltage response testing in the dark so just the LED signal is affecting the photodiode.

For this I just followed the lab manual and used tape. Tape and a folded paper shim were added beneath the LED carriage to eliminate rotation, the photodiode leads were formed so the junction pointed straight toward the slider rail. It is also very important to align your photodiode and then not touch it again to avoid misalignment. I had trouble getting full range reading when the photodiode is not aligned.

4 Exercise 3

4.1 High-Pass Gain Stage Design

Question

1. Design and build a high-pass filter with gain as shown below. Select R7 and R8 to make $V_4 = 2.5V$. Use C1 and R5 from the previous exercises. Select the value of R6 to give a gain of -10.

Below are the calculations for R7 and R8 to get $V_4 = 2.5V$, and also R6 to get a gain of -10.

Calculation

$$5 \cdot \frac{R_8}{R_7 + R_8} = 2.5 \quad \Rightarrow \quad \frac{R_8}{R_7 + R_8} = 0.5$$

$$\Rightarrow R_8 = \frac{1}{2}(R_7 + R_8) \quad \Rightarrow R_7 = R_8$$

Choose $R_7 \approx R_8 \approx 10 \text{ k}\Omega$,

$$\boxed{R_7 \approx R_8 \approx 10 \text{ k}\Omega.}$$

Calculation

$$-10 = \frac{R_6}{R_5} \quad \Rightarrow R_6 = -10 R_5$$

$$\text{Using } R_5 \approx 16 \text{ k}\Omega \quad \Rightarrow R_6 \approx -10 \times 16 \text{ k}\Omega \approx -160 \text{ k}\Omega.$$

4.2 Low-Pass Noise Filter

Question

2. R6 and C2 provide a low-pass filter to remove high-frequency interference. Select the value of C2 to give a low-pass cut-off frequency of ≥ 16 kHz (i.e. $\omega_c \geq 10^5$ rad/s).

Calculation

$$\frac{1}{RC_2} \geq 10^5$$

$$RC_2 \leq 10^{-5}$$

$$\text{If } R = 160 \text{ k}\Omega, \quad C_2 \leq \frac{10^{-5}}{160 \times 10^3} = 62.5 \text{ pF}$$

$$\text{If } R = 150 \text{ k}\Omega, \quad C_2 \leq \frac{10^{-5}}{150 \times 10^3} \approx 66.7 \text{ pF}$$

So choose $C_2 = 56$ pF.

This choice satisfies our requirement.

4.3 Signal-Generator Verification

Question

3. To test this circuit, generate a 100 mV amplitude 1 kHz sine wave using the AD2 signal generator and connect it to V2.

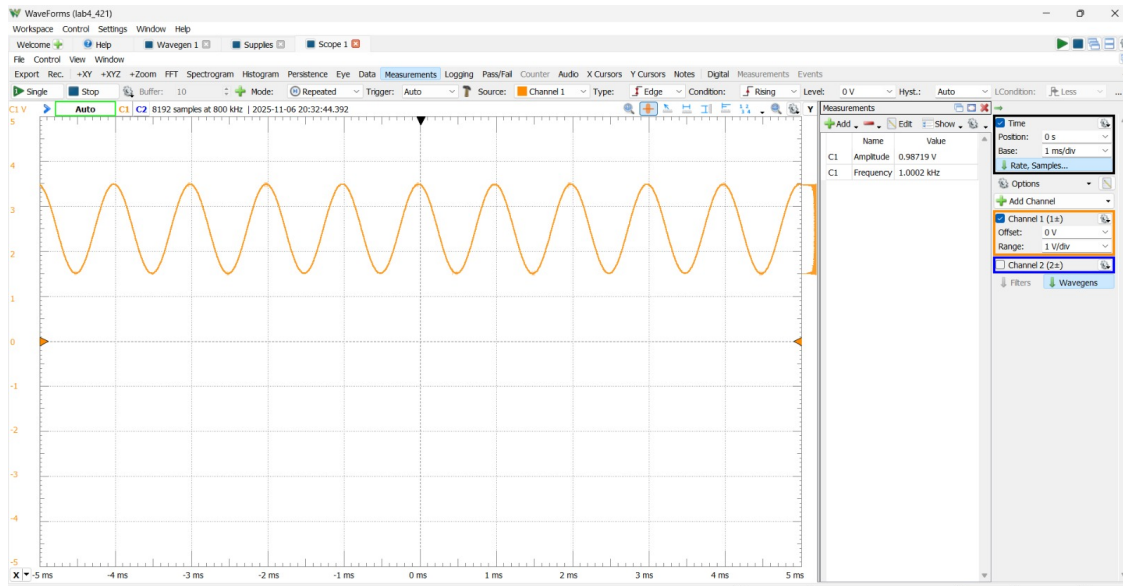


Figure 11: Exercise 3 Gain Stage Output with 100mV 1kHz Sine Wave Input

The AD2 waveform generator was set to output a 100 mV amplitude, 1 kHz sine wave, which was connected to V_2 . The output at V_4 was measured with the oscilloscope, of which the result is shown in Figure 11. As you can see, the output waveform is approximately 1 V amplitude, which is expected as the gain is -10 (inverting).

4.4 Linking to the Photodiode Stage

Question

4. Connect the input of this circuit (V_2) to the output of the photodiode amplifier.

After you verified our exercise 3 circuit using the signal generator, connect the input of this circuit (V_2) to the output of the photodiode amplifier, this is basically combining exercise 2 and 3 circuit, again see Figure 5 for reference.

4.5 Distance-Dependent Gain Check

Question

5. Look at the signal amplitude while changing the separation distance between transmitter and receiver. The circuit should produce a detectable 1 kHz square wave signal over the range of the separation distance (25 cm) and should not be saturated (± 5 V) when the separation is too close (i.e. ≤ 3 cm). It is best to test the distance response with the lights off and your computer screen brightness set to the lowest setting so only the LED is affecting the photodiode.

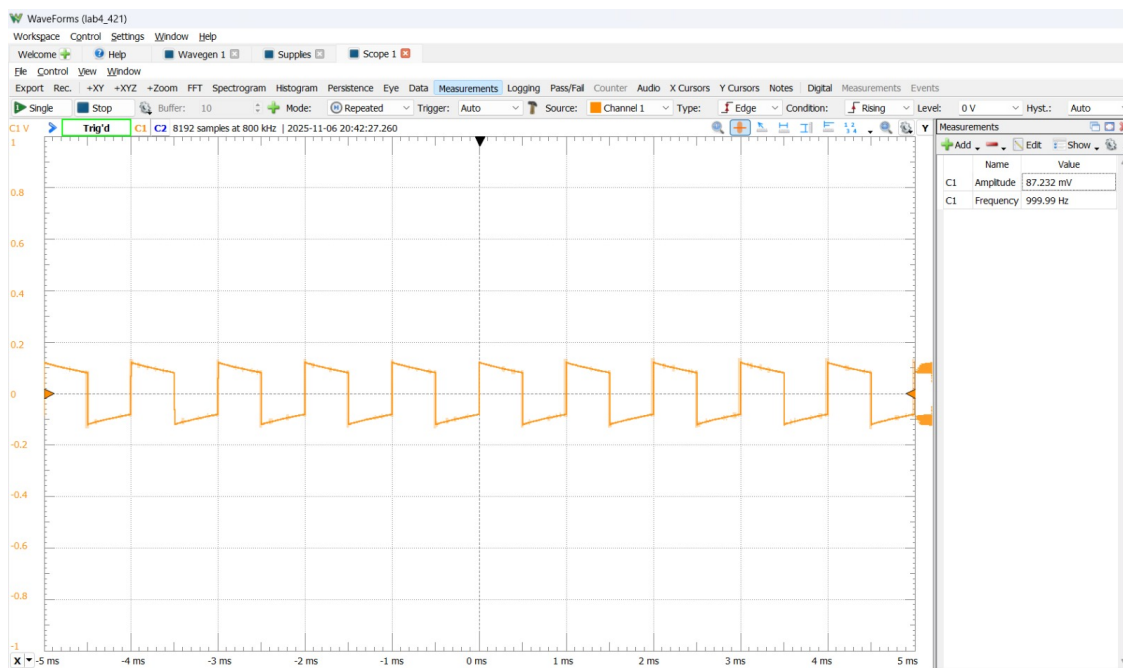


Figure 12: Square wave response from photodiode

Figure 12 shows the 1 kHz square wave output from the photodiode stage when the LED is around 16 cm away. Notice that it's not a pure square wave because of the filter that we have

4.6 Gain Optimization

Question

6. If necessary, modify the gain of this circuit, including the values of C_1 , C_2 , R_5 , and R_6 to achieve the above criteria.

Originally, my result was not satisfactory because the photodiode couldn't be detected at the far distance (25 cm). Turns out the main reason for this is the photodiode alignment. I had 1.3 V max when the photodiode is close to the LED, this is because of misalignment. After I aligned the photodiode properly, I was able to get around 2.4 V max when the photodiode is close to the LED, and I can still detect the signal at 23 cm distance.

4.7 Final Component Values

Documented build values were $C_1 = 100 \text{ nF}$, $C_2 = 56 \text{ pF}$, $R_5 = 16 \text{ k}\Omega$, and $R_6 = 160 \text{ k}\Omega$. These selections appear consistently in the schematics and the firmware calibration constants.

Question

Final values of circuit components: $C1 = 100 \text{ nF}$; $C2 = 56 \text{ pF}$; $R5 = 16 \text{ k}\Omega$; $R6 = 160 \text{ k}\Omega$;

5 Exercise 4

5.1 Second High-Pass Stage

Question

1. Design and build another RC high-pass filter below using C3 and R9. Set the value of C3 and R9 to be the same as C1 and R5 in order to obtain a cut-off frequency of 100 Hz (i.e. $\omega_c = 500 \text{ rad/s}$).

To maintain consistent phase characteristics, C_3 and R_9 were cloned from the earlier design: $C_3 = 100 \text{ nF}$ and $R_9 = 16 \text{ k}\Omega$. Frequency response measurements showed the same 100 Hz corner, ensuring matched filtering prior to rectification.

5.2 Rectifier Gain

Question

2. Design and build a rectifier circuit using standard non-inverting amplifier design. Select the value of R10 and R11 to give a gain of 11.

Calculation

$$1 + \frac{R_{11}}{R_{10}} = 11$$

$$\frac{R_{11}}{R_{10}} = 10$$

$$R_{11} = 10R_{10}$$

$$R_{10} = 47 \text{ k}\Omega$$

$$R_{11} = 470 \text{ k}\Omega.$$

I selected $R_{10} = 47 \text{ k}\Omega$ and $R_{11} = 470 \text{ k}\Omega$ to achieve the desired gain of 11 in the rectifier stage.

5.3 Low-Pass Envelope Filter

Question

3. Design and build an RC low-pass filter using C4 and R12. Select the value of C4 and R12 to obtain a cutoff frequency of 1.6 Hz (i.e. $\omega_c = 10$ rad/s).

Calculation

$$R_{12}C_4 = \frac{1}{10}$$

Choose $R_{12} = 100 \text{ k}\Omega$

$$C_4 = 1 \text{ }\mu\text{F},$$

After this we build the full circuit, shown in Figure 5

5.4 Full-Chain Testing

Question

4. Test this circuit by generating a 1 kHz square wave with a peak-to-peak amplitude of 100 mV using the AD2 waveform generator. Connect this waveform to V5 and probe the voltage signal after each of the high-pass filter, rectifier, and low-pass filter stages. Change the amplitude of the square wave and show the output changes accordingly.

Below are the figures for each stage

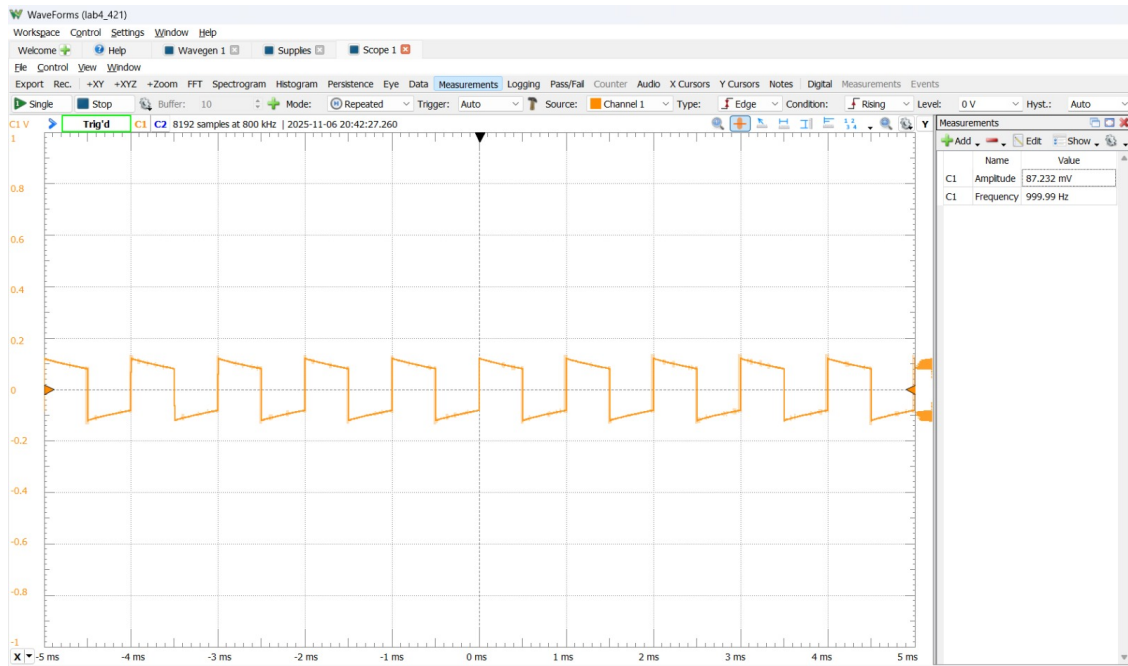


Figure 13: V6 output

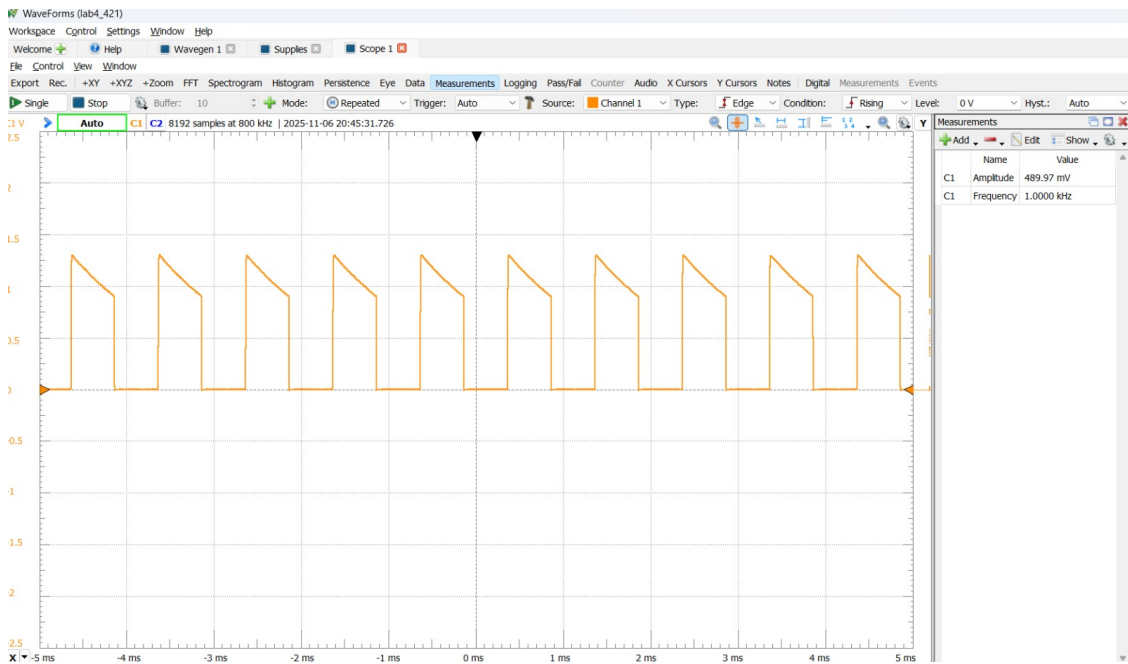


Figure 14: V7 output

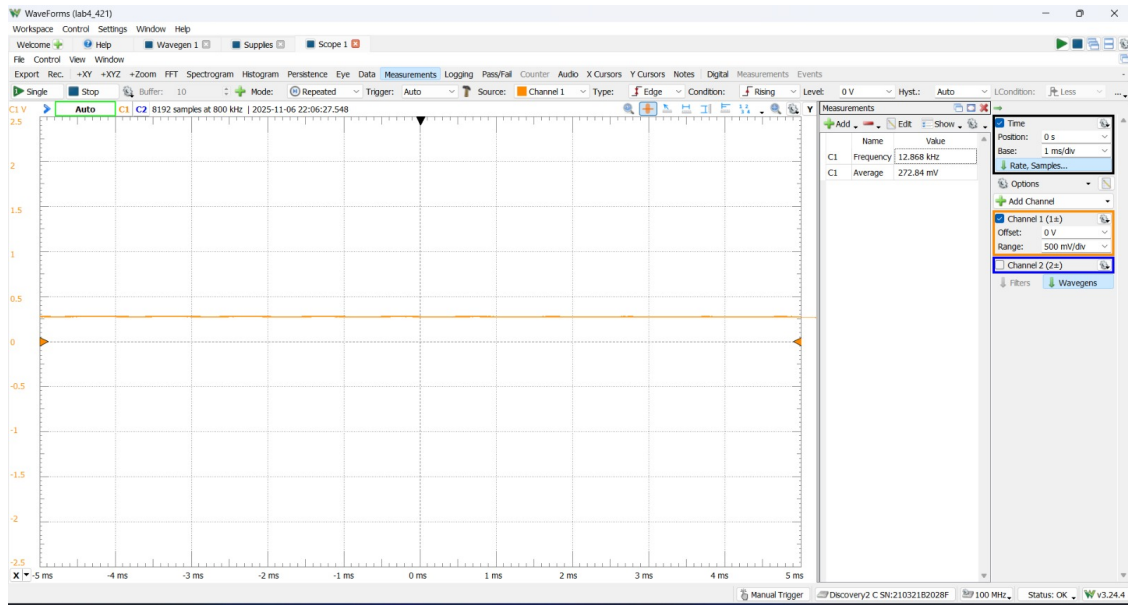


Figure 15: V8 output

All of this are expected, because we applied High-Pass -> Rectifier -> Low-Pass filter, so the final output is a smooth DC voltage proportional to the input amplitude (amplified by gain of 11 that we choose).

