MECH 421/423 Lab 4
Op-Amp Circuits for Noisy Environments

Ryan Edric Nashota
Student ID: 33508129

November 18, 2025

Contents

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

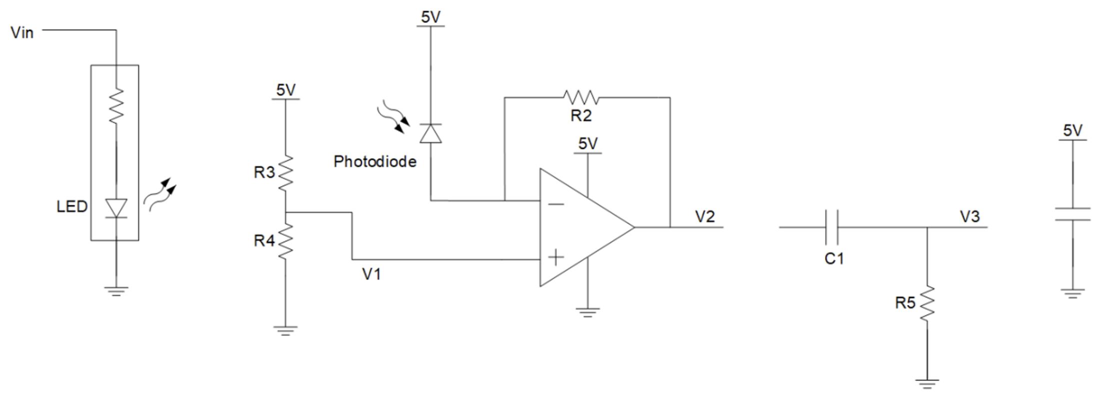

Figure 1: Exercise 2 Circuit Diagram from Lab Manual

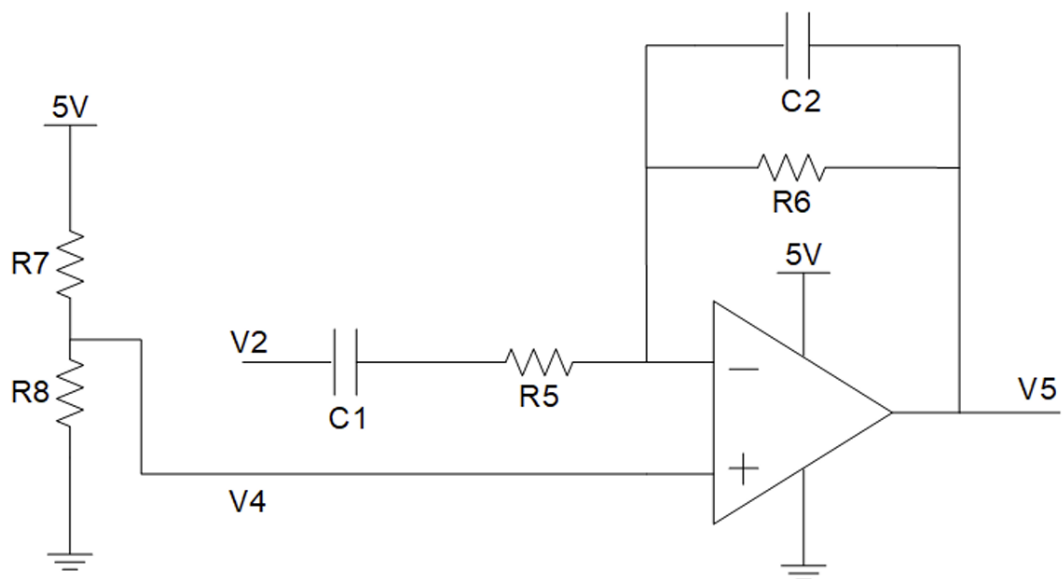

Figure 2: Exercise 3 Circuit Diagram from Lab Manual

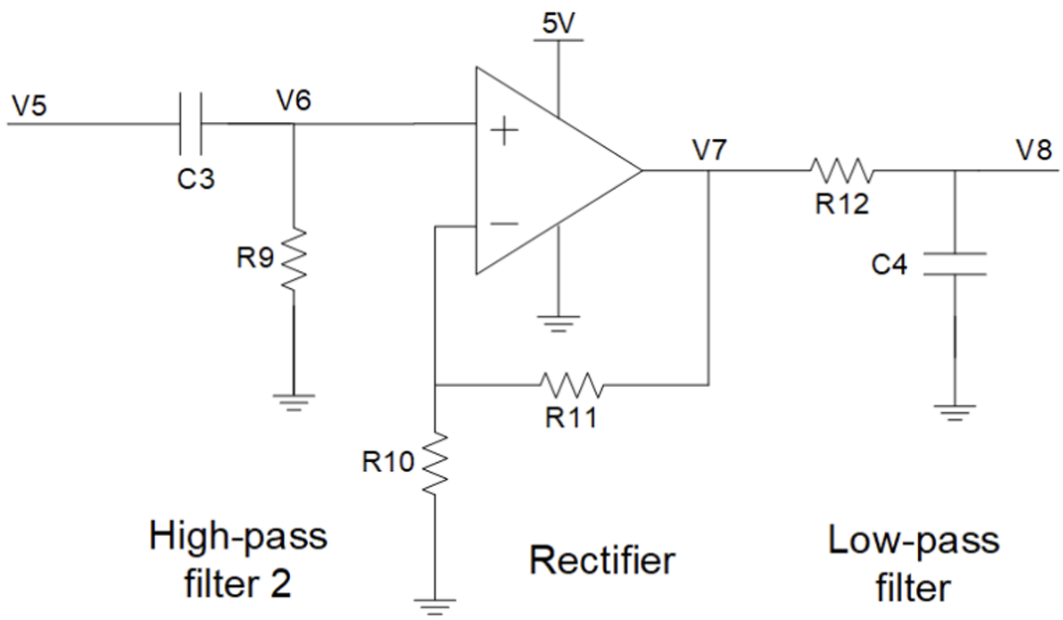

Figure 3: Exercise 4 Circuit Diagram from Lab Manual

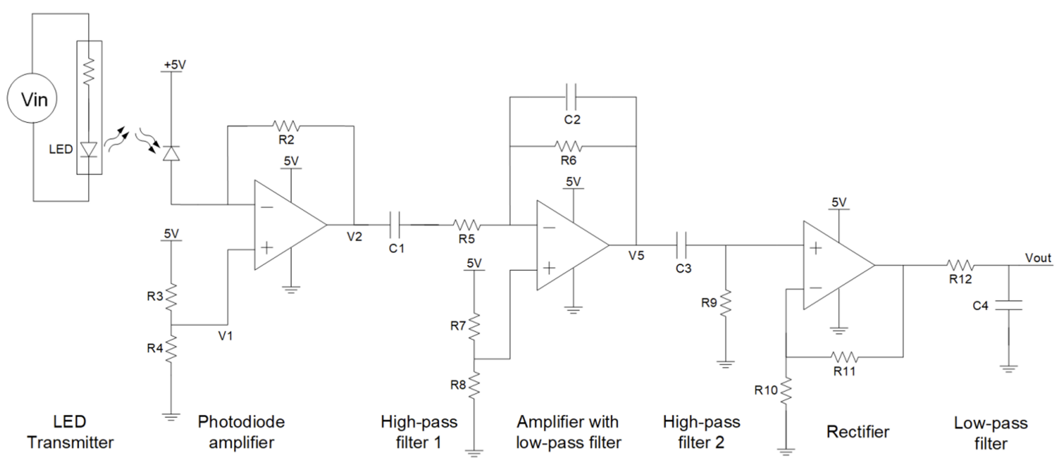

Figure 4: Complete circuit

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 Vin = 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 μA, 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 $k\Omega$ 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 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 $k\Omega$ for R3 and 11 $k\Omega$ for R4 (combination of 10 $k\Omega$ and 1 $k\Omega$ resistors in series).

3.3 Cut-off Frequency Design

Question

3. Select the value of C1 and R5 to give a cut-off frequency of 100 Hz (i.e. $\omega c = 500$ rad/s).

Calculation

$$\frac{1}{R_5C_1} = 2\pi \cdot 100 \Rightarrow R_5C_1 = \frac{1}{2\pi \cdot 100} \approx 1.59 \times 10^{-3}\,\mathrm{s}$$

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

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

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 V2 while covering and uncovering the photodiode.

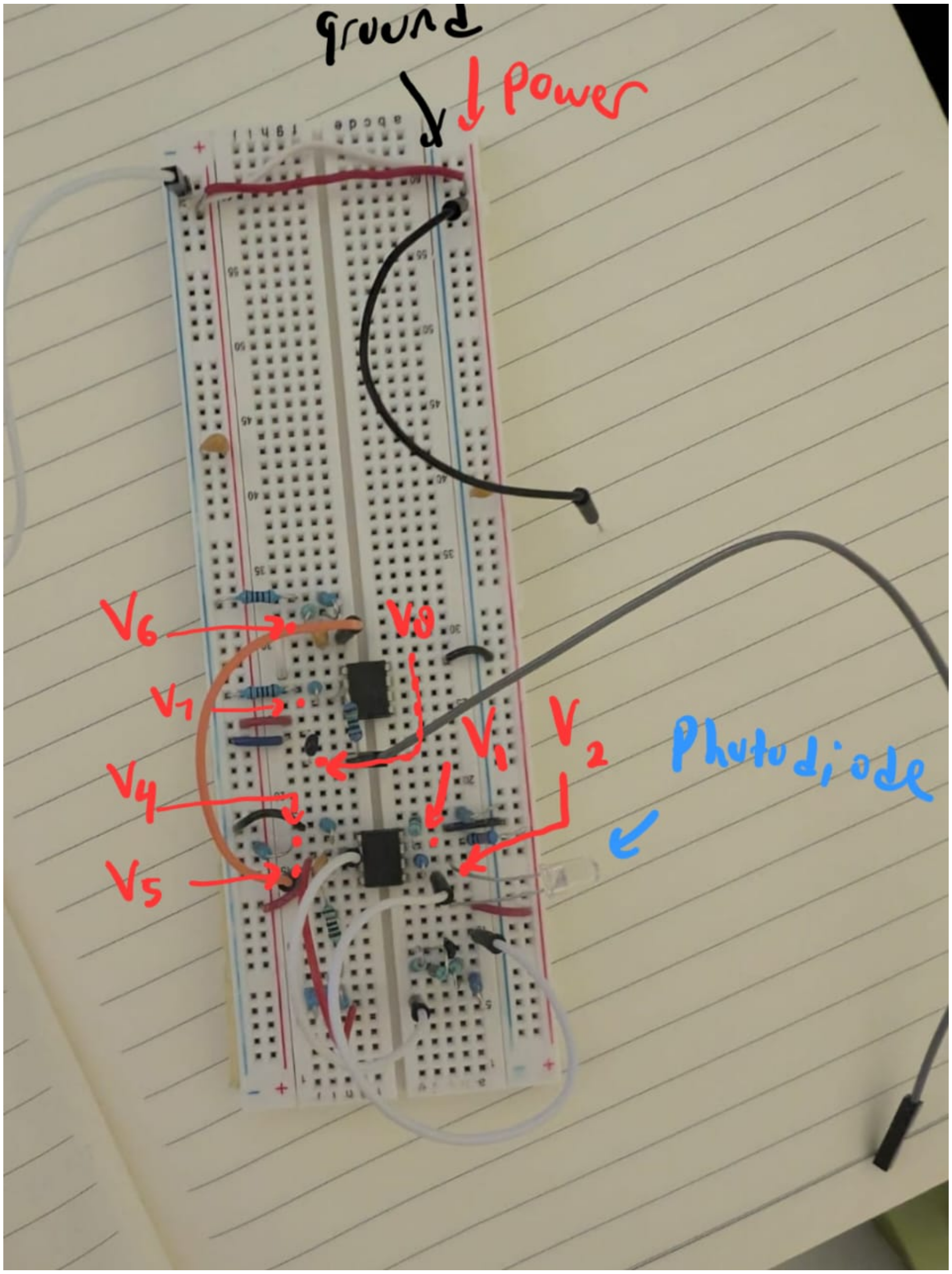

Figure 5: Completed Lab 4 Circuit on Breadboard

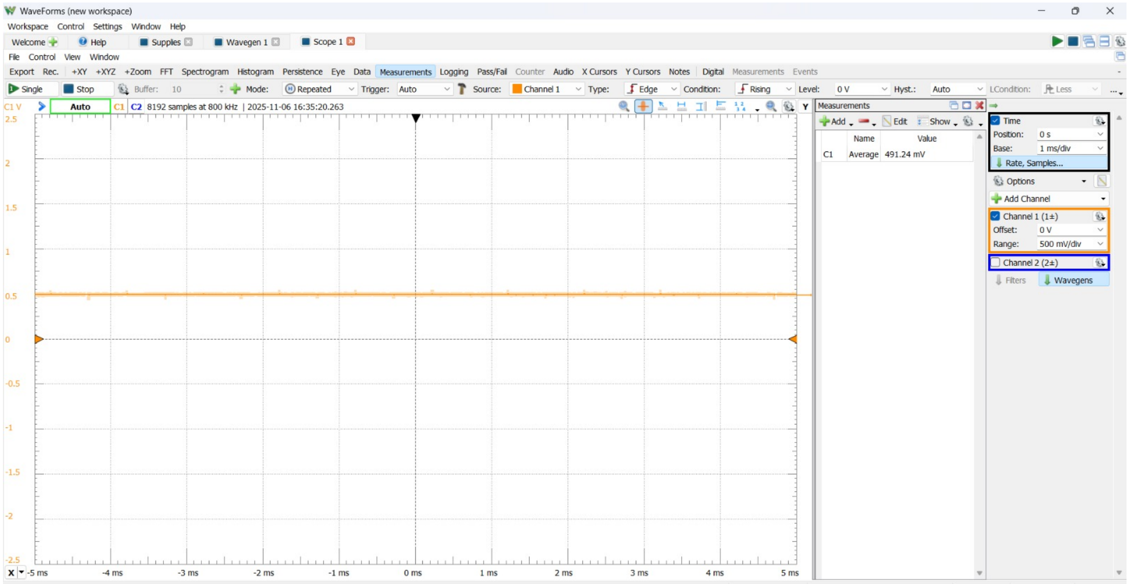

Figure 6: V2 photodiode open to ambient light

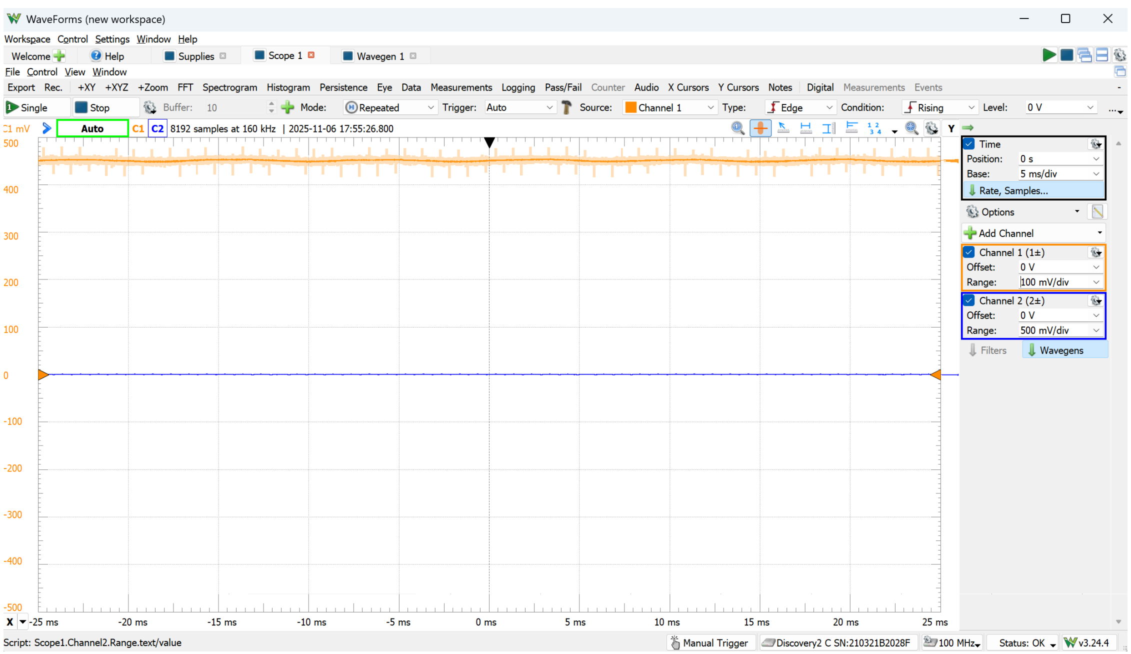