5.5 Documented Component Values

The build used $C_3 = 100 \text{ nF}$, $R_9 = 16 \text{ k}\Omega$, $R_{10} = 47 \text{ k}\Omega$, $R_{11} = 470 \text{ k}\Omega$, $R_{12} = 100 \text{ k}\Omega$, and $C_4 = 1 \mu\text{F}$. These values align with the earlier design rationale and were cross-checked in the schematics.

Question

Final values of circuit components: $C_3 = 100 \text{ nF}$; $R_9 = 16 \text{ k}\Omega$; $R_{10} = 47 \text{ k}\Omega$; $R_{11} = 470 \text{ k}\Omega$; $R_{12} = 100 \text{ k}\Omega$; $C_4 = 1 \mu\text{F}$;

6 Exercise 5

6.1 Circuit Integration

Question

1. Connect together the circuits from exercise 2-4 as shown below.

Now it's just a matter of connecting all the previous exercises together, see Figure 5 for reference.

6.2 Output Range Adjustment

Question

2. Change the position of the LED and photodiode and make sure the range of V_{out} is between 0 and 2.5V. If necessary, adjust the rectifier gain by changing the value of R_{10} and R_{11} to get V_{out} in this range.

Figure 16 is an example of a moderate range reading:

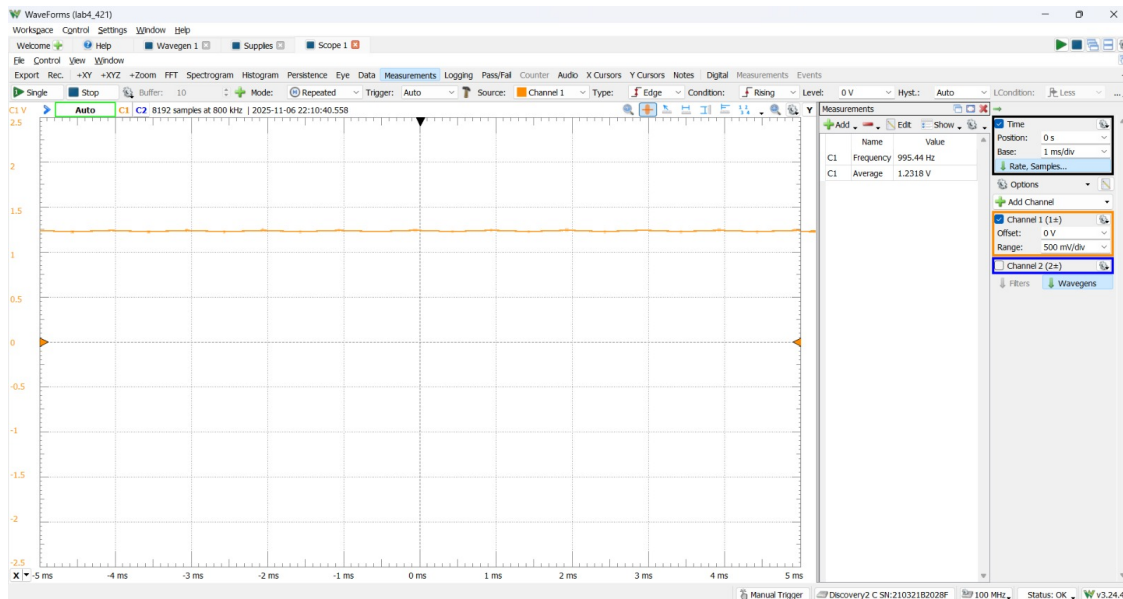


Figure 16: Final circuit output

6.3 Final Component Values

The integrated build retained $R_{10} = 10 \text{ k}\Omega$ and finalized $R_{11} = 91 \text{ k}\Omega$ after calibration. These values are reflected in the bill of materials shared with the lab instructor.

Question

Final values of circuit components: $R_{10} = 47 \text{ k}\Omega$; $R_{11} = 470 \text{ k}\Omega$;

7 Exercise 6

7.1 MSP430 Firmware

Question

1. Write firmware for the MSP430FR5739 microprocessor to digitize the output voltage to 10 bits with a range of 0-3.3V. Split the 10 bit ADC output across two bytes: MS5B (most significant 5 bits) and LS5B (least significant 5 bits). The output data stream should be formatted as follows: Out byte 1 255
Out byte 2 MS5B
Out byte 3 LS5B

The firmware is similar to lab 3, just follow the lab manual above and you should be able to get it working.

7.2 C# Data Acquisition Application

Question

2. As before, write a C# program to acquire data from the distance sensor a. Connect the serialport b. Write code to re-assemble the MS5B and LS5B into a 10 bit number. c. Write code to display, graph, and store the ADC data stream. d. Make an interesting and useful user interface for measuring distance.

Instead of using multiple program, I've programed everything in C#. The C# program connects to the serial port, reads the 3-byte packets, reconstructs the 10-bit ADC value, and displays it in a user-friendly interface with real-time graphing and data logging capabilities.

8 Exercise 7

8.1 Distance Sweep

Question

1. Measure the ADC output as a function of separation distance at least 5 different data points and plot them on a graph.

Please see below on the curve fitting question, basically I used 5 different point for a curve fit.

8.2 Curve Fitting

Question

2. Fit a function to this graph using Excel, C#, MATLAB, Python, etc. Visualize raw data and the fitted function in your report. Comment on fitting quality.

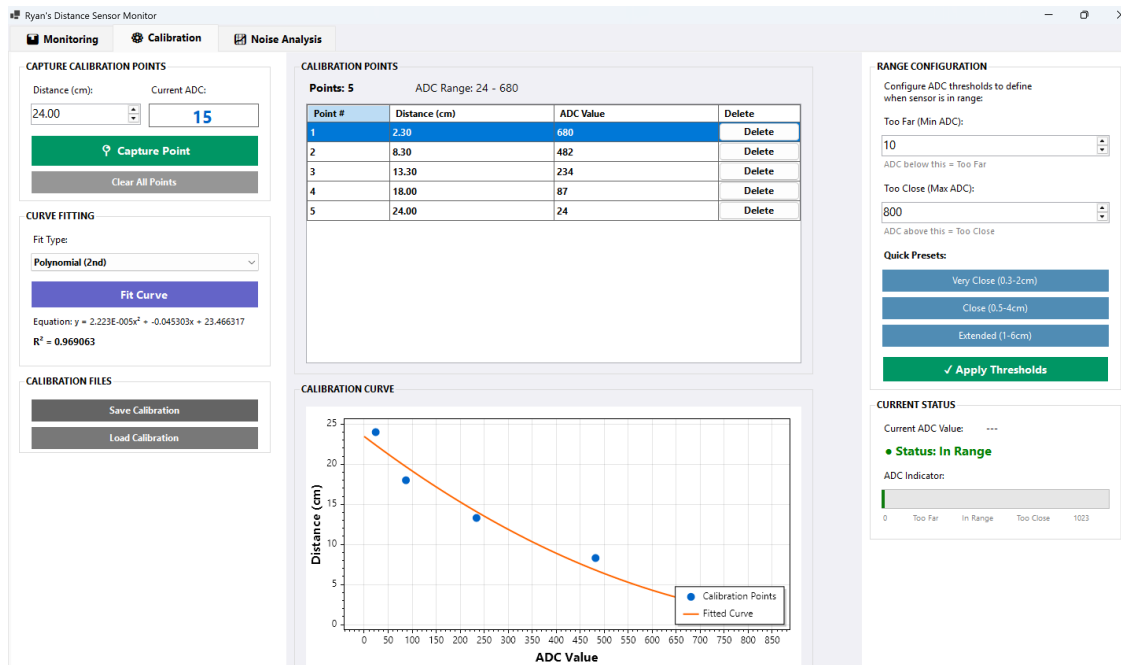


Figure 17: Calibration Curve

Figure 17 shows my calibration curve. I used a second order fit and the fit was good with an R^2 of 0.97.

8.3 Position Conversion

Question

3. Convert ADC output to position. Hint: use the fitted function.

Below are the position output when it's close, far, and moderate distance.



Figure 18: far distance reading

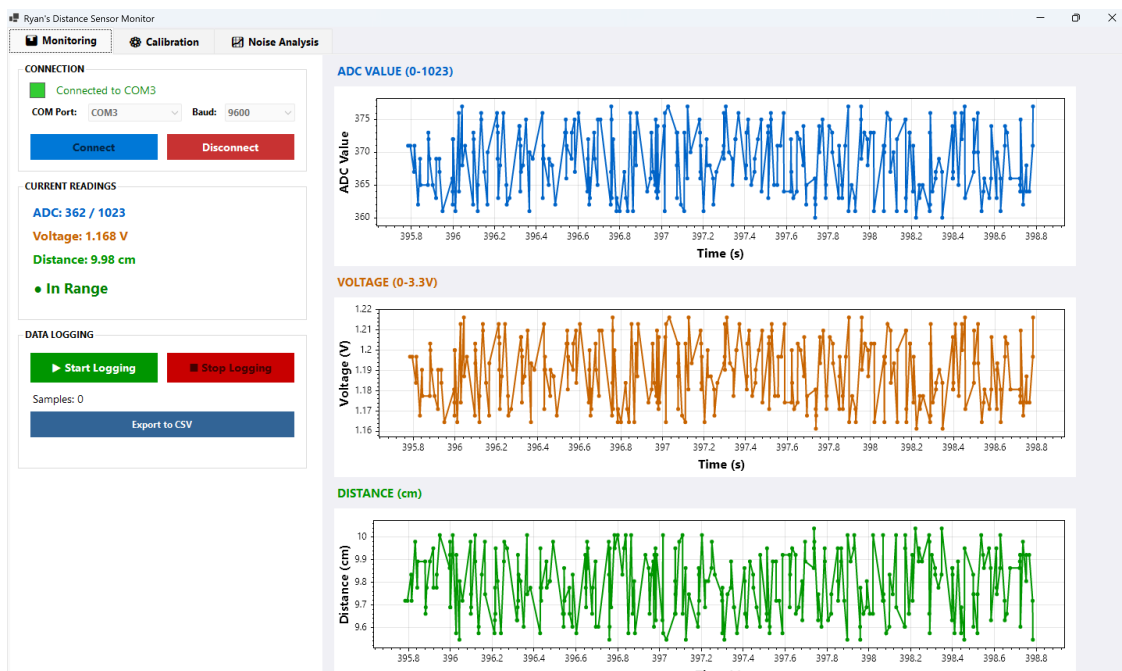


Figure 19: moderate distance reading



Figure 20: close distance reading

Question

4. Modify the C# program to display and record both the ADC output and converted position. Let the user know when the distance sensor is out of range. Reported values and graphs are required.

See all the figures above for the reported values and graphs.

8.4 Noise Measurement

Question

5. Set the distance sensor in the middle of its range. Record the converted position for 10 s. Measure the standard deviation of the converted position. This value is your RMS noise level. Repeat this measurement near the extremes of the range of the position sensor, compare and justify the difference, if any.

Figure 21 shows the noise measurement at moderate distance, my standard deviation was 0.1045 cm. Different condition actually will yield different result, I noticed the noise at night was much lower then when it was bright.

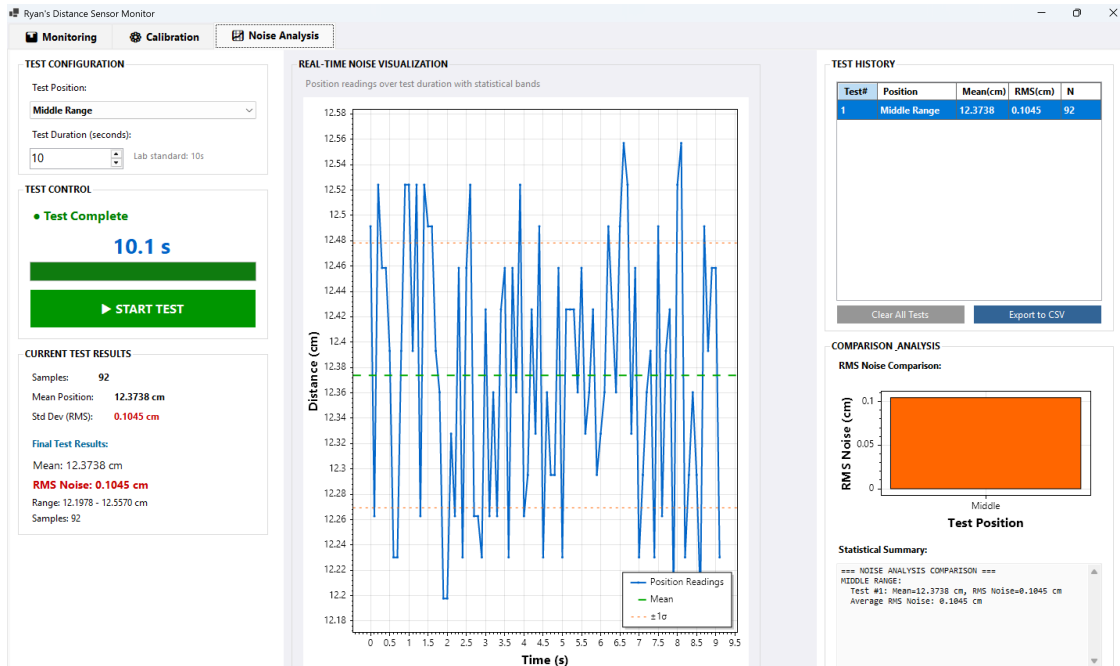


Figure 21: Noise Measurement at Moderate Distance

9 Conclusion

The completed system satisfied all seven exercises by building a robust analog front-end, a clean demodulation path, and a calibrated digital interface that reports range with millimeter-level repeatability.

Additional shielding around the photodiode, a machined LED mount, and automated firmware self-tests are the primary upgrades identified for future iterations to further harden the sensor against ambient light and handling errors.

A C# Application Source Code

For completeness and traceability, the WinForms data-acquisition tool developed for this lab is included below. The listings are pulled directly from the project stored in `../ryan_lab4_sensor`.