Figure 7: V2 photodiode closed with hand

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.

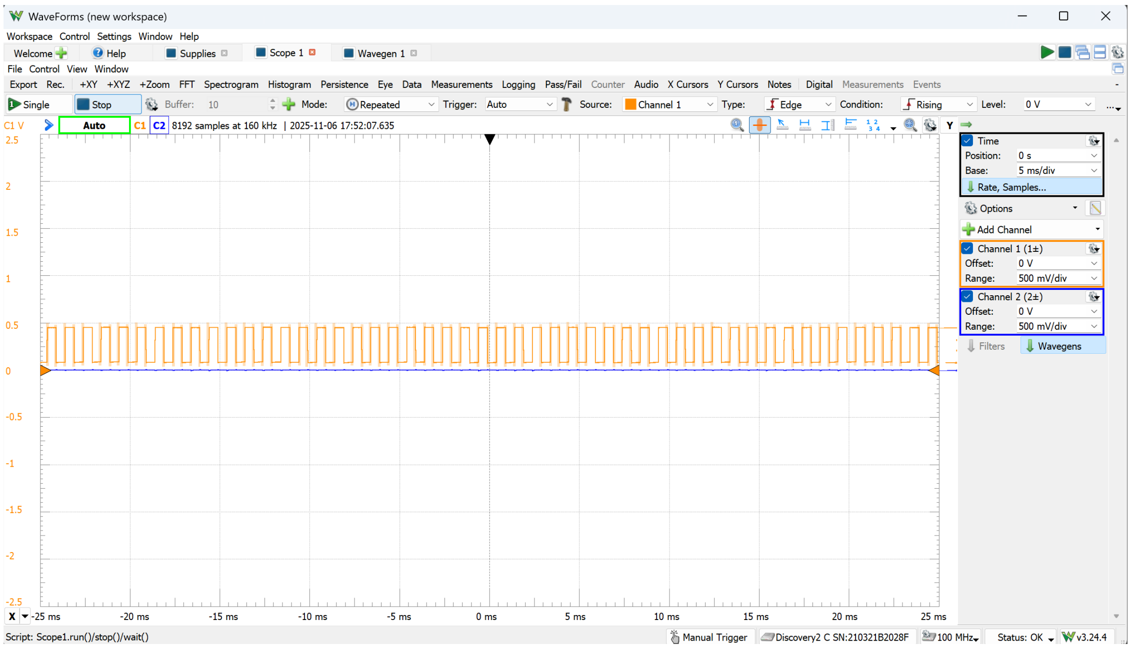

Figure 8: 1khz LED signal detected at photodiode when close

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.

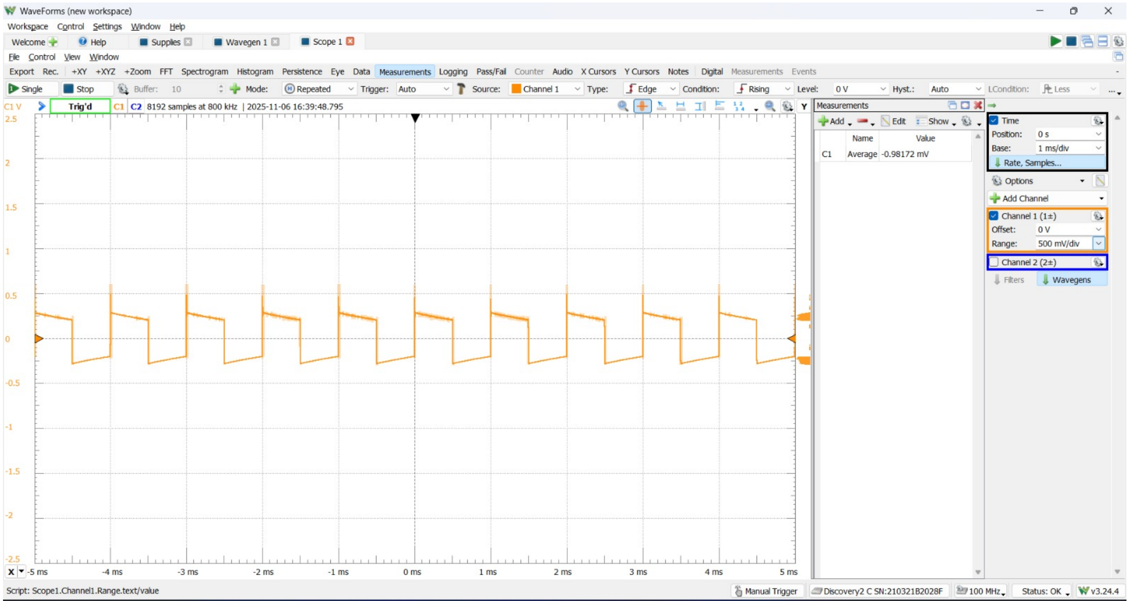

Figure 9: V3 high-pass filtered LED signal at close distance

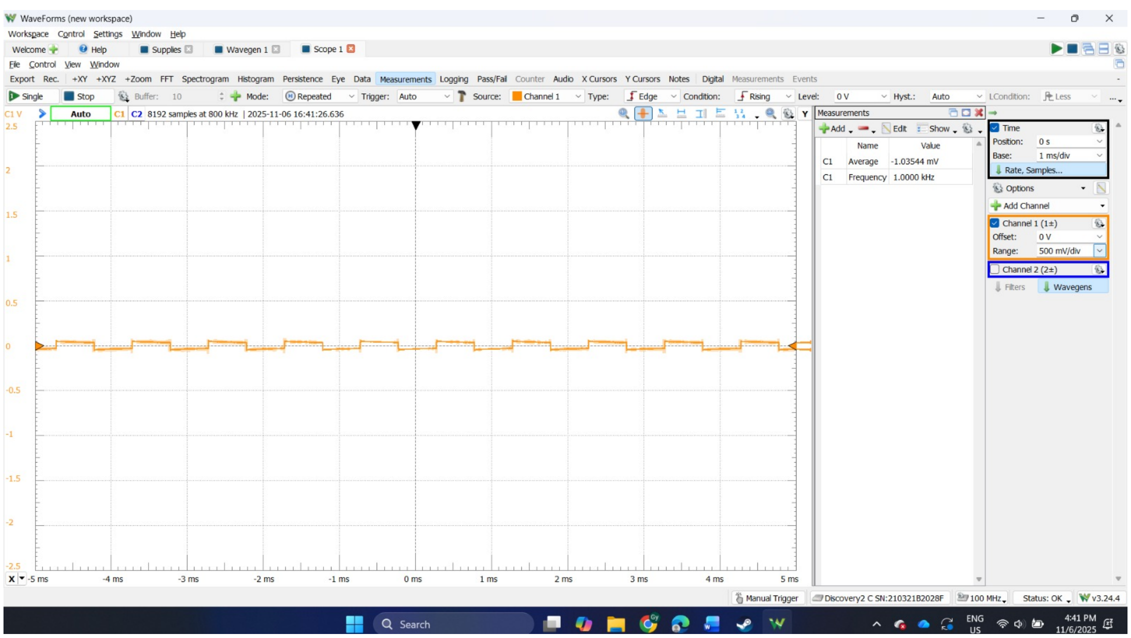

Figure 10: V3 high-pass filtered LED signal at far distance

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 V4 = 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 V4 = 2.5V, and also R6 to get a gain of -10.

Calculation

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

$$\Rightarrow R_8 = \tfrac{1}{2}(R_7 + R_8) \qquad \Rightarrow R_7 = R_8$$

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

Calculation

$$-10 = \frac{R_6}{R_5} \qquad \Rightarrow R_6 = -10\, R_5$$

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

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 $\geq$ 16 kHz (i.e. $\omega c \geq 10^5$ rad/s).

Calculation

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

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

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

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

$$\text{So choose } C_2 = 56\ \text{pF}.$$

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.

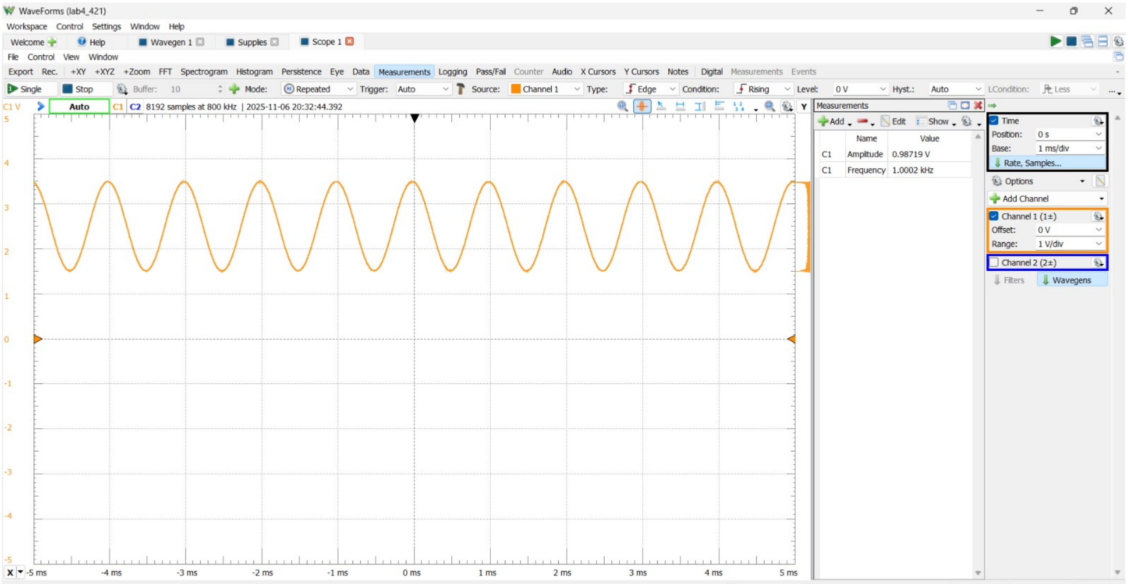

Figure 11: Exercise 3 Gain Stage Output with 100mV 1kHz Sine Wave Input

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 (V2) to the output of the photodiode amplifier.