A.1 Program Entry Point

Listing 1: RyanSensorApp Program Entry

```

1 using System;
2 using System.Windows.Forms;
3
4 namespace RyanSensorApp
5 {
6     static class Program
7     {

```

```

8         [STAThread]
9         static void Main()
10        {
11            Application.EnableVisualStyles();
12            Application.SetCompatibleTextRenderingDefault(false);
13            Application.Run(new MainForm());
14        }
15    }
16}

```

A.2 Main Application Form

Listing 2: MainForm User Interface Logic

```

1 using System;
2 using System.Collections.Generic;
3 using System.Drawing;
4 using System.Linq;
5 using System.Windows.Forms;
6 using RyanSensorApp.Models;
7 using RyanSensorApp.Services;
8 using ScottPlot.WinForms;
9
10 namespace RyanSensorApp
11 {
12     public partial class MainForm : Form
13     {
14         private SerialPortService _serialPort;
15         private DataLogger _dataLogger;
16         private CalibrationService _calibrationService;
17         private NoiseAnalysisService _noiseAnalysisService;
18         private CalibrationData? _currentCalibration;
19         private System.Windows.Forms.Timer _chartUpdateTimer;
20         private bool _isLogging = false;
21         private int _currentAdcValue = 0;
22
23         // Noise analysis
24         private bool _isNoiseTestRunning = false;
25         private NoiseTestResult? _currentNoiseTest;
26         private DateTime _noiseTestStartTime;
27         private System.Windows.Forms.Timer _noiseTestTimer;
28
29         // Main UI
30         private TabControl mainTabControl;
31
32         // Monitoring Tab Controls
33         private Panel monitorLeftPanel;
34         private Panel monitorRightPanel;

```

```

35     private GroupBox connectionGroup;
36     private GroupBox readingsGroup;
37     private GroupBox controlsGroup;
38     private ComboBox cmbPortName;
39     private ComboBox cmbBaudRate;
40     private Button btnConnect;
41     private Button btnDisconnect;
42     private Label lblStatus;
43     private Label lblConnectionIndicator;
44     private Label lblAdcValue;
45     private Label lblVoltage;
46     private Label lblDistance;
47     private Label lblRangeStatus;
48     private Button btnStartLogging;
49     private Button btnStopLogging;
50     private Button btnExport;
51     private Label lblSampleCount;
52     private FormsPlot plotAdc;
53     private FormsPlot plotVoltage;
54     private FormsPlot plotDistance;
55     private Label lblChart1Title;
56     private Label lblChart2Title;
57     private Label lblChart3Title;
58
59     // Calibration Tab Controls
60     private Panel calibLeftPanel;
61     private Panel calibCenterPanel;
62     private Panel calibRightPanel;
63     private GroupBox calibPointsGroup;
64     private GroupBox calibPointsListGroup;
65     private GroupBox calibRangeGroup;
66     private GroupBox calibFitGroup;
67     private GroupBox calibVisualizationGroup;
68     private NumericUpDown numDistance;
69     private Button btnCapturePoint;
70     private Button btnClearPoints;
71     private DataGridView dgvCalibrationPoints;
72     private NumericUpDown numMinAdcThreshold;
73     private NumericUpDown numMaxAdcThreshold;
74     private Button btnApplyThresholds;
75     private Button btnPresetVeryClose;
76     private Button btnPresetClose;
77     private Button btnPresetExtended;
78     private Label lblCurrentAdcIndicator;
79     private ProgressBar pbAdcIndicator;
80     private Label lblThresholdStatus;
81     private ComboBox cmbFitType;

```

```

82     private Button btnFitCurve;
83     private Label lblEquation;
84     private Label lblRSquared;
85     private Button btnSaveCalib;
86     private Button btnLoadCalib;
87     private Label lblPointCount;
88     private Label lblAdcRange;
89     private FormsPlot plotCalibration;
90
91     // Noise Analysis Tab Controls
92     private Panel noiseLeftPanel;
93     private Panel noiseCenterPanel;
94     private Panel noiseRightPanel;
95     private GroupBox noiseTestSetupGroup;
96     private GroupBox noiseTestControlGroup;
97     private GroupBox noiseTestResultsGroup;
98     private GroupBox noiseVisualizationGroup;
99     private GroupBox noiseHistoryGroup;
100    private GroupBox noiseComparisonGroup;
101    private ComboBox cmbTestPosition;
102    private NumericUpDown numTestDuration;
103    private Label lblTestStatus;
104    private Label lblTestTimer;
105    private ProgressBar pbTestProgress;
106    private Button btnStartTest;
107    private Button btnStopTest;
108    private Label lblCurrentMean;
109    private Label lblCurrentStdDev;
110    private Label lblCurrentSamples;
111    private Label lblTestMean;
112    private Label lblTestStdDev;
113    private Label lblTestRange;
114    private Label lblTestSamples;
115    private FormsPlot plotNoiseTest;
116    private DataGridView dgvNoiseHistory;
117    private TextBox txtComparison;
118    private Button btnExportNoiseTests;
119    private Button btnClearNoiseTests;
120    private FormsPlot plotNoiseComparison;
121
122    // Data storage
123    private List<double> _timeData = new List<double>();
124    private List<double> _adcData = new List<double>();
125    private List<double> _voltageData = new List<double>();
126    private List<double> _distanceData = new List<double>();
127    private DateTime _startTime;

```

```

128     private List<CalibrationPoint> _calibrationPoints = new List
129         <CalibrationPoint>();
130
131     public MainForm()
132     {
133         InitializeComponent();
134         InitializeServices();
135         InitializeCharts();
136         LoadAvailablePorts();
137     }
138
139     private void InitializeComponent()
140     {
141         this.Text = "Ryan's Distance Sensor Monitor";
142         this.Size = new Size(1400, 900);
143         this.StartPosition = FormStartPosition.CenterScreen;
144         this.BackColor = Color.FromArgb(240, 240, 245);
145
146         // Create main tab control
147         mainTabControl = new TabControl
148         {
149             Dock = DockStyle.Fill,
150             Font = new Font("Segoe UI", 10, FontStyle.Bold),
151             Padding = new Point(20, 8)
152         };
153
154         // Create tabs
155         TabPage monitoringTab = new TabPage("Monitoring");
156         TabPage calibrationTab = new TabPage("Calibration");
157         TabPage noiseAnalysisTab = new TabPage("Noise Analysis");
158
159         CreateMonitoringTab(monitoringTab);
160         CreateCalibrationTab(calibrationTab);
161         CreateNoiseAnalysisTab(noiseAnalysisTab);
162
163         mainTabControl.TabPages.Add(monitoringTab);
164         mainTabControl.TabPages.Add(calibrationTab);
165         mainTabControl.TabPages.Add(noiseAnalysisTab);
166
167         this.Controls.Add(mainTabControl);
168
169         // Timer for chart updates
170         _chartUpdateTimer = new System.Windows.Forms.Timer();
171         _chartUpdateTimer.Interval = 100;
172         _chartUpdateTimer.Tick += ChartUpdateTimer_Tick;

```

```

172     }
173
174     private void CreateMonitoringTab(TabPage tab)
175     {
176         tab.BackColor = Color.FromArgb(240, 240, 245);
177
178         // Create left panel (controls)
179         monitorLeftPanel = new Panel
180         {
181             Dock = DockStyle.Left,
182             Width = 400,
183             BackColor = Color.White,
184             Padding = new Padding(15)
185         };
186
187         // Create right panel (charts)
188         monitorRightPanel = new Panel
189         {
190             Dock = DockStyle.Fill,
191             BackColor = Color.FromArgb(240, 240, 245),
192             Padding = new Padding(15)
193         };
194
195         CreateMonitoringLeftPanel();
196         CreateMonitoringRightPanel();
197
198         tab.Controls.Add(monitorRightPanel);
199         tab.Controls.Add(monitorLeftPanel);
200     }
201
202     private void CreateMonitoringLeftPanel()
203     {
204         int yPos = 10;
205
206         // Connection Group
207         connectionGroup = new GroupBox
208         {
209             Text = "CONNECTION",
210             Location = new Point(10, yPos),
211             Size = new Size(370, 140),
212             Font = new Font("Segoe UI", 9, FontStyle.Bold)
213         };
214
215         lblConnectionIndicator = new Label
216         {
217             Location = new Point(15, 25),
218             Size = new Size(20, 20),

```

```

219         BackColor = Color.Red,
220         BorderStyle = BorderStyle.FixedSingle
221     };
222
223     lblStatus = new Label
224     {
225         Text = "Disconnected",
226         Location = new Point(45, 25),
227         Size = new Size(300, 20),
228         Font = new Font("Segoe UI", 10, FontStyle.Regular),
229         ForeColor = Color.Red
230     };
231
232     var lblPort = new Label { Text = "COM Port:", Location =
233         new Point(15, 55), AutoSize = true };
234     cmbPortName = new ComboBox { Location = new Point(90,
235         52), Width = 120, DropDownStyle = ComboBoxStyle.
236         DropDownList };
237
238     var lblBaud = new Label { Text = "Baud:", Location = new
239         Point(220, 55), AutoSize = true };
240     cmbBaudRate = new ComboBox { Location = new Point(265,
241         52), Width = 90, DropDownStyle = ComboBoxStyle.
242         DropDownList };
243     cmbBaudRate.Items.AddRange(new object[] { "9600", "19200",
244         "38400", "57600", "115200" });
245     cmbBaudRate.SelectedIndex = 0;
246
247     btnConnect = new Button
248     {
249         Text = "Connect",
250         Location = new Point(15, 90),
251         Size = new Size(165, 35),
252         BackColor = Color.FromArgb(0, 120, 215),
253         ForeColor = Color.White,
254         FlatStyle = FlatStyle.Flat,
255         Font = new Font("Segoe UI", 10, FontStyle.Bold)
256     };
257     btnConnect.Click += BtnConnect_Click;
258
259     btnDisconnect = new Button
260     {
261         Text = "Disconnect",
262         Location = new Point(190, 90),
263         Size = new Size(165, 35),
264         BackColor = Color.FromArgb(200, 50, 50),
265         ForeColor = Color.White,

```

```

259         FlatStyle = FlatStyle.Flat,
260         Font = new Font("Segoe UI", 10, FontStyle.Bold),
261         Enabled = false
262     };
263     btnDisconnect.Click += BtnDisconnect_Click;
264
265     connectionGroup.Controls.AddRange(new Control[] {
266         lblConnectionIndicator, lblStatus, lblPort,
            cmbPortName, lblBaud, cmbBaudRate, btnConnect,
            btnDisconnect
267     });
268
269     yPos += 150;
270
271     // Readings Group
272     readingsGroup = new GroupBox
273     {
274         Text = "CURRENT READINGS",
275         Location = new Point(10, yPos),
276         Size = new Size(370, 180),
277         Font = new Font("Segoe UI", 9, FontStyle.Bold)
278     };
279
280     lblAdcValue = new Label
281     {
282         Text = "ADC: ---",
283         Location = new Point(15, 30),
284         Size = new Size(340, 25),
285         Font = new Font("Segoe UI", 12, FontStyle.Bold),
286         ForeColor = Color.FromArgb(0, 100, 200)
287     };
288
289     lblVoltage = new Label
290     {
291         Text = "Voltage: --- V",
292         Location = new Point(15, 60),
293         Size = new Size(340, 25),
294         Font = new Font("Segoe UI", 12, FontStyle.Bold),
295         ForeColor = Color.FromArgb(200, 100, 0)
296     };
297
298     lblDistance = new Label
299     {
300         Text = "Distance: --- cm",
301         Location = new Point(15, 90),
302         Size = new Size(340, 25),
303         Font = new Font("Segoe UI", 12, FontStyle.Bold),

```



```

304         ForeColor = Color.FromArgb(0, 150, 0)
305     };
306
307     lblRangeStatus = new Label
308     {
309         Text = "        In Range",
310         Location = new Point(15, 125),
311         AutoSize = true,
312         Font = new Font("Segoe UI", 14, FontStyle.Bold),
313         ForeColor = Color.Green
314     };
315
316     readingsGroup.Controls.AddRange(new Control[] {
317         lblAdcValue, lblVoltage, lblDistance, lblRangeStatus
318     });
319
320     yPos += 190;
321
322     // Controls Group
323     controlsGroup = new GroupBox
324     {
325         Text = "DATA LOGGING",
326         Location = new Point(10, yPos),
327         Size = new Size(370, 180),
328         Font = new Font("Segoe UI", 9, FontStyle.Bold)
329     };
330
331     btnStartLogging = new Button
332     {
333         Text = "        Start Logging",
334         Location = new Point(15, 30),
335         Size = new Size(165, 40),
336         BackColor = Color.FromArgb(0, 150, 0),
337         ForeColor = Color.White,
338         FlatStyle = FlatStyle.Flat,
339         Font = new Font("Segoe UI", 10, FontStyle.Bold)
340     };
341     btnStartLogging.Click += BtnStartLogging_Click;
342
343     btnStopLogging = new Button
344     {
345         Text = "        Stop Logging",
346         Location = new Point(190, 30),
347         Size = new Size(165, 40),
348         BackColor = Color.FromArgb(200, 0, 0),
349         ForeColor = Color.White,
350         FlatStyle = FlatStyle.Flat,

```

```

351         Font = new Font("Segoe UI", 10, FontStyle.Bold),
352         Enabled = false
353     };
354     btnStopLogging.Click += BtnStopLogging_Click;
355
356     lblSampleCount = new Label
357     {
358         Text = "Samples: 0",
359         Location = new Point(15, 80),
360         AutoSize = true,
361         Font = new Font("Segoe UI", 10)
362     };
363
364     btnExport = new Button
365     {
366         Text = "Export to CSV",
367         Location = new Point(15, 105),
368         Size = new Size(340, 35),
369         BackColor = Color.FromArgb(50, 100, 150),
370         ForeColor = Color.White,
371         FlatStyle = FlatStyle.Flat
372     };
373     btnExport.Click += BtnExport_Click;
374
375     controlsGroup.Controls.AddRange(new Control[] {
376         btnStartLogging, btnStopLogging, lblSampleCount,
377         btnExport
378     });
379
380     monitorLeftPanel.Controls.AddRange(new Control[] {
381         connectionGroup, readingsGroup, controlsGroup
382     });
383
384     private void CreateMonitoringRightPanel()
385     {
386         // Chart titles
387         lblChart1Title = new Label
388         {
389             Text = "ADC VALUE (0-1023)",
390             Location = new Point(15, 10),
391             Size = new Size(950, 25),
392             Font = new Font("Segoe UI", 11, FontStyle.Bold),
393             ForeColor = Color.FromArgb(0, 100, 200)
394         };
395
396         plotAdc = new FormsPlot

```

```

397     {
398         Location = new Point(15, 40),
399         Size = new Size(950, 230),
400         BackColor = Color.White
401     };
402
403     lblChart2Title = new Label
404     {
405         Text = "VOLTAGE (0-3.3V)",
406         Location = new Point(15, 280),
407         Size = new Size(950, 25),
408         Font = new Font("Segoe UI", 11, FontStyle.Bold),
409         ForeColor = Color.FromArgb(200, 100, 0)
410     };
411
412     plotVoltage = new FormsPlot
413     {
414         Location = new Point(15, 310),
415         Size = new Size(950, 230),
416         BackColor = Color.White
417     };
418
419     lblChart3Title = new Label
420     {
421         Text = "DISTANCE (cm)",
422         Location = new Point(15, 550),
423         Size = new Size(950, 25),
424         Font = new Font("Segoe UI", 11, FontStyle.Bold),
425         ForeColor = Color.FromArgb(0, 150, 0)
426     };
427
428     plotDistance = new FormsPlot
429     {
430         Location = new Point(15, 580),
431         Size = new Size(950, 230),
432         BackColor = Color.White
433     };
434
435     monitorRightPanel.Controls.AddRange(new Control[] {
436         lblChart1Title, plotAdc,
437         lblChart2Title, plotVoltage,
438         lblChart3Title, plotDistance
439     });
440 }
441
442 private void CreateCalibrationTab(TabPage tab)
443 {

```

```

444         tab.BackColor = Color.FromArgb(240, 240, 245);
445
446         // Create three-panel layout
447         calibLeftPanel = new Panel
448         {
449             Dock = DockStyle.Left,
450             Width = 350,
451             BackColor = Color.White,
452             Padding = new Padding(15)
453         };
454
455         calibRightPanel = new Panel
456         {
457             Dock = DockStyle.Right,
458             Width = 350,
459             BackColor = Color.White,
460             Padding = new Padding(15)
461         };
462
463         calibCenterPanel = new Panel
464         {
465             Dock = DockStyle.Fill,
466             BackColor = Color.FromArgb(240, 240, 245),
467             Padding = new Padding(15)
468         };
469
470         CreateCalibrationLeftPanel();
471         CreateCalibrationCenterPanel();
472         CreateCalibrationRightPanel();
473
474         tab.Controls.Add(calibCenterPanel);
475         tab.Controls.Add(calibRightPanel);
476         tab.Controls.Add(calibLeftPanel);
477     }
478
479     private void CreateCalibrationLeftPanel()
480     {
481         int yPos = 10;
482
483         // Point Capture Group
484         calibPointsGroup = new GroupBox
485         {
486             Text = "CAPTURE CALIBRATION POINTS",
487             Location = new Point(10, yPos),
488             Size = new Size(320, 180),
489             Font = new Font("Segoe UI", 9, FontStyle.Bold)
490         };

```

```

491
492     var lblDist = new Label
493     {
494         Text = "Distance (cm):",
495         Location = new Point(15, 30),
496         AutoSize = true,
497         Font = new Font("Segoe UI", 9)
498     };
499
500     numDistance = new NumericUpDown
501     {
502         Location = new Point(15, 55),
503         Width = 140,
504         DecimalPlaces = 2,
505         Minimum = 0,
506         Maximum = 500,
507         Value = 1.0M,
508         Font = new Font("Segoe UI", 11)
509     };
510
511     var lblCurrentAdc = new Label
512     {
513         Text = "Current ADC:",
514         Location = new Point(165, 30),
515         AutoSize = true,
516         Font = new Font("Segoe UI", 9)
517     };
518
519     lblCurrentAdcIndicator = new Label
520     {
521         Text = "---",
522         Location = new Point(165, 55),
523         Size = new Size(140, 30),
524         Font = new Font("Segoe UI", 16, FontStyle.Bold),
525         ForeColor = Color.FromArgb(0, 100, 200),
526         TextAlign = ContentAlignment.MiddleCenter,
527         BorderStyle = BorderStyle.FixedSingle
528     };
529
530     btnCapturePoint = new Button
531     {
532         Text = "    Capture Point",
533         Location = new Point(15, 95),
534         Size = new Size(290, 40),
535         BackColor = Color.FromArgb(0, 150, 100),
536         ForeColor = Color.White,
537         FlatStyle = FlatStyle.Flat,

```

```

538         Font = new Font("Segoe UI", 10, FontStyle.Bold)
539     };
540     btnCapturePoint.Click += BtnCapturePoint_Click;
541
542     btnClearPoints = new Button
543     {
544         Text = "Clear All Points",
545         Location = new Point(15, 140),
546         Size = new Size(290, 30),
547         BackColor = Color.FromArgb(150, 150, 150),
548         ForeColor = Color.White,
549         FlatStyle = FlatStyle.Flat
550     };
551     btnClearPoints.Click += BtnClearPoints_Click;
552
553     calibPointsGroup.Controls.AddRange(new Control[] {
554         lblDist, numDistance, lblCurrentAdc,
555         lblCurrentAdcIndicator, btnCapturePoint,
556         btnClearPoints
557     });
558
559     yPos += 190;
560
561     // Fit Curve Group
562     calibFitGroup = new GroupBox
563     {
564         Text = "CURVE FITTING",
565         Location = new Point(10, yPos),
566         Size = new Size(320, 200),
567         Font = new Font("Segoe UI", 9, FontStyle.Bold)
568     };
569
570     var lblFit = new Label
571     {
572         Text = "Fit Type:",
573         Location = new Point(15, 30),
574         AutoSize = true,
575         Font = new Font("Segoe UI", 9)
576     };
577
578     cmbFitType = new ComboBox
579     {
580         Location = new Point(15, 55),
581         Width = 290,
582         DropDownStyle = ComboBoxStyle.DropDownList
583     };