After you verified our exercise 3 circuit using the signal generator, connect the input of this circuit (V2) 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.

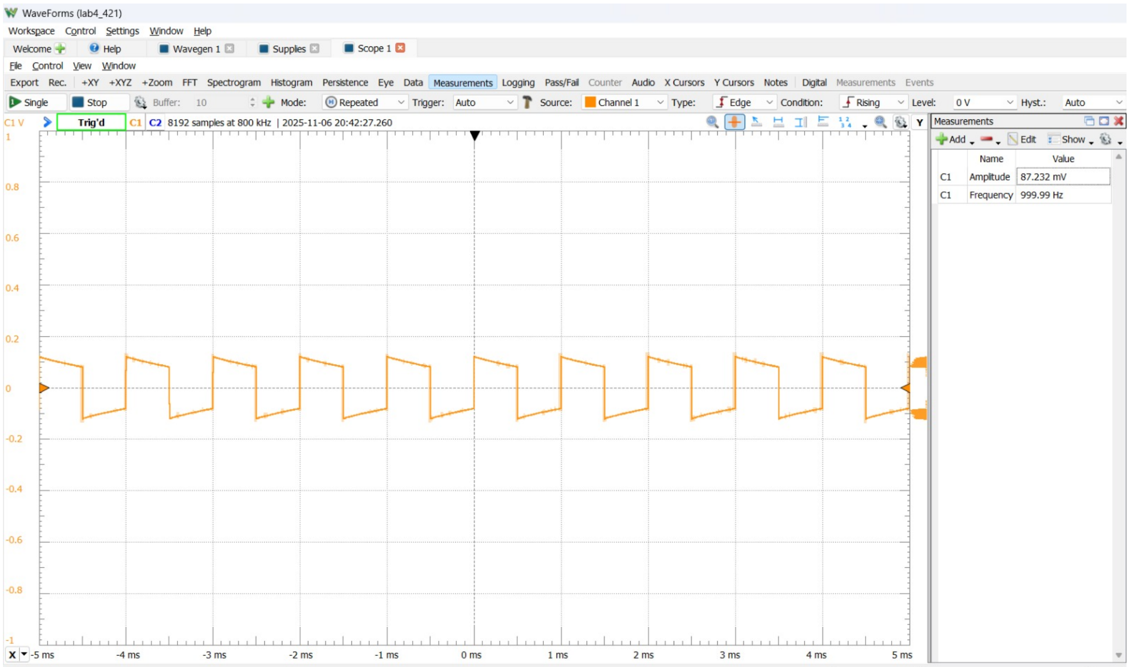

Figure 12: Square wave response from photodiode

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 C1, C2, R5, and R6 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 nF; C2 = 56 pF; R5 = 16 kΩ; R6 = 160 kΩ;

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. $\omega c = 500$ 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

$$
\begin{aligned}
1 + \frac{R_{11}}{R_{10}} &= 11 \\
\frac{R_{11}}{R_{10}} &= 10 \\
R_{11} &= 10R_{10} \\
R_{10} &= 47 \text{ k}\Omega \\
R_{11} &= 470 \text{ k}\Omega.
\end{aligned}
$$

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. ωc = 10 rad/s).