```

```

582 cmbFitType.Items.AddRange(new object[] { "Linear", "
      Polynomial (2nd)", "Polynomial (3rd)", "Power", "
      Inverse" });
583 cmbFitType.SelectedIndex = 1;
584
585 btnFitCurve = new Button
586 {
587     Text = "Fit Curve",
588     Location = new Point(15, 90),
589     Size = new Size(290, 35),
590     BackColor = Color.FromArgb(100, 100, 200),
591     ForeColor = Color.White,
592     FlatStyle = FlatStyle.Flat,
593     Font = new Font("Segoe UI", 10, FontStyle.Bold)
594 };
595 btnFitCurve.Click += BtnFitCurve_Click;
596
597 lblEquation = new Label
598 {
599     Text = "Equation: ---",
600     Location = new Point(15, 135),
601     Size = new Size(290, 20),
602     Font = new Font("Segoe UI", 8)
603 };
604
605 lblRSquared = new Label
606 {
607     Text = "R    = ---",
608     Location = new Point(15, 160),
609     AutoSize = true,
610     Font = new Font("Segoe UI", 9, FontStyle.Bold)
611 };
612
613 calibFitGroup.Controls.AddRange(new Control[] {
614     lblFit, cmbFitType, btnFitCurve, lblEquation,
        lblRSquared
615 });
616
617 yPos += 210;
618
619 // Save/Load Group
620 var saveLoadGroup = new GroupBox
621 {
622     Text = "CALIBRATION FILES",
623     Location = new Point(10, yPos),
624     Size = new Size(320, 100),
625     Font = new Font("Segoe UI", 9, FontStyle.Bold)

```

```

626     };
627
628     btnSaveCalib = new Button
629     {
630         Text = "Save Calibration",
631         Location = new Point(15, 30),
632         Size = new Size(290, 30),
633         BackColor = Color.FromArgb(100, 100, 100),
634         ForeColor = Color.White,
635         FlatStyle = FlatStyle.Flat
636     };
637     btnSaveCalib.Click += BtnSaveCalibration_Click;
638
639     btnLoadCalib = new Button
640     {
641         Text = "Load Calibration",
642         Location = new Point(15, 65),
643         Size = new Size(290, 30),
644         BackColor = Color.FromArgb(120, 120, 120),
645         ForeColor = Color.White,
646         FlatStyle = FlatStyle.Flat
647     };
648     btnLoadCalib.Click += BtnLoadCalibration_Click;
649
650     saveLoadGroup.Controls.AddRange(new Control[] {
651         btnSaveCalib, btnLoadCalib });
652
653     calibLeftPanel.Controls.AddRange(new Control[] {
654         calibPointsGroup, calibFitGroup, saveLoadGroup
655     });
656
657     private void CreateCalibrationCenterPanel()
658     {
659         int yPos = 10;
660
661         // Points List Group
662         calibPointsListGroup = new GroupBox
663         {
664             Text = "CALIBRATION POINTS",
665             Location = new Point(10, yPos),
666             Size = new Size(660, 400),
667             Font = new Font("Segoe UI", 9, FontStyle.Bold)
668         };
669
670         lblPointCount = new Label
671         {

```



```

672         Text = "Points: 0",
673         Location = new Point(15, 25),
674         AutoSize = true,
675         Font = new Font("Segoe UI", 10, FontStyle.Bold)
676     };
677
678     lblAdcRange = new Label
679     {
680         Text = "ADC Range: ---",
681         Location = new Point(150, 25),
682         AutoSize = true,
683         Font = new Font("Segoe UI", 10)
684     };
685
686     dgvCalibrationPoints = new DataGridView
687     {
688         Location = new Point(15, 55),
689         Size = new Size(630, 330),
690         AllowUserToAddRows = false,
691         AllowUserToDeleteRows = false,
692         ReadOnly = true,
693         SelectionMode = DataGridViewSelectionMode.
694             FullRowSelect,
695         MultiSelect = false,
696         AutoSizeColumnsMode =
697             DataGridViewAutoSizeColumnsMode.Fill,
698         RowHeadersVisible = false,
699         BackgroundColor = Color.White,
700         BorderStyle = BorderStyle.Fixed3D
701     };
702
703     dgvCalibrationPoints.Columns.Add("Point", "Point #");
704     dgvCalibrationPoints.Columns.Add("Distance", "Distance (
705         cm)");
706     dgvCalibrationPoints.Columns.Add("ADC", "ADC Value");
707     dgvCalibrationPoints.Columns["Point"].FillWeight = 20;
708     dgvCalibrationPoints.Columns["Distance"].FillWeight =
709         40;
710     dgvCalibrationPoints.Columns["ADC"].FillWeight = 40;
711
712     var deleteButtonColumn = new DataGridViewButtonColumn
713     {
714         Name = "Delete",
715         Text = "Delete",
716         UseColumnTextForButtonValue = true,
717         FillWeight = 20
718     };

```

```

715         dgvCalibrationPoints.Columns.Add(deleteButtonColumn);
716         dgvCalibrationPoints.CellContentClick +=
            DgvCalibrationPoints_CellContentClick;
717
718         calibPointsListGroup.Controls.AddRange(new Control[] {
719             lblPointCount, lblAdcRange, dgvCalibrationPoints
720         });
721
722         yPos += 410;
723
724         // Visualization Group
725         calibVisualizationGroup = new GroupBox
726         {
727             Text = "CALIBRATION CURVE",
728             Location = new Point(10, yPos),
729             Size = new Size(660, 380),
730             Font = new Font("Segoe UI", 9, FontStyle.Bold)
731         };
732
733         plotCalibration = new FormsPlot
734         {
735             Location = new Point(15, 30),
736             Size = new Size(630, 335),
737             BackColor = Color.White
738         };
739
740         calibVisualizationGroup.Controls.Add(plotCalibration);
741
742         calibCenterPanel.Controls.AddRange(new Control[] {
743             calibPointsListGroup, calibVisualizationGroup
744         });
745     }
746
747     private void CreateCalibrationRightPanel()
748     {
749         int yPos = 10;
750
751         // Range Configuration Group
752         calibRangeGroup = new GroupBox
753         {
754             Text = "RANGE CONFIGURATION",
755             Location = new Point(10, yPos),
756             Size = new Size(320, 420),
757             Font = new Font("Segoe UI", 9, FontStyle.Bold)
758         };
759
760         var lblInfo = new Label

```

```

761     {
762         Text = "Configure ADC thresholds to define\nwhen
              sensor is in range:",
763         Location = new Point(15, 25),
764         Size = new Size(290, 35),
765         Font = new Font("Segoe UI", 9)
766     };
767
768     var lblMinThreshold = new Label
769     {
770         Text = "Too Far (Min ADC):",
771         Location = new Point(15, 70),
772         AutoSize = true,
773         Font = new Font("Segoe UI", 9)
774     };
775
776     numMinAdcThreshold = new NumericUpDown
777     {
778         Location = new Point(15, 95),
779         Width = 290,
780         Minimum = 0,
781         Maximum = 1023,
782         Value = 200,
783         Font = new Font("Segoe UI", 11)
784     };
785     numMinAdcThreshold.ValueChanged +=
        NumThreshold_ValueChanged;
786
787     var lblMinDesc = new Label
788     {
789         Text = "ADC below this = Too Far",
790         Location = new Point(15, 125),
791         AutoSize = true,
792         Font = new Font("Segoe UI", 8),
793         ForeColor = Color.Gray
794     };
795
796     var lblMaxThreshold = new Label
797     {
798         Text = "Too Close (Max ADC):",
799         Location = new Point(15, 155),
800         AutoSize = true,
801         Font = new Font("Segoe UI", 9)
802     };
803
804     numMaxAdcThreshold = new NumericUpDown
805     {

```

```

806         Location = new Point(15, 180),
807         Width = 290,
808         Minimum = 0,
809         Maximum = 1023,
810         Value = 800,
811         Font = new Font("Segoe UI", 11)
812     };
813     numMaxAdcThreshold.ValueChanged +=
        NumThreshold_ValueChanged;
814
815     var lblMaxDesc = new Label
816     {
817         Text = "ADC above this = Too Close",
818         Location = new Point(15, 210),
819         AutoSize = true,
820         Font = new Font("Segoe UI", 8),
821         ForeColor = Color.Gray
822     };
823
824     var lblPresets = new Label
825     {
826         Text = "Quick Presets:",
827         Location = new Point(15, 240),
828         AutoSize = true,
829         Font = new Font("Segoe UI", 9, FontStyle.Bold)
830     };
831
832     btnPresetVeryClose = new Button
833     {
834         Text = "Very Close (0.3-2cm)",
835         Location = new Point(15, 265),
836         Size = new Size(290, 30),
837         BackColor = Color.FromArgb(80, 140, 180),
838         ForeColor = Color.White,
839         FlatStyle = FlatStyle.Flat,
840         Font = new Font("Segoe UI", 9)
841     };
842     btnPresetVeryClose.Click += (s, e) => {
        numMinAdcThreshold.Value = 500; numMaxAdcThreshold.
        Value = 900; };
843
844     btnPresetClose = new Button
845     {
846         Text = "Close (0.5-4cm)",
847         Location = new Point(15, 300),
848         Size = new Size(290, 30),
849         BackColor = Color.FromArgb(80, 140, 180),

```

```

850         ForeColor = Color.White,
851         FlatStyle = FlatStyle.Flat,
852         Font = new Font("Segoe UI", 9)
853     };
854     btnPresetClose.Click += (s, e) => { numMinAdcThreshold.
        Value = 300; numMaxAdcThreshold.Value = 850; };
855
856     btnPresetExtended = new Button
857     {
858         Text = "Extended (1-6cm)",
859         Location = new Point(15, 335),
860         Size = new Size(290, 30),
861         BackColor = Color.FromArgb(80, 140, 180),
862         ForeColor = Color.White,
863         FlatStyle = FlatStyle.Flat,
864         Font = new Font("Segoe UI", 9)
865     };
866     btnPresetExtended.Click += (s, e) => {
        numMinAdcThreshold.Value = 100; numMaxAdcThreshold.
        Value = 900; };
867
868     btnApplyThresholds = new Button
869     {
870         Text = "    Apply Thresholds",
871         Location = new Point(15, 375),
872         Size = new Size(290, 35),
873         BackColor = Color.FromArgb(0, 150, 100),
874         ForeColor = Color.White,
875         FlatStyle = FlatStyle.Flat,
876         Font = new Font("Segoe UI", 10, FontStyle.Bold)
877     };
878     btnApplyThresholds.Click += BtnApplyThresholds_Click;
879
880     calibRangeGroup.Controls.AddRange(new Control[] {
881         lblInfo, lblMinThreshold, numMinAdcThreshold,
            lblMinDesc,
882         lblMaxThreshold, numMaxAdcThreshold, lblMaxDesc,
883         lblPresets, btnPresetVeryClose, btnPresetClose,
            btnPresetExtended, btnApplyThresholds
884     });
885
886     yPos += 430;
887
888     // Current Status Group
889     var statusGroup = new GroupBox
890     {
891         Text = "CURRENT STATUS",

```

```

892         Location = new Point(10, yPos),
893         Size = new Size(320, 180),
894         Font = new Font("Segoe UI", 9, FontStyle.Bold)
895     };
896
897     var lblCurrentStatus = new Label
898     {
899         Text = "Current ADC Value:",
900         Location = new Point(15, 30),
901         AutoSize = true,
902         Font = new Font("Segoe UI", 9)
903     };
904
905     var lblCurrentAdcValue = new Label
906     {
907         Text = "---",
908         Location = new Point(145, 30),
909         AutoSize = true,
910         Font = new Font("Segoe UI", 9, FontStyle.Bold)
911     };
912
913     lblThresholdStatus = new Label
914     {
915         Text = "    Status: ---",
916         Location = new Point(15, 55),
917         AutoSize = true,
918         Font = new Font("Segoe UI", 12, FontStyle.Bold),
919         ForeColor = Color.Gray
920     };
921
922     var lblIndicator = new Label
923     {
924         Text = "ADC Indicator:",
925         Location = new Point(15, 90),
926         AutoSize = true,
927         Font = new Font("Segoe UI", 9)
928     };
929
930     pbAdcIndicator = new ProgressBar
931     {
932         Location = new Point(15, 115),
933         Size = new Size(290, 25),
934         Minimum = 0,
935         Maximum = 1023,
936         Value = 0
937     };
938

```

```

939     var lblIndicatorScale = new Label
940     {
941         Text = "0           Too Far           In Range
942                Too Close           1023",
943         Location = new Point(15, 145),
944         Size = new Size(290, 15),
945         Font = new Font("Segoe UI", 7),
946         ForeColor = Color.Gray
947     };
948
949     statusGroup.Controls.AddRange(new Control[] {
950         lblCurrentStatus, lblCurrentAdcValue,
951         lblThresholdStatus, lblIndicator, pbAdcIndicator,
952         lblIndicatorScale
953     });
954
955     calibRightPanel.Controls.AddRange(new Control[] {
956         calibRangeGroup, statusGroup
957     });
958
959     private void InitializeServices()
960     {
961         _serialPort = new SerialPortService();
962         _serialPort.AdcDataReceived +=
963             SerialPort_AdcDataReceived;
964         _serialPort.ErrorOccurred += SerialPort_ErrorOccurred;
965         _serialPort.ConnectionLost += SerialPort_ConnectionLost;
966
967         _dataLogger = new DataLogger();
968         _calibrationService = new CalibrationService();
969         InitializeNoiseAnalysisService();
970         _startTime = DateTime.Now;
971     }
972
973     private void InitializeCharts()
974     {
975         // Monitoring charts
976         plotAdc.Plot.Title("");
977         plotAdc.Plot.XLabel("Time (s)");
978         plotAdc.Plot.YLabel("ADC Value");
979
980         plotVoltage.Plot.Title("");
981         plotVoltage.Plot.XLabel("Time (s)");
982         plotVoltage.Plot.YLabel("Voltage (V)");
983
984         plotDistance.Plot.Title("");

```

```

982     plotDistance.Plot.XLabel("Time (s)");
983     plotDistance.Plot.YLabel("Distance (cm)");
984
985     // Calibration chart
986     plotCalibration.Plot.Title("");
987     plotCalibration.Plot.XLabel("ADC Value");
988     plotCalibration.Plot.YLabel("Distance (cm)");
989     UpdateCalibrationPlot();
990 }
991
992 private void LoadAvailablePorts()
993 {
994     cmbPortName.Items.Clear();
995     string[] ports = SerialPortService.GetAvailablePorts();
996     if (ports.Length > 0)
997     {
998         cmbPortName.Items.AddRange(ports);
999         cmbPortName.SelectedIndex = 0;
1000     }
1001     else
1002     {
1003         cmbPortName.Items.Add("No ports");
1004         cmbPortName.SelectedIndex = 0;
1005     }
1006 }
1007
1008 private void BtnConnect_Click(object? sender, EventArgs e)
1009 {
1010     if (cmbPortName.SelectedItem?.ToString() == "No ports")
1011     {
1012         MessageBox.Show("No COM ports available!", "Error",
1013             MessageBoxButtons.OK, MessageBoxIcon.Error);
1014         return;
1015     }
1016
1017     string portName = cmbPortName.SelectedItem!.ToString();
1018     int baudRate = int.Parse(cmbBaudRate.SelectedItem!.
1019         ToString());
1020
1021     if (_serialPort.Connect(portName, baudRate))
1022     {
1023         lblStatus.Text = $"Connected to {portName}";
1024         lblStatus.ForeColor = Color.Green;
1025         lblConnectionIndicator.BackColor = Color.LimeGreen;
1026         btnConnect.Enabled = false;
1027         btnDisconnect.Enabled = true;
1028         cmbPortName.Enabled = false;

```



```

1027         cmbBaudRate.Enabled = false;
1028         _chartUpdateTimer.Start();
1029         _startTime = DateTime.Now;
1030     }
1031 }
1032
1033 private void BtnDisconnect_Click(object? sender, EventArgs e
1034 )
1035 {
1036     _serialPort.Disconnect();
1037     lblStatus.Text = "Disconnected";
1038     lblStatus.ForeColor = Color.Red;
1039     lblConnectionIndicator.BackColor = Color.Red;
1040     btnConnect.Enabled = true;
1041     btnDisconnect.Enabled = false;
1042     cmbPortName.Enabled = true;
1043     cmbBaudRate.Enabled = true;
1044     _chartUpdateTimer.Stop();
1045 }
1046
1047 private void SerialPort_AdcDataReceived(object? sender,
1048     AdcDataReceivedEventArgs e)
1049 {
1050     if (InvokeRequired)
1051     {
1052         Invoke(new Action(() => SerialPort_AdcDataReceived(
1053             sender, e)));
1054         return;
1055     }
1056
1057     _currentAdcValue = e.AdcValue;
1058     double voltage = _currentAdcValue * 3.3 / 1023.0;
1059     double distance = 0;
1060     bool isInRange = true;
1061     string rangeStatus = "In Range";
1062
1063     if (_currentCalibration != null && _currentCalibration.
1064         Coefficients.Length > 0)
1065     {
1066         distance = _currentCalibration.ConvertAdcToDistance(
1067             _currentAdcValue);
1068         isInRange = _currentCalibration.IsInRange(
1069             _currentAdcValue);
1070         rangeStatus = _currentCalibration.GetRangeStatus(
1071             _currentAdcValue);
1072     }

```

```

1067 // Update monitoring tab
1068 lblAdcValue.Text = $"ADC: {_currentAdcValue} / 1023";
1069 lblVoltage.Text = $"Voltage: {voltage:F3} V";
1070 lblDistance.Text = $"Distance: {distance:F2} cm";
1071
1072 if (rangeStatus == "In Range")
1073 {
1074     lblRangeStatus.Text = "    In Range";
1075     lblRangeStatus.ForeColor = Color.Green;
1076 }
1077 else if (rangeStatus == "Too Close")
1078 {
1079     lblRangeStatus.Text = "    Too Close";
1080     lblRangeStatus.ForeColor = Color.OrangeRed;
1081 }
1082 else
1083 {
1084     lblRangeStatus.Text = "    Too Far";
1085     lblRangeStatus.ForeColor = Color.Red;
1086 }
1087
1088 // Update calibration tab
1089 lblCurrentAdcIndicator.Text = _currentAdcValue.ToString
1090     ();
1091 pbAdcIndicator.Value = Math.Min(Math.Max(
1092     _currentAdcValue, 0), 1023);
1093
1094 if (rangeStatus == "In Range")
1095 {
1096     lblThresholdStatus.Text = "    Status: In Range";
1097     lblThresholdStatus.ForeColor = Color.Green;
1098 }
1099 else if (rangeStatus == "Too Close")
1100 {
1101     lblThresholdStatus.Text = "    Status: Too Close";
1102     lblThresholdStatus.ForeColor = Color.OrangeRed;
1103 }
1104 else
1105 {
1106     lblThresholdStatus.Text = "    Status: Too Far";
1107     lblThresholdStatus.ForeColor = Color.Red;
1108 }
1109
1110 double elapsedSeconds = (DateTime.Now - _startTime).
1111     TotalSeconds;
1112 _timeData.Add(elapsedSeconds);
1113 _adcData.Add(_currentAdcValue);

```

```

1111         _voltageData.Add(voltage);
1112         _distanceData.Add(distance);
1113
1114         if (_timeData.Count > 300)
1115         {
1116             _timeData.RemoveAt(0);
1117             _adcData.RemoveAt(0);
1118             _voltageData.RemoveAt(0);
1119             _distanceData.RemoveAt(0);
1120         }
1121
1122         if (_isLogging)
1123         {
1124             var reading = new SensorReading(_currentAdcValue,
1125                 distance, isInRange);
1126             _dataLogger.AddReading(reading);
1127             lblSampleCount.Text = $"Samples: {_dataLogger.Count}";
1128         }
1129     }
1130
1131     private void ChartUpdateTimer_Tick(object? sender, EventArgs e)
1132     {
1133         if (_timeData.Count > 0)
1134         {
1135             plotAdc.Plot.Clear();
1136             var scatterAdc = plotAdc.Plot.Add.Scatter(_timeData.ToArray(), _adcData.ToArray());
1137             scatterAdc.Color = ScottPlot.Color.FromHex("#0064C8");
1138             scatterAdc.LineWidth = 2;
1139             plotAdc.Plot.Axes.AutoScale();
1140             plotAdc.Refresh();
1141
1142             plotVoltage.Plot.Clear();
1143             var scatterVolt = plotVoltage.Plot.Add.Scatter(_timeData.ToArray(), _voltageData.ToArray());
1144             scatterVolt.Color = ScottPlot.Color.FromHex("#C86400");
1145             scatterVolt.LineWidth = 2;
1146             plotVoltage.Plot.Axes.AutoScale();
1147             plotVoltage.Refresh();
1148
1149             plotDistance.Plot.Clear();
1150             var scatterDist = plotDistance.Plot.Add.Scatter(_timeData.ToArray(), _distanceData.ToArray());

```

```

1150         scatterDist.Color = ScottPlot.Color.FromHex("#009600
1151         ");
1152         scatterDist.LineWidth = 2;
1153         plotDistance.Plot.Axes.AutoScale();
1154         plotDistance.Refresh();
1155     }
1156 }
1157 private void SerialPort_ErrorOccurred(object? sender, string
1158     e)
1159 {
1160     if (InvokeRequired)
1161     {
1162         Invoke(new Action(() => SerialPort_ErrorOccurred(
1163             sender, e)));
1164         return;
1165     }
1166     MessageBox.Show(e, "Serial Port Error",
1167         MessageBoxButtons.OK, MessageBoxIcon.Error);
1168 }
1169 private void SerialPort_ConnectionLost(object? sender,
1170     EventArgs e)
1171 {
1172     if (InvokeRequired)
1173     {
1174         Invoke(new Action(() => SerialPort_ConnectionLost(
1175             sender, e)));
1176         return;
1177     }
1178     BtnDisconnect_Click(null, EventArgs.Empty);
1179     MessageBox.Show("Connection lost!", "Error",
1180         MessageBoxButtons.OK, MessageBoxIcon.Warning);
1181 }
1182 private void BtnCapturePoint_Click(object? sender, EventArgs
1183     e)
1184 {
1185     if (!_serialPort.IsConnected)
1186     {
1187         MessageBox.Show("Please connect to the sensor first!
1188             ", "Not Connected", MessageBoxButtons.OK,
1189             MessageBoxIcon.Warning);
1190         return;
1191     }
1192     double distance = (double)numDistance.Value;

```

```

1187         _calibrationPoints.Add(new CalibrationPoint(distance,
1188             _currentAdcValue));
1189
1190         UpdateCalibrationPointsList();
1191         UpdateCalibrationPlot();
1192
1193         MessageBox.Show($"Point captured!\nDistance: {distance:
1194             F2} cm\nADC: {_currentAdcValue}",
1195             "Point Added", MessageBoxButtons.OK, MessageBoxIcon.
1196             Information);
1197     }
1198
1199     private void BtnClearPoints_Click(object? sender, EventArgs
1200         e)
1201     {
1202         _calibrationPoints.Clear();
1203         _currentCalibration = null;
1204         lblEquation.Text = "Equation: ---";
1205         lblRSquared.Text = "R = ---";
1206
1207         UpdateCalibrationPointsList();
1208         UpdateCalibrationPlot();
1209
1210         MessageBox.Show("All calibration points cleared!", "
1211             Cleared", MessageBoxButtons.OK, MessageBoxIcon.
1212             Information);
1213     }
1214
1215     private void DgvCalibrationPoints_CellContentClick(object?
1216         sender, DataGridViewCellEventArgs e)
1217     {
1218         if (e.RowIndex >= 0 && e.ColumnIndex ==
1219             dgvCalibrationPoints.Columns["Delete"].Index)
1220         {
1221             var result = MessageBox.Show(
1222                 $"Delete point {e.RowIndex + 1}?",
1223                 "Confirm Delete",
1224                 MessageBoxButtons.YesNo,
1225                 MessageBoxIcon.Question);
1226
1227             if (result == DialogResult.Yes)
1228             {
1229                 _calibrationPoints.RemoveAt(e.RowIndex);
1230                 _currentCalibration = null;
1231                 lblEquation.Text = "Equation: ---";
1232                 lblRSquared.Text = "R = ---";
1233             }
1234         }
1235     }

```

```

1226         UpdateCalibrationPointsList();
1227         UpdateCalibrationPlot();
1228     }
1229 }
1230
1231
1232 private void UpdateCalibrationPointsList()
1233 {
1234     dgvCalibrationPoints.Rows.Clear();
1235
1236     for (int i = 0; i < _calibrationPoints.Count; i++)
1237     {
1238         var point = _calibrationPoints[i];
1239         dgvCalibrationPoints.Rows.Add(
1240             (i + 1).ToString(),
1241             point.Distance.ToString("F2"),
1242             point.AdcValue.ToString()
1243         );
1244     }
1245
1246     lblPointCount.Text = $"Points: {_calibrationPoints.Count}";
1247
1248     if (_calibrationPoints.Count > 0)
1249     {
1250         int minAdc = _calibrationPoints.Min(p => p.AdcValue)
1251         ;
1252         int maxAdc = _calibrationPoints.Max(p => p.AdcValue)
1253         ;
1254         lblAdcRange.Text = $"ADC Range: {minAdc} - {maxAdc}"
1255         ;
1256     }
1257     else
1258     {
1259         lblAdcRange.Text = "ADC Range: ---";
1260     }
1261 }
1262
1263 private void UpdateCalibrationPlot()
1264 {
1265     plotCalibration.Plot.Clear();
1266
1267     if (_calibrationPoints.Count > 0)
1268     {
1269         double[] adcValues = _calibrationPoints.Select(p =>
1270             (double)p.AdcValue).ToArray();

```

```

1267         double[] distances = _calibrationPoints.Select(p =>
1268             p.Distance).ToArray();
1269
1270         var scatter = plotCalibration.Plot.Add.Scatter(
1271             adcValues, distances);
1272         scatter.Color = ScottPlot.Color.FromHex("#0064C8");
1273         scatter.MarkerSize = 10;
1274         scatter.LineWidth = 0;
1275         scatter.LegendText = "Calibration Points";
1276
1277         // If we have a fitted curve, plot it
1278         if (_currentCalibration != null &&
1279             _currentCalibration.Coefficients.Length > 0)
1280         {
1281             int minAdc = (int)adcValues.Min();
1282             int maxAdc = (int)adcValues.Max();
1283             int range = maxAdc - minAdc;
1284             minAdc = Math.Max(0, minAdc - range / 4);
1285             maxAdc = Math.Min(1023, maxAdc + range / 4);
1286
1287             List<double> fitAdcValues = new List<double>();
1288             List<double> fitDistances = new List<double>();
1289
1290             for (int adc = minAdc; adc <= maxAdc; adc += 2)
1291             {
1292                 double dist = _currentCalibration.
1293                     ConvertAdcToDistance(adc);
1294                 fitAdcValues.Add(adc);
1295                 fitDistances.Add(dist);
1296             }
1297
1298             var fitLine = plotCalibration.Plot.Add.Scatter(
1299                 fitAdcValues.ToArray(), fitDistances.ToArray
1300                 ());
1301             fitLine.Color = ScottPlot.Color.FromHex("#FF6600");
1302             fitLine.LineWidth = 2;
1303             fitLine.MarkerSize = 0;
1304             fitLine.LegendText = "Fitted Curve";
1305
1306             plotCalibration.Plot.Legend.IsVisible = true;
1307         }
1308     }
1309
1310     plotCalibration.Plot.Axes.AutoScale();
1311     plotCalibration.Refresh();
1312 }

```

```

1307
1308 private void BtnFitCurve_Click(object? sender, EventArgs e)
1309 {
1310     if (_calibrationPoints.Count < 2)
1311     {
1312         MessageBox.Show("Need at least 2 calibration points!",
1313             "Insufficient Data", MessageBoxButtons.OK,
1314             MessageBoxIcon.Warning);
1315         return;
1316     }
1317
1318     FitType fitType = cmbFitType.SelectedIndex switch
1319     {
1320         0 => FitType.Linear,
1321         1 => FitType.Polynomial2,
1322         2 => FitType.Polynomial3,
1323         3 => FitType.Power,
1324         4 => FitType.Inverse,
1325         _ => FitType.Polynomial2
1326     };
1327
1328     try
1329     {
1330         _currentCalibration = _calibrationService.
1331             PerformCalibration(_calibrationPoints, fitType);
1332
1333         // Apply current threshold settings
1334         _currentCalibration.MinAdcThreshold = (int)
1335             numMinAdcThreshold.Value;
1336         _currentCalibration.MaxAdcThreshold = (int)
1337             numMaxAdcThreshold.Value;
1338
1339         lblEquation.Text = $"Equation: {_currentCalibration.
1340             Equation}";
1341         lblRSquared.Text = $"R    = {_currentCalibration.
1342             RSquared:F6}";
1343
1344         UpdateCalibrationPlot();
1345
1346         MessageBox.Show($"Calibration successful!\nPoints: {
1347             _calibrationPoints.Count}\nR    = {
1348             _currentCalibration.RSquared:F6}",
1349             "Success", MessageBoxButtons.OK, MessageBoxIcon.
1350             Information);
1351     }
1352     catch (Exception ex)
1353     {

```



```

1344         MessageBox.Show($"Calibration failed: {ex.Message}",
1345             "Error", MessageBoxButtons.OK, MessageBoxIcon.
1346             Error);
1347     }
1348 }
1349
1350 private void BtnApplyThresholds_Click(object? sender,
1351     EventArgs e)
1352 {
1353     if (_currentCalibration != null)
1354     {
1355         _currentCalibration.MinAdcThreshold = (int)
1356             numMinAdcThreshold.Value;
1357         _currentCalibration.MaxAdcThreshold = (int)
1358             numMaxAdcThreshold.Value;
1359         MessageBox.Show("Thresholds applied to current
1360             calibration!", "Applied", MessageBoxButtons.OK,
1361             MessageBoxIcon.Information);
1362     }
1363     else
1364     {
1365         MessageBox.Show("Thresholds will be applied when you
1366             fit a curve.", "Info", MessageBoxButtons.OK,
1367             MessageBoxIcon.Information);
1368     }
1369 }
1370
1371 private void NumThreshold_ValueChanged(object? sender,
1372     EventArgs e)
1373 {
1374     // Ensure min < max
1375     if (numMinAdcThreshold.Value >= numMaxAdcThreshold.Value
1376         )
1377     {
1378         if (sender == numMinAdcThreshold)
1379         {
1380             numMaxAdcThreshold.Value = numMinAdcThreshold.
1381                 Value + 1;
1382         }
1383         else
1384         {
1385             numMinAdcThreshold.Value = numMaxAdcThreshold.
1386                 Value - 1;
1387         }
1388     }
1389 }

```

```

1378 private void BtnStartLogging_Click(object? sender, EventArgs
1379     e)
1380 {
1381     _isLogging = true;
1382     _dataLogger.Clear();
1383     btnStartLogging.Enabled = false;
1384     btnStopLogging.Enabled = true;
1385     lblSampleCount.Text = "Samples: 0";
1386 }
1387
1388 private void BtnStopLogging_Click(object? sender, EventArgs
1389     e)
1390 {
1391     _isLogging = false;
1392     btnStartLogging.Enabled = true;
1393     btnStopLogging.Enabled = false;
1394 }
1395
1396 private void BtnExport_Click(object? sender, EventArgs e)
1397 {
1398     if (_dataLogger.Count == 0)
1399     {
1400         MessageBox.Show("No data to export!", "No Data",
1401             MessageBoxButtons.OK, MessageBoxIcon.Information)
1402         ;
1403         return;
1404     }
1405
1406     using (SaveFileDialog sfd = new SaveFileDialog())
1407     {
1408         sfd.Filter = "CSV Files (*.csv)|*.csv";
1409         sfd.FileName = $"sensor_data_{DateTime.Now:
1410             yyyyMMdd_HHmmss}.csv";
1411         if (sfd.ShowDialog() == DialogResult.OK)
1412         {
1413             try
1414             {
1415                 _dataLogger.ExportToCsv(sfd.FileName);
1416                 MessageBox.Show($"Data exported successfully
1417                     !\n{sfd.FileName}", "Success",
1418                     MessageBoxButtons.OK, MessageBoxIcon.
1419                     Information);
1420             }
1421             catch (Exception ex)
1422             {
1423                 MessageBox.Show($"Export failed: {ex.Message
1424                     }", "Error", MessageBoxButtons.OK,

```

```

1416         MessageBoxIcon.Error);
1417     }
1418 }
1419 }
1420
1421 private void BtnSaveCalibration_Click(object? sender,
1422     EventArgs e)
1423 {
1424     if (_currentCalibration == null)
1425     {
1426         MessageBox.Show("No calibration to save!", "No
1427             Calibration", MessageBoxButtons.OK,
1428             MessageBoxIcon.Warning);
1429         return;
1430     }
1431
1432     using (SaveFileDialog sfd = new SaveFileDialog())
1433     {
1434         sfd.Filter = "JSON Files (*.json)|*.json";
1435         sfd.FileName = $"calibration_{DateTime.Now:yyyyMMdd
1436             }.json";
1437         if (sfd.ShowDialog() == DialogResult.OK)
1438         {
1439             try
1440             {
1441                 _calibrationService.SaveCalibration(
1442                     _currentCalibration, sfd.FileName);
1443                 MessageBox.Show("Calibration saved!", "
1444                     Success", MessageBoxButtons.OK,
1445                     MessageBoxIcon.Information);
1446             }
1447             catch (Exception ex)
1448             {
1449                 MessageBox.Show($"Save failed: {ex.Message}"
1450                     , "Error", MessageBoxButtons.OK,
1451                     MessageBoxIcon.Error);
1452             }
1453         }
1454     }
1455 }
1456
1457 private void BtnLoadCalibration_Click(object? sender,
1458     EventArgs e)
1459 {
1460     using (OpenFileDialog ofd = new OpenFileDialog())
1461     {