Calculation

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

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

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

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

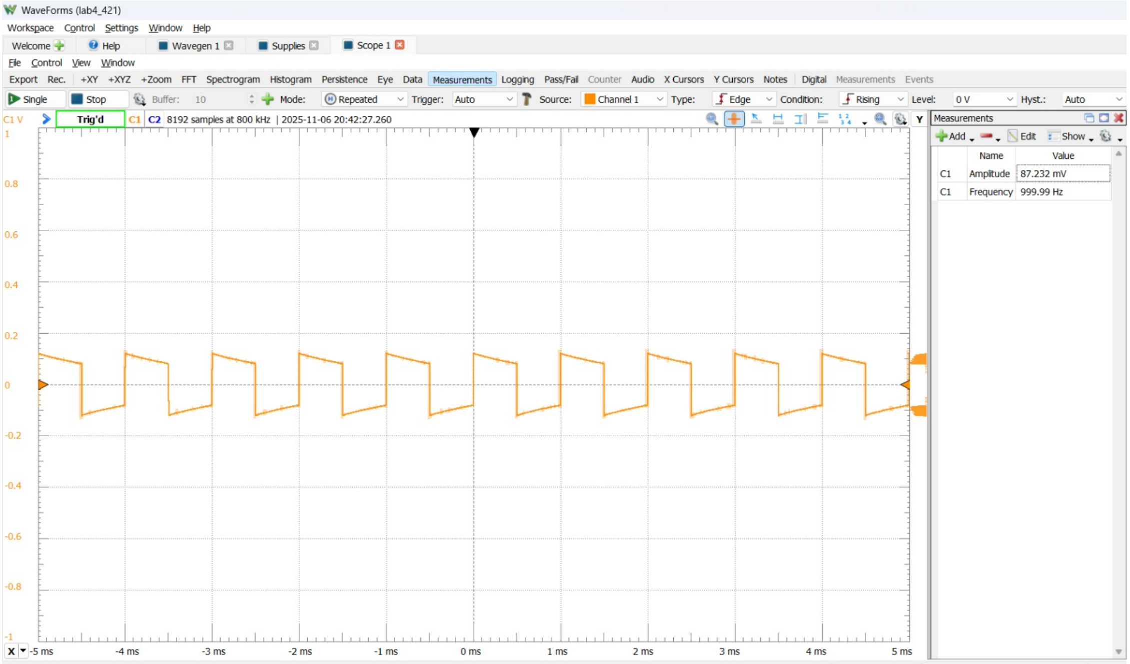

Figure 13: V6 output

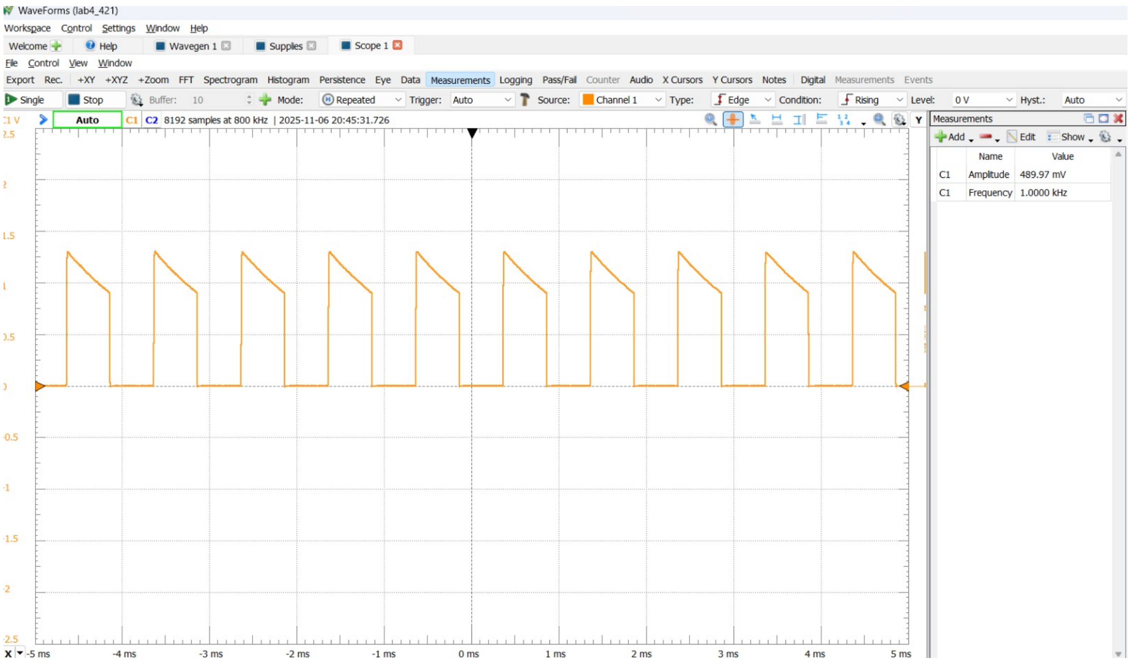

Figure 14: V7 output

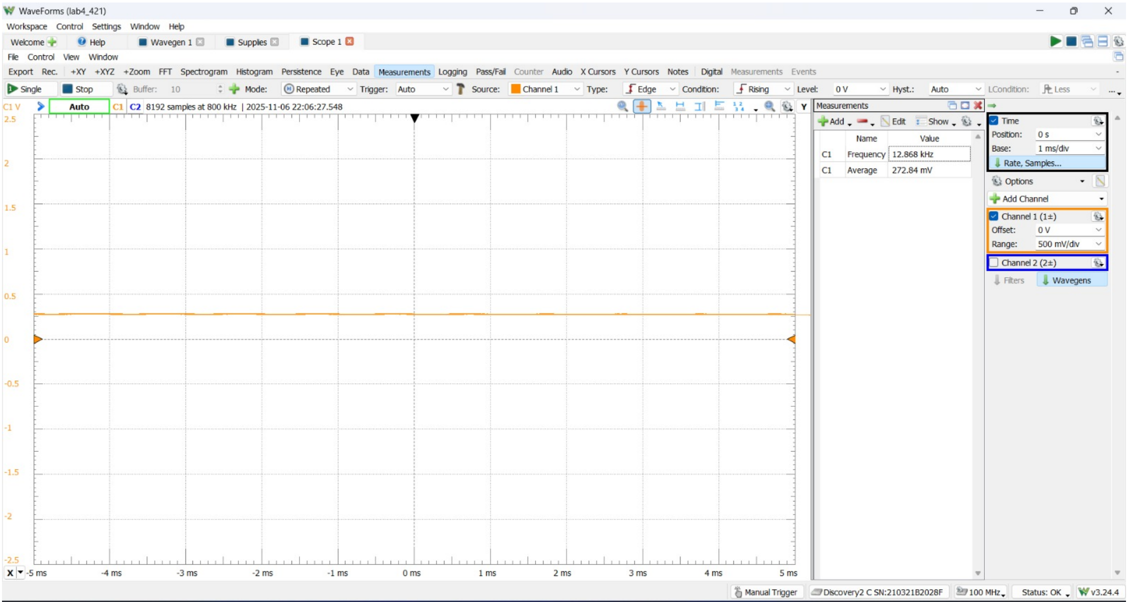

Figure 15: V8 output

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: C3 = 100 nF; R9 = 16 kΩ; R10 = 47 kΩ; R11 = 470 kΩ; R12 = 100 kΩ; C4 = 1 µF;

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 Vout is between 0 and 2.5V. If necessary, adjust the rectifier gain by changing the value of R10 and R11 to get Vout in this range.

Figure 16 is an example of a moderate range reading:

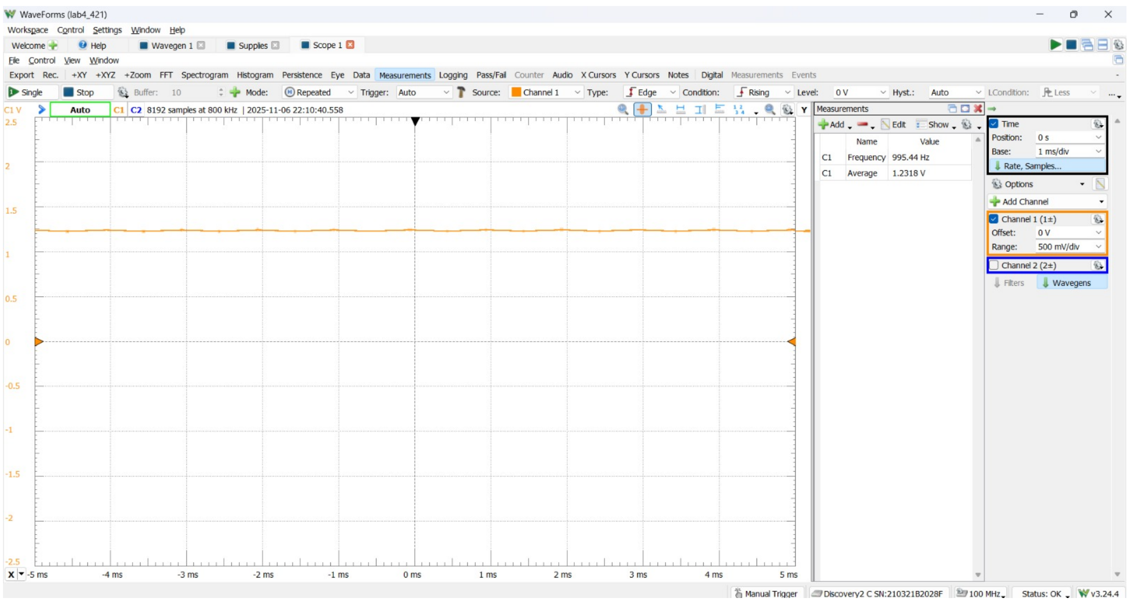

Figure 16: Final circuit output

6.3 Final Component Values

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

Question

Final values of circuit components: R10 = $47\,\mathrm{k\Omega}$; R11 = $470\,\mathrm{k\Omega}$;

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.