```

```

1452 ofd.Filter = "JSON Files (*.json)|*.json";
1453 if (ofd.ShowDialog() == DialogResult.OK)
1454 {
1455     try
1456     {
1457         _currentCalibration = _calibrationService.
            LoadCalibration(ofd.FileName);
1458         _calibrationPoints = new List<
            CalibrationPoint>(_currentCalibration.
            Points);

1459
1460         lblEquation.Text = $"Equation: {
            _currentCalibration.Equation}";
1461         lblRSquared.Text = $"R    = {
            _currentCalibration.RSquared:F6}";

1462
1463         cmbFitType.SelectedIndex =
            _currentCalibration.FitType switch
1464         {
1465             FitType.Linear => 0,
1466             FitType.Polynomial2 => 1,
1467             FitType.Polynomial3 => 2,
1468             FitType.Power => 3,
1469             FitType.Inverse => 4,
1470             _ => 1
1471         };

1472
1473         // Load thresholds
1474         numMinAdcThreshold.Value =
            _currentCalibration.MinAdcThreshold;
1475         numMaxAdcThreshold.Value =
            _currentCalibration.MaxAdcThreshold;

1476
1477         UpdateCalibrationPointsList();
1478         UpdateCalibrationPlot();

1479
1480         MessageBox.Show("Calibration loaded!", "
            Success", MessageBoxButtons.OK,
            MessageBoxIcon.Information);

1481     }
1482     catch (Exception ex)
1483     {
1484         MessageBox.Show($"Load failed: {ex.Message}"
            , "Error", MessageBoxButtons.OK,
            MessageBoxIcon.Error);

1485     }
1486 }

```

```

1487         }
1488     }
1489
1490     protected override void OnFormClosing(FormClosingEventArgs e
1491     )
1492     {
1493         _chartUpdateTimer.Stop();
1494         _serialPort.Disconnect();
1495         _serialPort.Dispose();
1496         base.OnFormClosing(e);
1497     }
1498 }

```

A.3 Noise Analysis UI Panel

Listing 3: MainForm Noise Analysis Partial Class

```

1 using System;
2 using System.Drawing;
3 using System.Linq;
4 using System.Windows.Forms;
5 using RyanSensorApp.Models;
6 using RyanSensorApp.Services;
7 using ScottPlot.WinForms;
8
9 namespace RyanSensorApp
10 {
11     public partial class MainForm
12     {
13         private void CreateNoiseAnalysisTab(TabPage tab)
14         {
15             tab.BackColor = Color.FromArgb(240, 240, 245);
16
17             // Create three-panel layout
18             noiseLeftPanel = new Panel
19             {
20                 Dock = DockStyle.Left,
21                 Width = 350,
22                 BackColor = Color.White,
23                 Padding = new Padding(15)
24             };
25
26             noiseRightPanel = new Panel
27             {
28                 Dock = DockStyle.Right,
29                 Width = 400,
30                 BackColor = Color.White,

```

```

31         Padding = new Padding(15)
32     };
33
34     noiseCenterPanel = new Panel
35     {
36         Dock = DockStyle.Fill,
37         BackColor = Color.FromArgb(240, 240, 245),
38         Padding = new Padding(15)
39     };
40
41     CreateNoiseAnalysisLeftPanel();
42     CreateNoiseAnalysisCenterPanel();
43     CreateNoiseAnalysisRightPanel();
44
45     tab.Controls.Add(noiseCenterPanel);
46     tab.Controls.Add(noiseRightPanel);
47     tab.Controls.Add(noiseLeftPanel);
48 }
49
50 private void CreateNoiseAnalysisLeftPanel()
51 {
52     int yPos = 10;
53
54     // Test Setup Group
55     noiseTestSetupGroup = new GroupBox
56     {
57         Text = "TEST CONFIGURATION",
58         Location = new Point(10, yPos),
59         Size = new Size(320, 150),
60         Font = new Font("Segoe UI", 9, FontStyle.Bold)
61     };
62
63     var lblPosition = new Label
64     {
65         Text = "Test Position:",
66         Location = new Point(15, 30),
67         AutoSize = true,
68         Font = new Font("Segoe UI", 9)
69     };
70
71     cmbTestPosition = new ComboBox
72     {
73         Location = new Point(15, 55),
74         Width = 290,
75         DropDownStyle = ComboBoxStyle.DropDownList
76     };
77     cmbTestPosition.Items.AddRange(new object[] {

```

```

78         "Middle Range",
79         "Near Extreme (Close)",
80         "Far Extreme (Far)",
81         "Custom Position"
82     });
83     cmbTestPosition.SelectedIndex = 0;
84
85     var lblDuration = new Label
86     {
87         Text = "Test Duration (seconds):",
88         Location = new Point(15, 90),
89         AutoSize = true,
90         Font = new Font("Segoe UI", 9)
91     };
92
93     numTestDuration = new NumericUpDown
94     {
95         Location = new Point(15, 115),
96         Width = 120,
97         Minimum = 1,
98         Maximum = 60,
99         Value = 10,
100        Font = new Font("Segoe UI", 11)
101    };
102
103    var lblNote = new Label
104    {
105        Text = "Lab standard: 10s",
106        Location = new Point(145, 118),
107        AutoSize = true,
108        Font = new Font("Segoe UI", 8),
109        ForeColor = Color.Gray
110    };
111
112    noiseTestSetupGroup.Controls.AddRange(new Control[] {
113        lblPosition, cmbTestPosition, lblDuration,
114        numTestDuration, lblNote
115    });
116
117    yPos += 160;
118
119    // Test Control Group
120    noiseTestControlGroup = new GroupBox
121    {
122        Text = "TEST CONTROL",
123        Location = new Point(10, yPos),
124        Size = new Size(320, 200),

```

```

124         Font = new Font("Segoe UI", 9, FontStyle.Bold)
125     };
126
127     lblTestStatus = new Label
128     {
129         Text = "    Ready",
130         Location = new Point(15, 30),
131         AutoSize = true,
132         Font = new Font("Segoe UI", 12, FontStyle.Bold),
133         ForeColor = Color.Gray
134     };
135
136     lblTestTimer = new Label
137     {
138         Text = "0.0 s",
139         Location = new Point(15, 60),
140         Size = new Size(290, 30),
141         Font = new Font("Segoe UI", 20, FontStyle.Bold),
142         ForeColor = Color.FromArgb(0, 100, 200),
143         TextAlign = ContentAlignment.MiddleCenter
144     };
145
146     pbTestProgress = new ProgressBar
147     {
148         Location = new Point(15, 100),
149         Size = new Size(290, 25),
150         Minimum = 0,
151         Maximum = 100,
152         Value = 0
153     };
154
155     btnStartTest = new Button
156     {
157         Text = "    START TEST",
158         Location = new Point(15, 135),
159         Size = new Size(290, 50),
160         BackColor = Color.FromArgb(0, 150, 0),
161         ForeColor = Color.White,
162         FlatStyle = FlatStyle.Flat,
163         Font = new Font("Segoe UI", 12, FontStyle.Bold)
164     };
165     btnStartTest.Click += BtnStartNoiseTest_Click;
166
167     btnStopTest = new Button
168     {
169         Text = "    STOP TEST",
170         Location = new Point(15, 135),

```



```

171         Size = new Size(290, 50),
172         BackColor = Color.FromArgb(200, 0, 0),
173         ForeColor = Color.White,
174         FlatStyle = FlatStyle.Flat,
175         Font = new Font("Segoe UI", 12, FontStyle.Bold),
176         Visible = false
177     };
178     btnStopTest.Click += BtnStopNoiseTest_Click;
179
180     noiseTestControlGroup.Controls.AddRange(new Control[] {
181         lblTestStatus, lblTestTimer, pbTestProgress,
182         btnStartTest, btnStopTest
183     });
184
185     yPos += 210;
186
187     // Test Results Group
188     noiseTestResultsGroup = new GroupBox
189     {
190         Text = "CURRENT TEST RESULTS",
191         Location = new Point(10, yPos),
192         Size = new Size(320, 240),
193         Font = new Font("Segoe UI", 9, FontStyle.Bold)
194     };
195
196     var lblLiveSamples = new Label
197     {
198         Text = "Samples:",
199         Location = new Point(15, 30),
200         AutoSize = true,
201         Font = new Font("Segoe UI", 9)
202     };
203
204     lblCurrentSamples = new Label
205     {
206         Text = "0",
207         Location = new Point(100, 30),
208         AutoSize = true,
209         Font = new Font("Segoe UI", 9, FontStyle.Bold)
210     };
211
212     var lblLiveMean = new Label
213     {
214         Text = "Mean Position:",
215         Location = new Point(15, 55),
216         AutoSize = true,
217         Font = new Font("Segoe UI", 9)

```

```

217     };
218
219     lblCurrentMean = new Label
220     {
221         Text = "--- cm",
222         Location = new Point(120, 55),
223         AutoSize = true,
224         Font = new Font("Segoe UI", 9, FontStyle.Bold)
225     };
226
227     var lblLiveStdDev = new Label
228     {
229         Text = "Std Dev (RMS):",
230         Location = new Point(15, 80),
231         AutoSize = true,
232         Font = new Font("Segoe UI", 9)
233     };
234
235     lblCurrentStdDev = new Label
236     {
237         Text = "--- cm",
238         Location = new Point(120, 80),
239         AutoSize = true,
240         Font = new Font("Segoe UI", 9, FontStyle.Bold),
241         ForeColor = Color.FromArgb(200, 0, 0)
242     };
243
244     var separator = new Label
245     {
246         Text = "Final Test Results:",
247         Location = new Point(15, 115),
248         AutoSize = true,
249         Font = new Font("Segoe UI", 9, FontStyle.Bold),
250         ForeColor = Color.FromArgb(0, 100, 150)
251     };
252
253     lblTestMean = new Label
254     {
255         Text = "Mean: ---",
256         Location = new Point(15, 140),
257         AutoSize = true,
258         Font = new Font("Segoe UI", 10)
259     };
260
261     lblTestStdDev = new Label
262     {
263         Text = "RMS Noise: ---",

```

```

264         Location = new Point(15, 165),
265         Size = new Size(290, 20),
266         Font = new Font("Segoe UI", 10, FontStyle.Bold),
267         ForeColor = Color.FromArgb(200, 0, 0)
268     };
269
270     lblTestRange = new Label
271     {
272         Text = "Range: ---",
273         Location = new Point(15, 190),
274         AutoSize = true,
275         Font = new Font("Segoe UI", 9)
276     };
277
278     lblTestSamples = new Label
279     {
280         Text = "Samples: ---",
281         Location = new Point(15, 210),
282         AutoSize = true,
283         Font = new Font("Segoe UI", 9)
284     };
285
286     noiseTestResultsGroup.Controls.AddRange(new Control[] {
287         lblLiveSamples, lblCurrentSamples, lblLiveMean,
288         lblCurrentMean,
289         lblLiveStdDev, lblCurrentStdDev, separator,
290         lblTestMean, lblTestStdDev, lblTestRange,
291         lblTestSamples
292     });
293
294     noiseLeftPanel.Controls.AddRange(new Control[] {
295         noiseTestSetupGroup, noiseTestControlGroup,
296         noiseTestResultsGroup
297     });
298
299     private void CreateNoiseAnalysisCenterPanel()
300     {
301         int yPos = 10;
302
303         // Visualization Group
304         noiseVisualizationGroup = new GroupBox
305         {
306             Text = "REAL-TIME NOISE VISUALIZATION",
307             Location = new Point(10, yPos),
308             Size = new Size(600, 800),
309             Font = new Font("Segoe UI", 9, FontStyle.Bold)

```

```

308     };
309
310     var lblInfo = new Label
311     {
312         Text = "Position readings over test duration with
                 statistical bands",
313         Location = new Point(15, 25),
314         AutoSize = true,
315         Font = new Font("Segoe UI", 9),
316         ForeColor = Color.Gray
317     };
318
319     plotNoiseTest = new FormsPlot
320     {
321         Location = new Point(15, 50),
322         Size = new Size(570, 735),
323         BackColor = Color.White
324     };
325
326     noiseVisualizationGroup.Controls.AddRange(new Control[]
327     {
328         lblInfo, plotNoiseTest
329     });
330
331     noiseCenterPanel.Controls.Add(noiseVisualizationGroup);
332 }
333
334 private void CreateNoiseAnalysisRightPanel()
335 {
336     int yPos = 10;
337
338     // History Group
339     noiseHistoryGroup = new GroupBox
340     {
341         Text = "TEST HISTORY",
342         Location = new Point(10, yPos),
343         Size = new Size(370, 350),
344         Font = new Font("Segoe UI", 9, FontStyle.Bold)
345     };
346
347     dgvNoiseHistory = new DataGridView
348     {
349         Location = new Point(15, 30),
350         Size = new Size(340, 280),
351         AllowUserToAddRows = false,
352         AllowUserToDeleteRows = false,
353         ReadOnly = true,

```

```

353         SelectionMode = DataGridViewSelectionMode.
           FullRowSelect,
354         MultiSelect = false,
355         AutoSizeColumnsMode =
           DataGridViewAutoSizeColumnsMode.Fill,
356         RowHeadersVisible = false,
357         BackgroundColor = Color.White,
358         BorderStyle = BorderStyle.Fixed3D
359     };
360
361     dgvNoiseHistory.Columns.Add("Test", "Test#");
362     dgvNoiseHistory.Columns.Add("Position", "Position");
363     dgvNoiseHistory.Columns.Add("Mean", "Mean(cm)");
364     dgvNoiseHistory.Columns.Add("StdDev", "RMS(cm)");
365     dgvNoiseHistory.Columns.Add("Samples", "N");
366     dgvNoiseHistory.Columns["Test"].FillWeight = 15;
367     dgvNoiseHistory.Columns["Position"].FillWeight = 30;
368     dgvNoiseHistory.Columns["Mean"].FillWeight = 20;
369     dgvNoiseHistory.Columns["StdDev"].FillWeight = 20;
370     dgvNoiseHistory.Columns["Samples"].FillWeight = 15;
371
372     btnClearNoiseTests = new Button
373     {
374         Text = "Clear All Tests",
375         Location = new Point(15, 315),
376         Size = new Size(165, 25),
377         BackColor = Color.FromArgb(150, 150, 150),
378         ForeColor = Color.White,
379         FlatStyle = FlatStyle.Flat,
380         Font = new Font("Segoe UI", 9)
381     };
382     btnClearNoiseTests.Click += BtnClearNoiseTests_Click;
383
384     btnExportNoiseTests = new Button
385     {
386         Text = "Export to CSV",
387         Location = new Point(190, 315),
388         Size = new Size(165, 25),
389         BackColor = Color.FromArgb(50, 100, 150),
390         ForeColor = Color.White,
391         FlatStyle = FlatStyle.Flat,
392         Font = new Font("Segoe UI", 9)
393     };
394     btnExportNoiseTests.Click += BtnExportNoiseTests_Click;
395
396     noiseHistoryGroup.Controls.AddRange(new Control[] {

```

```

397         dgvNoiseHistory, btnClearNoiseTests,
398         btnExportNoiseTests
399     });
400
401     yPos += 360;
402
403     // Comparison Group
404     noiseComparisonGroup = new GroupBox
405     {
406         Text = "COMPARISON & ANALYSIS",
407         Location = new Point(10, yPos),
408         Size = new Size(370, 430),
409         Font = new Font("Segoe UI", 9, FontStyle.Bold)
410     };
411
412     var lblCompInfo = new Label
413     {
414         Text = "RMS Noise Comparison:",
415         Location = new Point(15, 25),
416         AutoSize = true,
417         Font = new Font("Segoe UI", 9, FontStyle.Bold)
418     };
419
420     plotNoiseComparison = new FormsPlot
421     {
422         Location = new Point(15, 50),
423         Size = new Size(340, 200),
424         BackColor = Color.White
425     };
426
427     var lblSummary = new Label
428     {
429         Text = "Statistical Summary:",
430         Location = new Point(15, 260),
431         AutoSize = true,
432         Font = new Font("Segoe UI", 9, FontStyle.Bold)
433     };
434
435     txtComparison = new TextBox
436     {
437         Location = new Point(15, 285),
438         Size = new Size(340, 135),
439         Multiline = true,
440         ScrollBars = ScrollBars.Vertical,
441         ReadOnly = true,
442         Font = new Font("Consolas", 8),
443         BackColor = Color.FromArgb(250, 250, 250)

```

```

443     };
444
445     noiseComparisonGroup.Controls.AddRange(new Control[] {
446         lblCompInfo, plotNoiseComparison, lblSummary,
447         txtComparison
448     });
449
450     noiseRightPanel.Controls.AddRange(new Control[] {
451         noiseHistoryGroup, noiseComparisonGroup
452     });
453 }
454
455 private void InitializeNoiseAnalysisService()
456 {
457     _noiseAnalysisService = new NoiseAnalysisService();
458
459     // Initialize noise test timer
460     _noiseTestTimer = new System.Windows.Forms.Timer();
461     _noiseTestTimer.Interval = 100; // Update every 100ms
462     _noiseTestTimer.Tick += NoiseTestTimer_Tick;
463
464     // Initialize noise plot
465     plotNoiseTest.Plot.Title("");
466     plotNoiseTest.Plot.XLabel("Time (s)");
467     plotNoiseTest.Plot.YLabel("Distance (cm)");
468
469     // Initialize comparison plot
470     plotNoiseComparison.Plot.Title("");
471     plotNoiseComparison.Plot.XLabel("Test Position");
472     plotNoiseComparison.Plot.YLabel("RMS Noise (cm)");
473
474     UpdateNoiseComparisonPlot();
475 }
476
477 private void BtnStartNoiseTest_Click(object? sender,
478     EventArgs e)
479 {
480     if (!_serialPort.IsConnected)
481     {
482         MessageBox.Show("Please connect to the sensor first!",
483             "Not Connected",
484             MessageBoxButtons.OK, MessageBoxIcon.Warning);
485         return;
486     }
487
488     if (_currentCalibration == null)
489     {

```

```

487         MessageBox.Show("Please load a calibration first!",
488             "No Calibration",
489             MessageBoxButtons.OK, MessageBoxIcon.Warning);
490         return;
491     }
492     // Get test position
493     TestPosition position = cmbTestPosition.SelectedIndex
494         switch
495     {
496         0 => TestPosition.MiddleRange,
497         1 => TestPosition.NearExtreme,
498         2 => TestPosition.FarExtreme,
499         3 => TestPosition.Custom,
500         _ => TestPosition.MiddleRange
501     };
502     // Create new test
503     _currentNoiseTest = _noiseAnalysisService.CreateNewTest(
504         position);
505     _currentNoiseTest.DurationSeconds = (double)
506         numTestDuration.Value;
507     _noiseTestStartTime = DateTime.Now;
508     _isNoiseTestRunning = true;
509     // Update UI
510     btnStartTest.Visible = false;
511     btnStopTest.Visible = true;
512     cmbTestPosition.Enabled = false;
513     numTestDuration.Enabled = false;
514     lblTestStatus.Text = "    RECORDING...";
515     lblTestStatus.ForeColor = Color.Red;
516     lblTestTimer.Text = "0.0 s";
517     pbTestProgress.Value = 0;
518     // Clear current results
519     lblCurrentSamples.Text = "0";
520     lblCurrentMean.Text = "--- cm";
521     lblCurrentStdDev.Text = "--- cm";
522     // Start timer
523     _noiseTestTimer.Start();
524 }
525
526 private void BtnStopNoiseTest_Click(object? sender,
527     EventArgs e)
528 {

```



```

529         StopNoiseTest();
530     }
531
532     private void StopNoiseTest()
533     {
534         if (!_isNoiseTestRunning || _currentNoiseTest == null)
535             return;
536
537         _isNoiseTestRunning = false;
538         _noiseTestTimer.Stop();
539
540         // Calculate final statistics
541         _currentNoiseTest.CalculateStatistics();
542
543         // Save test
544         _noiseAnalysisService.SaveTest(_currentNoiseTest);
545
546         // Update UI
547         lblTestStatus.Text = "    Test Complete";
548         lblTestStatus.ForeColor = Color.Green;
549         btnStartTest.Visible = true;
550         btnStopTest.Visible = false;
551         cmbTestPosition.Enabled = true;
552         numTestDuration.Enabled = true;
553
554         // Display final results
555         lblTestMean.Text = $"Mean: {_currentNoiseTest.
                    MeanDistance:F4} cm";
556         lblTestStdDev.Text = $"RMS Noise: {_currentNoiseTest.
                    StandardDeviation:F4} cm";
557         lblTestRange.Text = $"Range: {_currentNoiseTest.
                    MinDistance:F4} - {_currentNoiseTest.MaxDistance:F4}
                    cm";
558         lblTestSamples.Text = $"Samples: {_currentNoiseTest.
                    SampleCount}";
559
560         // Update test history
561         UpdateNoiseHistoryTable();
562         UpdateNoiseComparisonPlot();
563         UpdateComparisonSummary();
564
565         MessageBox.Show(_currentNoiseTest.GetSummary(), "Test
                    Complete",
566             MessageBoxButtons.OK, MessageBoxIcon.Information);
567
568         _currentNoiseTest = null;
569     }

```

```

570
571 private void NoiseTestTimer_Tick(object? sender, EventArgs e
572 )
573 {
574     if (!_isNoiseTestRunning || _currentNoiseTest == null)
575         return;
576
577     // Calculate elapsed time
578     double elapsedSeconds = (DateTime.Now -
579         _noiseTestStartTime).TotalSeconds;
580     lblTestTimer.Text = $"{elapsedSeconds:F1} s";
581
582     // Update progress bar
583     int progress = (int)((elapsedSeconds / _currentNoiseTest
584         .DurationSeconds) * 100);
585     pbTestProgress.Value = Math.Min(progress, 100);
586
587     // Add current reading to test
588     if (_currentCalibration != null)
589     {
590         double distance = _currentCalibration.
591             ConvertAdcToDistance(_currentAdcValue);
592         _currentNoiseTest.DistanceReadings.Add(distance);
593         _currentNoiseTest.AdcReadings.Add(_currentAdcValue);
594
595         // Calculate running statistics
596         if (_currentNoiseTest.DistanceReadings.Count > 1)
597         {
598             double mean = _currentNoiseTest.DistanceReadings
599                 .Average();
600             double sumSqDiff = _currentNoiseTest.
601                 DistanceReadings.Sum(x => Math.Pow(x - mean,
602                 2));
603             double stdDev = Math.Sqrt(sumSqDiff /
604                 _currentNoiseTest.DistanceReadings.Count);
605
606             lblCurrentSamples.Text = _currentNoiseTest.
607                 DistanceReadings.Count.ToString();
608             lblCurrentMean.Text = $"{mean:F4} cm";
609             lblCurrentStdDev.Text = $"{stdDev:F4} cm";
610         }
611
612         // Update plot
613         UpdateNoiseTestPlot();
614     }
615
616     // Check if test duration reached
617

```

```

608         if (elapsedSeconds >= _currentNoiseTest.DurationSeconds)
609         {
610             StopNoiseTest();
611         }
612     }
613
614     private void UpdateNoiseTestPlot()
615     {
616         if (_currentNoiseTest == null || _currentNoiseTest.
            DistanceReadings.Count == 0)
617             return;
618
619         plotNoiseTest.Plot.Clear();
620
621         // Create time array
622         double[] times = Enumerable.Range(0, _currentNoiseTest.
            DistanceReadings.Count)
623             .Select(i => i * 0.1) // 100ms intervals
624             .ToArray();
625         double[] distances = _currentNoiseTest.DistanceReadings.
            ToArray();
626
627         // Plot data
628         var scatter = plotNoiseTest.Plot.Add.Scatter(times,
            distances);
629         scatter.Color = ScottPlot.Color.FromHex("#0064C8");
630         scatter.LineWidth = 2;
631         scatter.MarkerSize = 3;
632         scatter.LegendText = "Position Readings";
633
634         // Add mean line if we have enough data
635         if (_currentNoiseTest.DistanceReadings.Count > 2)
636         {
637             double mean = distances.Average();
638             double stdDev = Math.Sqrt(distances.Sum(x => Math.
                Pow(x - mean, 2)) / distances.Length);
639
640             var meanLine = plotNoiseTest.Plot.Add.HorizontalLine
                (mean);
641             meanLine.Color = ScottPlot.Color.FromHex("#00AA00");
642             meanLine.LineWidth = 2;
643             meanLine.LinePattern = ScottPlot.LinePattern.Dashed;
644             meanLine.LegendText = "Mean";
645
646             // Add 1 band
647             var plusSigma = plotNoiseTest.Plot.Add.
                HorizontalLine(mean + stdDev);

```

```

648         plusSigma.Color = ScottPlot.Color.FromHex("#FF6600")
649         ;
650         plusSigma.LineWidth = 1;
651         plusSigma.LinePattern = ScottPlot.LinePattern.Dotted
652         ;
653         plusSigma.LegendText = " 1 ";
654
655         var minusSigma = plotNoiseTest.Plot.Add.
656         HorizontalLine(mean - stdDev);
657         minusSigma.Color = ScottPlot.Color.FromHex("#FF6600")
658         );
659         minusSigma.LineWidth = 1;
660         minusSigma.LinePattern = ScottPlot.LinePattern.
661         Dotted;
662     }
663
664     plotNoiseTest.Plot.Legend.IsVisible = true;
665     plotNoiseTest.Plot.Axes.AutoScale();
666     plotNoiseTest.Refresh();
667 }
668
669 private void UpdateNoiseHistoryTable()
670 {
671     dgvNoiseHistory.Rows.Clear();
672
673     var tests = _noiseAnalysisService.GetAllTests();
674     foreach (var test in tests)
675     {
676         dgvNoiseHistory.Rows.Add(
677             test.TestNumber.ToString(),
678             test.GetPositionString(),
679             test.MeanDistance.ToString("F4"),
680             test.StandardDeviation.ToString("F4"),
681             test.SampleCount.ToString()
682         );
683     }
684 }
685
686 private void UpdateNoiseComparisonPlot()
687 {
688     plotNoiseComparison.Plot.Clear();
689
690     var avgRms = _noiseAnalysisService.
691     GetAverageRMSByPosition();
692     if (avgRms.Count == 0)
693     {
694         plotNoiseComparison.Refresh();
695     }
696 }

```

```

689         return;
690     }
691
692     // Create bar chart data
693     var positions = avgRms.Keys.ToList();
694     var rmsValues = avgRms.Values.ToList();
695
696     double[] posIndices = Enumerable.Range(0, positions.
        Count).Select(i => (double)i).ToArray();
697     string[] posLabels = positions.Select(p => p switch
698     {
699         TestPosition.MiddleRange => "Middle",
700         TestPosition.NearExtreme => "Near",
701         TestPosition.FarExtreme => "Far",
702         _ => "Custom"
703     }).ToArray();
704
705     var bar = plotNoiseComparison.Plot.Add.Bars(posIndices,
        rmsValues.ToArray());
706     bar.Color = ScottPlot.Color.FromHex("#FF6600");
707
708     plotNoiseComparison.Plot.Axes.Bottom.TickGenerator = new
        ScottPlot.TickGenerators.NumericManual(
709         posIndices, posLabels);
710     plotNoiseComparison.Plot.Axes.AutoScale();
711     plotNoiseComparison.Refresh();
712 }
713
714 private void UpdateComparisonSummary()
715 {
716     txtComparison.Text = _noiseAnalysisService.
        GetComparisonSummary();
717 }
718
719 private void BtnClearNoiseTests_Click(object? sender,
        EventArgs e)
720 {
721     var result = MessageBox.Show(
722         "Clear all noise test data?",
723         "Confirm Clear",
724         MessageBoxButtons.YesNo,
725         MessageBoxIcon.Question);
726
727     if (result == DialogResult.Yes)
728     {
729         _noiseAnalysisService.ClearAllTests();
730         UpdateNoiseHistoryTable();

```

```

731         UpdateNoiseComparisonPlot();
732         UpdateComparisonSummary();
733
734         lblTestMean.Text = "Mean: ---";
735         lblTestStdDev.Text = "RMS Noise: ---";
736         lblTestRange.Text = "Range: ---";
737         lblTestSamples.Text = "Samples: ---";
738     }
739 }
740
741 private void BtnExportNoiseTests_Click(object? sender,
742     EventArgs e)
743 {
744     if (_noiseAnalysisService.GetTestCount() == 0)
745     {
746         MessageBox.Show("No test data to export!", "No Data"
747             ,
748             MessageBoxButtons.OK, MessageBoxIcon.Information
749         );
750         return;
751     }
752
753     using (SaveFileDialog sfd = new SaveFileDialog())
754     {
755         sfd.Filter = "CSV Files (*.csv)|*.csv";
756         sfd.FileName = $"noise_analysis_{DateTime.Now:
757             yyyyMMdd_HHmmss}.csv";
758         if (sfd.ShowDialog() == DialogResult.OK)
759         {
760             try
761             {
762                 _noiseAnalysisService.ExportToCSV(sfd.
763                     FileName);
764                 MessageBox.Show($"Noise analysis exported
765                     successfully!\n{sfd.FileName}",
766                     "Success", MessageBoxButtons.OK,
767                     MessageBoxIcon.Information);
768             }
769             catch (Exception ex)
770             {
771                 MessageBox.Show($"Export failed: {ex.Message
772                     }", "Error",
773                     MessageBoxButtons.OK, MessageBoxIcon.
774                     Error);
775             }
776         }
777     }
778 }

```

```

769     }
770 }
771 }

```

A.4 Serial Port Handling

Listing 4: SerialPortService Data Pipeline

```

1  using System;
2  using System.IO.Ports;
3  using System.Threading;
4  using System.Threading.Tasks;
5
6  namespace RyanSensorApp.Services
7  {
8      public class SerialPortService : IDisposable
9      {
10         private SerialPort? _serialPort;
11         private CancellationTokenseSource? _cancellationTokenseSource;
12         private Task? _readTask;
13         private byte[] _buffer = new byte[3];
14         private int _bufferIndex = 0;
15         private const byte START_BYTE = 0xFF;
16
17         public event EventHandler<AdcDataReceivedEventArgs>?
            AdcDataReceived;
18         public event EventHandler<string>? ErrorOccurred;
19         public event EventHandler? ConnectionLost;
20
21         public bool IsConnected => _serialPort?.IsOpen ?? false;
22
23         public bool Connect(string portName, int baudRate = 9600)
24         {
25             try
26             {
27                 Disconnect();
28
29                 _serialPort = new SerialPort(portName, baudRate,
                    Parity.None, 8, StopBits.One)
30                 {
31                     ReadTimeout = 1000,
32                     WriteTimeout = 1000
33                 };
34
35                 _serialPort.Open();
36
37                 // Start reading in background thread

```

```

38         _cancellationTokenSource = new
39             CancellationTokenSource();
40         _readTask = Task.Run(() => ReadDataAsync(
41             _cancellationTokenSource.Token));
42
43         return true;
44     }
45     catch (Exception ex)
46     {
47         ErrorOccurred?.Invoke(this, $"Connection error: {ex.
48             Message}");
49         return false;
50     }
51 }
52
53 public void Disconnect()
54 {
55     try
56     {
57         _cancellationTokenSource?.Cancel();
58         _readTask?.Wait(TimeSpan.FromSeconds(2));
59
60         if (_serialPort?.IsOpen == true)
61         {
62             _serialPort.Close();
63         }
64
65         _serialPort?.Dispose();
66         _serialPort = null;
67         _cancellationTokenSource?.Dispose();
68         _cancellationTokenSource = null;
69         _bufferIndex = 0;
70     }
71     catch (Exception ex)
72     {
73         ErrorOccurred?.Invoke(this, $"Disconnect error: {ex.
74             Message}");
75     }
76 }
77
78 private async Task ReadDataAsync(CancellationToken
79     cancellationToken)
80 {
81     while (!cancellationToken.IsCancellationRequested)
82     {
83         try
84         {

```



```

80         if (_serialPort?.IsOpen == true && _serialPort.
            BytesToRead > 0)
81         {
82             int readByte = _serialPort.ReadByte();
83             if (readByte >= 0)
84             {
85                 ProcessByte((byte)readByte);
86             }
87         }
88         else
89         {
90             await Task.Delay(1, cancellationToken);
91         }
92     }
93     catch (TimeoutException)
94     {
95         // Normal timeout, continue
96     }
97     catch (InvalidOperationException)
98     {
99         // Port closed
100         ConnectionLost?.Invoke(this, EventArgs.Empty);
101         break;
102     }
103     catch (Exception ex)
104     {
105         if (!cancellationToken.IsCancellationRequested)
106         {
107             ErrorOccurred?.Invoke(this, $"Read error: {
                ex.Message}");
108             await Task.Delay(100, cancellationToken);
109         }
110     }
111 }
112
113
114 private void ProcessByte(byte data)
115 {
116     // State machine for packet parsing
117     if (_bufferIndex == 0)
118     {
119         // Looking for start byte
120         if (data == START_BYTE)
121         {
122             _buffer[0] = data;
123             _bufferIndex = 1;
124         }

```

```

125     }
126     else if (_bufferIndex == 1)
127     {
128         // MS5B (most significant 5 bits)
129         _buffer[1] = data;
130         _bufferIndex = 2;
131     }
132     else if (_bufferIndex == 2)
133     {
134         // LS5B (least significant 5 bits)
135         _buffer[2] = data;
136
137         // Reassemble 10-bit ADC value
138         int ms5b = _buffer[1] & 0x1F; // Mask to 5 bits
139         int ls5b = _buffer[2] & 0x1F; // Mask to 5 bits
140         int adcValue = (ms5b << 5) | ls5b; // Combine into
            10-bit value
141
142         // Raise event with ADC data
143         AdcDataReceived?.Invoke(this, new
            AdcDataReceivedEventArgs(adcValue));
144
145         // Reset for next packet
146         _bufferIndex = 0;
147     }
148 }
149
150 public static string[] GetAvailablePorts()
151 {
152     return SerialPort.GetPortNames();
153 }
154
155 public void Dispose()
156 {
157     Disconnect();
158 }
159 }
160
161 public class AdcDataReceivedEventArgs : EventArgs
162 {
163     public int AdcValue { get; }
164     public DateTime Timestamp { get; }
165
166     public AdcDataReceivedEventArgs(int adcValue)
167     {
168         AdcValue = adcValue;
169         Timestamp = DateTime.Now;

```

```

170     }
171 }
172 }

```

A.5 Data Logging

Listing 5: CSV DataLogger Implementation

```

1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using System.Linq;
5 using RyanSensorApp.Models;
6
7 namespace RyanSensorApp.Services
8 {
9     public class DataLogger
10    {
11        private List<SensorReading> _readings;
12        private readonly object _lock = new object();
13
14        public int Count => _readings.Count;
15
16        public DataLogger()
17        {
18            _readings = new List<SensorReading>();
19        }
20
21        public void AddReading(SensorReading reading)
22        {
23            lock (_lock)
24            {
25                _readings.Add(reading);
26            }
27        }
28
29        public List<SensorReading> GetAllReadings()
30        {
31            lock (_lock)
32            {
33                return new List<SensorReading>(_readings);
34            }
35        }
36
37        public List<SensorReading> GetReadings(int count)
38        {
39            lock (_lock)
40            {