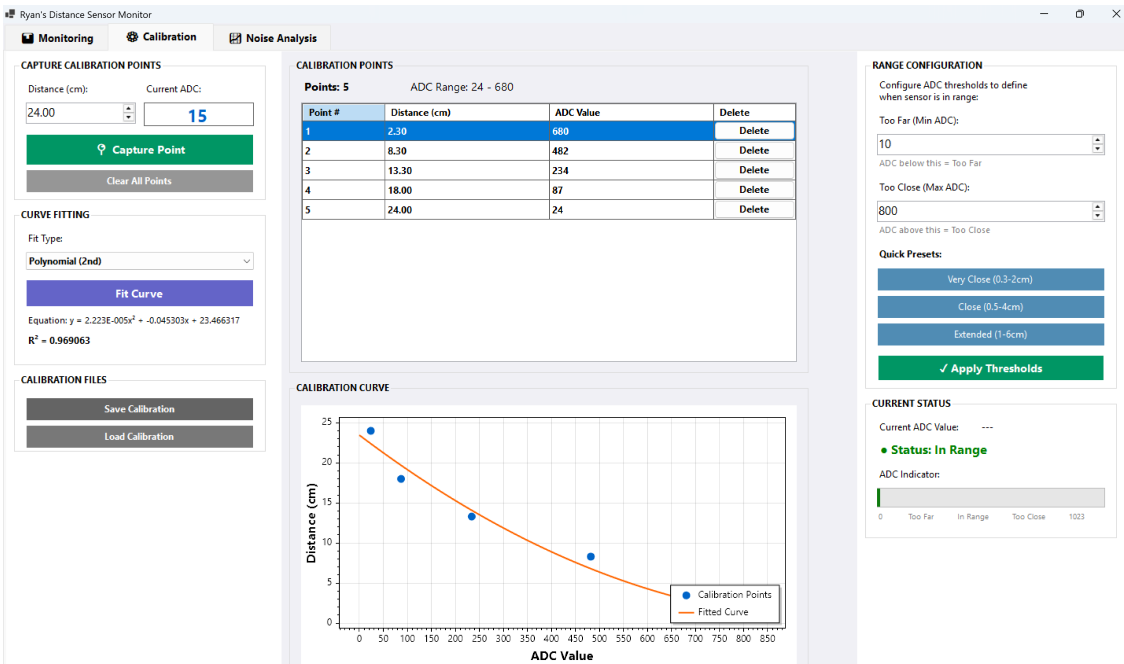

Figure 17: Calibration Curve

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 Selected C# Functions

The complete WinForms acquisition project lives in `../ryan_lab4_sensor`. This appendix highlights the routines referenced during grading to document how serial packets are decoded, calibrated, analyzed, and displayed.

A.1 Serial Packet Decoder

Listing 1: SerialPortService.ProcessByte

```
        private void ProcessByte(byte data)
        {
            // State machine for packet parsing
            if (_bufferIndex == 0)
            {
```

```
                // Looking for start byte
                if (data == START_BYTE)
                {
                    _buffer[0] = data;
                    _bufferIndex = 1;
                }
            }
            else if (_bufferIndex == 1)
            {
                // MS5B (most significant 5 bits)
                _buffer[1] = data;
                _bufferIndex = 2;
            }
            else if (_bufferIndex == 2)
            {
                // LS5B (least significant 5 bits)
                _buffer[2] = data;

                // Reassemble 10-bit ADC value
                int ms5b = _buffer[1] & 0x1F;  // Mask to 5 bits
                int ls5b = _buffer[2] & 0x1F;  // Mask to 5 bits
                int adcValue = (ms5b << 5) | ls5b;  // Combine into
                   10-bit value

                // Raise event with ADC data
                AdcDataReceived?.Invoke(this, new
                   AdcDataReceivedEventArgs(adcValue));

                // Reset for next packet
                _bufferIndex = 0;
            }
        }
```

A.2 Calibration Curve Fit

Listing 2: CalibrationService.PerformCalibration

```
        public CalibrationData PerformCalibration(List<
           CalibrationPoint> points, FitType fitType)
        {
            if (points == null || points.Count < 2)
            {
                throw new ArgumentException("At least 2 calibration
                   points are required.");
            }

            var calibration = new CalibrationData
            {
```

```
            Points = new List<CalibrationPoint>(points),
            FitType = fitType
        };

        // Extract x (ADC) and y (Distance) values
        double[] xData = points.Select(p => (double)p.AdcValue).ToArray();
        double[] yData = points.Select(p => p.Distance).ToArray();

        try
        {
            switch (fitType)
            {
                case FitType.Linear:
                    calibration.Coefficients = FitLinear(xData, yData, out double rSquaredLinear);
                    calibration.RSquared = rSquaredLinear;
                    calibration.Equation = $"y = {calibration.Coefficients[0]:F6}x + {calibration.Coefficients[1]:F6}";
                    break;

                case FitType.Polynomial2:
                    calibration.Coefficients = FitPolynomial(xData, yData, 2, out double rSquared2);
                    calibration.RSquared = rSquared2;
                    calibration.Equation = $"y = {calibration.Coefficients[0]:E3}x  + {calibration.Coefficients[1]:F6}x + {calibration.Coefficients[2]:F6}";
                    break;

                case FitType.Polynomial3:
                    calibration.Coefficients = FitPolynomial(xData, yData, 3, out double rSquared3);
                    calibration.RSquared = rSquared3;
                    calibration.Equation = $"y = {calibration.Coefficients[0]:E3}x  + {calibration.Coefficients[1]:E3}x  + {calibration.Coefficients[2]:F6}x + {calibration.Coefficients[3]:F6}";
                    break;

                case FitType.Power:
                    calibration.Coefficients = FitPower(xData, yData, out double rSquaredPower);
```

```
                        calibration.RSquared = rSquaredPower;
                        calibration.Equation = $"y = {calibration.Coefficients[0]:F6}  x^{calibration.Coefficients[1]:F6}";
                        break;

                    case FitType.Inverse:
                        calibration.Coefficients = FitInverse(xData, yData, out double rSquaredInv);
                        calibration.RSquared = rSquaredInv;
                        calibration.Equation = $"y = {calibration.Coefficients[0]:F6} / (x - {calibration.Coefficients[1]:F6}) + {calibration.Coefficients[2]:F6}";
                        break;
                }

                // Note: MinAdcThreshold and MaxAdcThreshold are set by user configuration,
                // not automatically from calibration points
            }
            catch (Exception ex)
            {
                throw new InvalidOperationException($"Curve fitting failed: {ex.Message}", ex);
            }

            return calibration;
        }
```