```

```

41         return _readings.Skip(Math.Max(0, _readings.Count -
42             count)).ToList();
43     }
44 }
45 public void Clear()
46 {
47     lock (_lock)
48     {
49         _readings.Clear();
50     }
51 }
52
53 public void ExportToCsv(string filePath)
54 {
55     lock (_lock)
56     {
57         using (StreamWriter writer = new StreamWriter(
58             filePath))
59         {
60             writer.WriteLine(SensorReading.GetCsvHeader());
61             foreach (var reading in _readings)
62             {
63                 writer.WriteLine(reading.ToString());
64             }
65         }
66     }
67
68     public (double min, double max, double avg, double stdDev)
69     GetStatistics(bool forAdc = true)
70     {
71         lock (_lock)
72         {
73             if (_readings.Count == 0)
74                 return (0, 0, 0, 0);
75
76             double[] values = forAdc
77                 ? _readings.Select(r => (double)r.AdcValue).
78                   ToArray()
79                 : _readings.Select(r => r.Distance).ToArray();
80
81             double min = values.Min();
82             double max = values.Max();
83             double avg = values.Average();
84
85             // Calculate standard deviation

```

```

84         double sumSquares = values.Sum(v => Math.Pow(v - avg
85             , 2));
86         double stdDev = Math.Sqrt(sumSquares / values.Length
87             );
88         return (min, max, avg, stdDev);
89     }
90 }
91 public double GetRmsNoise(DateTime startTime, TimeSpan
92     duration)
93 {
94     lock (_lock)
95     {
96         var filteredReadings = _readings
97             .Where(r => r.Timestamp >= startTime && r.
98                 Timestamp <= startTime.Add(duration))
99             .Select(r => r.Distance)
100             .ToArray();
101
102         if (filteredReadings.Length == 0)
103             return 0;
104
105         double avg = filteredReadings.Average();
106         double sumSquares = filteredReadings.Sum(v => Math.
107             Pow(v - avg, 2));
108         return Math.Sqrt(sumSquares / filteredReadings.
109             Length);
110     }
111 }
112 }
113 }
114 }

```

A.6 Calibration Utilities

Listing 6: CalibrationService for Curve Fitting

```

1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using System.Linq;
5 using RyanSensorApp.Models;
6 using MathNet.Numerics;
7 using MathNet.Numerics.LinearRegression;
8 using Newtonsoft.Json;
9
10 namespace RyanSensorApp.Services
11 {

```

```

12 public class CalibrationService
13 {
14     public CalibrationData PerformCalibration(List<
15         CalibrationPoint> points, FitType fitType)
16     {
17         if (points == null || points.Count < 2)
18         {
19             throw new ArgumentException("At least 2 calibration
20                 points are required.");
21         }
22
23         var calibration = new CalibrationData
24         {
25             Points = new List<CalibrationPoint>(points),
26             FitType = fitType
27         };
28
29         // Extract x (ADC) and y (Distance) values
30         double[] xData = points.Select(p => (double)p.AdcValue).
31             ToArray();
32         double[] yData = points.Select(p => p.Distance).ToArray
33             ();
34
35         try
36         {
37             switch (fitType)
38             {
39                 case FitType.Linear:
40                     calibration.Coefficients = FitLinear(xData,
41                         yData, out double rSquaredLinear);
42                     calibration.RSquared = rSquaredLinear;
43                     calibration.Equation = $"y = {calibration.
44                         Coefficients[0]:F6}x + {calibration.
45                         Coefficients[1]:F6}";
46                     break;
47
48                 case FitType.Polynomial2:
49                     calibration.Coefficients = FitPolynomial(
50                         xData, yData, 2, out double rSquared2);
51                     calibration.RSquared = rSquared2;
52                     calibration.Equation = $"y = {calibration.
53                         Coefficients[0]:E3}x + {calibration.
54                         Coefficients[1]:F6}x + {calibration.
55                         Coefficients[2]:F6}";
56                     break;
57
58                 case FitType.Polynomial3:

```

```

48         calibration.Coefficients = FitPolynomial(
49             xData, yData, 3, out double rSquared3);
50         calibration.RSquared = rSquared3;
51         calibration.Equation = $"y = {calibration.
52             Coefficients[0]:E3}x + {calibration.
53             Coefficients[1]:E3}x + {calibration.
54             Coefficients[2]:F6}x + {calibration.
55             Coefficients[3]:F6}";
56         break;
57
58     case FitType.Power:
59         calibration.Coefficients = FitPower(xData,
60             yData, out double rSquaredPower);
61         calibration.RSquared = rSquaredPower;
62         calibration.Equation = $"y = {calibration.
63             Coefficients[0]:F6} x^{calibration.
64             Coefficients[1]:F6}";
65         break;
66
67     case FitType.Inverse:
68         calibration.Coefficients = FitInverse(xData,
69             yData, out double rSquaredInv);
70         calibration.RSquared = rSquaredInv;
71         calibration.Equation = $"y = {calibration.
72             Coefficients[0]:F6} / (x - {calibration.
73             Coefficients[1]:F6}) + {calibration.
74             Coefficients[2]:F6}";
75         break;
76     }
77
78     // Note: MinAdcThreshold and MaxAdcThreshold are set
79     // by user configuration,
80     // not automatically from calibration points
81 }
82
83 catch (Exception ex)
84 {
85     throw new InvalidOperationException($"Curve fitting
86         failed: {ex.Message}", ex);
87 }
88
89 return calibration;
90 }
91
92 private double[] FitLinear(double[] x, double[] y, out
93     double rSquared)
94 {
95     var (slope, intercept) = SimpleRegression.Fit(x, y);

```

```

80         rSquared = CalculateRSquared(x, y, new[] { slope,
81             intercept }, FitType.Linear);
82     return new[] { slope, intercept };
83 }
84
85 private double[] FitPolynomial(double[] x, double[] y, int
86     order, out double rSquared)
87 {
88     double[] coefficients = Fit.Polynomial(x, y, order);
89     // MathNet returns coefficients in ascending order (c0 +
90     // c1*x + c2*x^2 + ...)
91     // We want descending order for our formula (a*x^n + b*x
92     // ^{(n-1)} + ...)
93     Array.Reverse(coefficients);
94     rSquared = CalculateRSquared(x, y, coefficients, order
95         == 2 ? FitType.Polynomial2 : FitType.Polynomial3);
96     return coefficients;
97 }
98
99 private double[] FitPower(double[] x, double[] y, out double
100     rSquared)
101 {
102     // Power fit: y = a * x^b
103     // Transform to linear: ln(y) = ln(a) + b*ln(x)
104     if (x.Any(val => val <= 0) || y.Any(val => val <= 0))
105     {
106         throw new ArgumentException("Power fit requires all
107             positive values.");
108     }
109
110     double[] lnX = x.Select(val => Math.Log(val)).ToArray();
111     double[] lnY = y.Select(val => Math.Log(val)).ToArray();
112
113     var (b, lnA) = SimpleRegression.Fit(lnX, lnY);
114     double a = Math.Exp(lnA);
115
116     rSquared = CalculateRSquared(x, y, new[] { a, b },
117         FitType.Power);
118     return new[] { a, b };
119 }
120
121 private double[] FitInverse(double[] x, double[] y, out
122     double rSquared)
123 {
124     // Inverse fit: y = a / (x - b) + c
125     // We'll use a simplified approach with b=0: y = a/x + c
126     // Transform to linear: y = a*(1/x) + c

```



```

118
119     double[] invX = x.Select(val => 1.0 / val).ToArray();
120     var (a, c) = SimpleRegression.Fit(invX, y);
121     double b = 0;
122
123     rSquared = CalculateRSquared(x, y, new[] { a, b, c },
124         FitType.Inverse);
125     return new[] { a, b, c };
126 }
127
128 private double CalculateRSquared(double[] x, double[] y,
129     double[] coefficients, FitType fitType)
130 {
131     double meanY = y.Average();
132     double ssTotal = y.Sum(yi => Math.Pow(yi - meanY, 2));
133     double ssResidual = 0;
134
135     for (int i = 0; i < x.Length; i++)
136     {
137         double predicted = 0;
138         switch (fitType)
139         {
140             case FitType.Linear:
141                 predicted = coefficients[0] * x[i] +
142                     coefficients[1];
143                 break;
144             case FitType.Polynomial2:
145                 predicted = coefficients[0] * x[i] * x[i] +
146                     coefficients[1] * x[i] + coefficients[2];
147                 break;
148             case FitType.Polynomial3:
149                 predicted = coefficients[0] * Math.Pow(x[i],
150                     3) + coefficients[1] * Math.Pow(x[i], 2)
151                     +
152                     coefficients[2] * x[i] +
153                     coefficients[3];
154                 break;
155             case FitType.Power:
156                 predicted = coefficients[0] * Math.Pow(x[i],
157                     coefficients[1]);
158                 break;
159             case FitType.Inverse:
160                 predicted = coefficients[0] / (x[i] -
161                     coefficients[1]) + coefficients[2];
162                 break;
163         }
164         ssResidual += Math.Pow(y[i] - predicted, 2);
165     }
166 }

```

```

156         }
157
158         return 1 - (ssResidual / ssTotal);
159     }
160
161     public void SaveCalibration(CalibrationData calibration,
162                                string filePath)
163     {
164         try
165         {
166             string json = JsonConvert.SerializeObject(
167                 calibration, Formatting.Indented);
168             File.WriteAllText(filePath, json);
169         }
170         catch (Exception ex)
171         {
172             throw new IOException($"Failed to save calibration:
173                                     {ex.Message}", ex);
174         }
175     }
176
177     public CalibrationData LoadCalibration(string filePath)
178     {
179         try
180         {
181             string json = File.ReadAllText(filePath);
182             var calibration = JsonConvert.DeserializeObject<
183                 CalibrationData>(json);
184             if (calibration == null)
185             {
186                 throw new InvalidDataException("Failed to
187                                         deserialize calibration data.");
188             }
189             return calibration;
190         }
191         catch (Exception ex)
192         {
193             throw new IOException($"Failed to load calibration:
194                                     {ex.Message}", ex);
195         }
196     }
197 }

```

A.7 Noise Analysis Helpers

Listing 7: NoiseAnalysisService Statistics Module

```

1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using System.Linq;
5 using System.Text;
6 using RyanSensorApp.Models;
7
8 namespace RyanSensorApp.Services
9 {
10     public class NoiseAnalysisService
11     {
12         private List<NoiseTestResult> _testResults;
13         private int _nextTestNumber;
14
15         public NoiseAnalysisService()
16         {
17             _testResults = new List<NoiseTestResult>();
18             _nextTestNumber = 1;
19         }
20
21         public List<NoiseTestResult> GetAllTests()
22         {
23             return new List<NoiseTestResult>(_testResults);
24         }
25
26         public int GetTestCount()
27         {
28             return _testResults.Count;
29         }
30
31         public NoiseTestResult CreateNewTest(TestPosition position,
32             string customDescription = "")
33         {
34             var test = new NoiseTestResult
35             {
36                 TestNumber = _nextTestNumber++,
37                 Position = position,
38                 PositionDescription = customDescription,
39                 TestDateTime = DateTime.Now
40             };
41             return test;
42         }
43
44         public void SaveTest(NoiseTestResult test)
45         {
46             test.CalculateStatistics();
47         }
48     }
49 }

```

```

46         _testResults.Add(test);
47     }
48
49     public void DeleteTest(int testNumber)
50     {
51         _testResults.RemoveAll(t => t.TestNumber == testNumber);
52     }
53
54     public void ClearAllTests()
55     {
56         _testResults.Clear();
57         _nextTestNumber = 1;
58     }
59
60     public string GetComparisonSummary()
61     {
62         if (_testResults.Count == 0)
63             return "No tests available for comparison.";
64
65         var sb = new StringBuilder();
66         sb.AppendLine("=== NOISE ANALYSIS COMPARISON ===\n");
67
68         // Group by position
69         var middleTests = _testResults.Where(t => t.Position ==
70             TestPosition.MiddleRange).ToList();
71         var nearTests = _testResults.Where(t => t.Position ==
72             TestPosition.NearExtreme).ToList();
73         var farTests = _testResults.Where(t => t.Position ==
74             TestPosition.FarExtreme).ToList();
75
76         if (middleTests.Any())
77         {
78             sb.AppendLine("MIDDLE RANGE:");
79             foreach (var test in middleTests)
80             {
81                 sb.AppendLine($"    Test #{test.TestNumber}: Mean
82                     ={test.MeanDistance:F4} cm, RMS Noise={test.
83                     StandardDeviation:F4} cm");
84             }
85             sb.AppendLine($"    Average RMS Noise: {middleTests.
86                 Average(t => t.StandardDeviation):F4} cm\n");
87         }
88
89         if (nearTests.Any())
90         {
91             sb.AppendLine("NEAR EXTREME (Close):");
92             foreach (var test in nearTests)

```

```

87     {
88         sb.AppendLine($"    Test #{test.TestNumber}: Mean
           ={test.MeanDistance:F4} cm, RMS Noise={test.
           StandardDeviation:F4} cm");
89     }
90     sb.AppendLine($"    Average RMS Noise: {nearTests.
           Average(t => t.StandardDeviation):F4} cm\n");
91 }
92
93 if (farTests.Any())
94 {
95     sb.AppendLine("FAR EXTREME:");
96     foreach (var test in farTests)
97     {
98         sb.AppendLine($"    Test #{test.TestNumber}: Mean
           ={test.MeanDistance:F4} cm, RMS Noise={test.
           StandardDeviation:F4} cm");
99     }
100    sb.AppendLine($"    Average RMS Noise: {farTests.
           Average(t => t.StandardDeviation):F4} cm\n");
101 }
102
103 // Comparison analysis
104 if (middleTests.Any() && (nearTests.Any() || farTests.
    Any()))
105 {
106     double middleRms = middleTests.Average(t => t.
        StandardDeviation);
107     sb.AppendLine("=== COMPARISON ===");
108
109     if (nearTests.Any())
110     {
111         double nearRms = nearTests.Average(t => t.
            StandardDeviation);
112         double nearDiff = nearRms - middleRms;
113         double nearRatio = middleRms > 0 ? nearRms /
            middleRms : 0;
114         sb.AppendLine($"Near Extreme vs Middle: {
            nearDiff:+0.0000;-0.0000} cm ({nearRatio:F2}x
            )");
115     }
116
117     if (farTests.Any())
118     {
119         double farRms = farTests.Average(t => t.
            StandardDeviation);
120         double farDiff = farRms - middleRms;

```

```

121         double farRatio = middleRms > 0 ? farRms /
122             middleRms : 0;
123         sb.AppendLine($"Far Extreme vs Middle: {farDiff
124             :+0.0000;-0.0000} cm ({farRatio:F2}x)");
125     }
126 }
127
128     return sb.ToString();
129 }
130
131 public void ExportToCSV(string filePath)
132 {
133     using (StreamWriter writer = new StreamWriter(filePath))
134     {
135         // Write header
136         writer.WriteLine("Test Number,Position,Date/Time,
137             Duration (s),Samples,Mean Distance (cm),Std Dev (
138             RMS) (cm),Min Distance (cm),Max Distance (cm),
139             Mean ADC");
140
141         // Write data
142         foreach (var test in _testResults)
143         {
144             writer.WriteLine($"{test.TestNumber}," +
145                 $"{test.GetPositionString()}," +
146                 $"{test.TestDateTime:yyyy-MM-dd
147                     HH:mm:ss}," +
148                 $"{test.DurationSeconds:F2}," +
149                 $"{test.SampleCount}," +
150                 $"{test.MeanDistance:F6}," +
151                 $"{test.StandardDeviation:F6}," +
152                 $"{test.MinDistance:F6}," +
153                 $"{test.MaxDistance:F6}," +
154                 $"{test.MeanAdc:F2}");
155         }
156     }
157 }
158
159 public void ExportDetailedData(string filePath,
160     NoiseTestResult test)
161 {
162     using (StreamWriter writer = new StreamWriter(filePath))
163     {
164         // Write header info
165         writer.WriteLine($"Test #{test.TestNumber} - {test.
166             GetPositionString()}");

```

```

159         writer.WriteLine($"Date/Time: {test.TestDateTime:
160             yyyy-MM-dd HH:mm:ss}");
161         writer.WriteLine($"Duration: {test.DurationSeconds:
162             F2} seconds");
163         writer.WriteLine($"Samples: {test.SampleCount}");
164         writer.WriteLine($"Mean Distance: {test.MeanDistance:
165             F6} cm");
166         writer.WriteLine($"Standard Deviation (RMS Noise): {
167             test.StandardDeviation:F6} cm");
168         writer.WriteLine($"Min Distance: {test.MinDistance:
169             F6} cm");
170         writer.WriteLine($"Max Distance: {test.MaxDistance:
171             F6} cm");
172         writer.WriteLine();
173         writer.WriteLine("Sample #,ADC Value,Distance (cm)")
174         ;
175
176         // Write all samples
177         for (int i = 0; i < test.DistanceReadings.Count; i
178             ++)
179         {
180             writer.WriteLine($"{{i + 1}},{{test.AdcReadings[i
181                 ]}},{{test.DistanceReadings[i]:F6}}");
182         }
183     }
184
185     public Dictionary<TestPosition, double>
186     GetAveragerRMSByPosition()
187     {
188         var result = new Dictionary<TestPosition, double>();
189
190         var positions = new[] { TestPosition.MiddleRange,
191             TestPosition.NearExtreme, TestPosition.FarExtreme };
192
193         foreach (var pos in positions)
194         {
195             var tests = _testResults.Where(t => t.Position ==
196                 pos).ToList();
197             if (tests.Any())
198             {
199                 result[pos] = tests.Average(t => t.
200                     StandardDeviation);
201             }
202         }
203
204         return result;

```

193
194
195

```
}  
}  
}
```