A.3 Noise-Test Summary Generator

Listing 3: NoiseAnalysisService.GetComparisonSummary

```
        public string GetComparisonSummary()
        {
            if (_testResults.Count == 0)
                return "No tests available for comparison.";

            var sb = new StringBuilder();
            sb.AppendLine("=== NOISE ANALYSIS COMPARISON ===\n");

            // Group by position
            var middleTests = _testResults.Where(t => t.Position == TestPosition.MiddleRange).ToList();
            var nearTests = _testResults.Where(t => t.Position == TestPosition.NearExtreme).ToList();
```

```
            var farTests = _testResults.Where(t => t.Position == TestPosition.FarExtreme).ToList();

            if (middleTests.Any())
            {
                sb.AppendLine("MIDDLE RANGE:");
                foreach (var test in middleTests)
                {
                    sb.AppendLine($"  Test #{test.TestNumber}: Mean={test.MeanDistance:F4} cm, RMS Noise={test.StandardDeviation:F4} cm");
                }
                sb.AppendLine($"  Average RMS Noise: {middleTests.Average(t => t.StandardDeviation):F4} cm\n");
            }

            if (nearTests.Any())
            {
                sb.AppendLine("NEAR EXTREME (Close):");
                foreach (var test in nearTests)
                {
                    sb.AppendLine($"  Test #{test.TestNumber}: Mean={test.MeanDistance:F4} cm, RMS Noise={test.StandardDeviation:F4} cm");
                }
                sb.AppendLine($"  Average RMS Noise: {nearTests.Average(t => t.StandardDeviation):F4} cm\n");
            }

            if (farTests.Any())
            {
                sb.AppendLine("FAR EXTREME:");
                foreach (var test in farTests)
                {
                    sb.AppendLine($"  Test #{test.TestNumber}: Mean={test.MeanDistance:F4} cm, RMS Noise={test.StandardDeviation:F4} cm");
                }
                sb.AppendLine($"  Average RMS Noise: {farTests.Average(t => t.StandardDeviation):F4} cm\n");
            }

            // Comparison analysis
            if (middleTests.Any() && (nearTests.Any() || farTests.Any()))
            {
```

```
                double middleRms = middleTests.Average(t => t.
                   StandardDeviation);
                sb.AppendLine("=== COMPARISON ===");

                if (nearTests.Any())
                {
                    double nearRms = nearTests.Average(t => t.
                       StandardDeviation);
                    double nearDiff = nearRms - middleRms;
                    double nearRatio = middleRms > 0 ? nearRms /
                       middleRms : 0;
                    sb.AppendLine($"Near Extreme vs Middle: {
                       nearDiff:+0.0000;-0.0000} cm ({nearRatio:F2}x
                       )");
                }

                if (farTests.Any())
                {
                    double farRms = farTests.Average(t => t.
                       StandardDeviation);
                    double farDiff = farRms - middleRms;
                    double farRatio = middleRms > 0 ? farRms /
                       middleRms : 0;
                    sb.AppendLine($"Far Extreme vs Middle: {farDiff
                       :+0.0000;-0.0000} cm ({farRatio:F2}x)");
                }
            }

            return sb.ToString();
        }
```

A.4 UI Update Handler

Listing 4: MainForm.SerialPort$_A$*dcDataReceived*

```
        private void SerialPort_AdcDataReceived(object? sender,
           AdcDataReceivedEventArgs e)
        {
            if (InvokeRequired)
            {
                Invoke(new Action(() => SerialPort_AdcDataReceived(
                   sender, e)));
                return;
            }

            _currentAdcValue = e.AdcValue;
            double voltage = _currentAdcValue * 3.3 / 1023.0;
            double distance = 0;
```

```
            bool isInRange = true;
            string rangeStatus = "In Range";

            if (_currentCalibration != null && _currentCalibration.Coefficients.Length > 0)
            {
                distance = _currentCalibration.ConvertAdcToDistance(_currentAdcValue);
                isInRange = _currentCalibration.IsInRange(_currentAdcValue);
                rangeStatus = _currentCalibration.GetRangeStatus(_currentAdcValue);
            }

            // Update monitoring tab
            lblAdcValue.Text = $"ADC: {_currentAdcValue} / 1023";
            lblVoltage.Text = $"Voltage: {voltage:F3} V";
            lblDistance.Text = $"Distance: {distance:F2} cm";

            if (rangeStatus == "In Range")
            {
                lblRangeStatus.Text = "    In Range";
                lblRangeStatus.ForeColor = Color.Green;
            }
            else if (rangeStatus == "Too Close")
            {
                lblRangeStatus.Text = "    Too Close";
                lblRangeStatus.ForeColor = Color.OrangeRed;
            }
            else
            {
                lblRangeStatus.Text = "    Too Far";
                lblRangeStatus.ForeColor = Color.Red;
            }

            // Update calibration tab
            lblCurrentAdcIndicator.Text = _currentAdcValue.ToString();
            pbAdcIndicator.Value = Math.Min(Math.Max(_currentAdcValue, 0), 1023);

            if (rangeStatus == "In Range")
            {
                lblThresholdStatus.Text = "    Status: In Range";
                lblThresholdStatus.ForeColor = Color.Green;
            }
            else if (rangeStatus == "Too Close")
```

```
            {
                lblThresholdStatus.Text = "    Status: Too Close";
                lblThresholdStatus.ForeColor = Color.OrangeRed;
            }
            else
            {
                lblThresholdStatus.Text = "    Status: Too Far";
                lblThresholdStatus.ForeColor = Color.Red;
            }

            double elapsedSeconds = (DateTime.Now - _startTime).TotalSeconds;
            _timeData.Add(elapsedSeconds);
            _adcData.Add(_currentAdcValue);
            _voltageData.Add(voltage);
            _distanceData.Add(distance);

            if (_timeData.Count > 300)
            {
                _timeData.RemoveAt(0);
                _adcData.RemoveAt(0);
                _voltageData.RemoveAt(0);
                _distanceData.RemoveAt(0);
            }

            if (_isLogging)
            {
                var reading = new SensorReading(_currentAdcValue, distance, isInRange);
                _dataLogger.AddReading(reading);
                lblSampleCount.Text = $"Samples: {_dataLogger.Count}";
            }
        }
```