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

Gyan Edbert Zesiro
Student ID: 38600060

November 17, 2025

## Contents

# 1 Introduction

This lab investigates the design of an optical distance sensor built from discrete op amp stages. Each exercise reinforces a different portion of the signal chain—from generating the optical carrier, through transimpedance conversion and filtering, to rectification, digitization, and calibration.

# 2 Methodology

The experimental procedure was organized into three phases that progressively built up a complete optical distance sensor, from the analog front-end to digital acquisition and calibration. Each phase was completed only after verifying correct operation of the previous one.

## Phase 1: Analog Front-End

a) Configure the Analog Discovery 2 (AD2) waveform generator to drive the red LED with a 5 V amplitude square wave and 2.5 V DC offset. Verify visible flashing at 1 Hz and continuous illumination at 1 kHz.

b) Mechanically mount the LED on the linear rail and align it with the photodiode on the breadboard, ensuring smooth motion and stable geometry over the full travel range.

c) Design and assemble the photodiode transimpedance amplifier by selecting $R_2$ to convert a 1 µA photocurrent into a 100 mV deviation about a bias voltage $V_1$.

d) Design the bias network using $R_3$ and $R_4$ to set $V_1 \approx 0.5\,\text{V}$ from the 5 V supply, then build and verify the bias node on the breadboard.

e) Choose and install $R_5$ and $C_1$ to implement the first high-pass filter with a cutoff near 100 Hz, and confirm the expected frequency response using the AD2.

f) Characterize the effect of ambient light by observing $V_2$ with the photodiode alternately exposed and covered, documenting the change in DC level and the small 1 kHz carrier component.

g) Connect the high-pass filter output to the AC gain stage and verify that the 1 kHz waveform amplitude changes predictably with LED–photodiode separation over the specified distance range.

## Phase 2: Signal Conditioning and Rectification

a) Design and build a second RC high-pass filter using $C_3$ and $R_9$ with the same cutoff frequency as the first stage to further suppress low-frequency ambient-light variations.

b) Design and construct a non-inverting rectifier stage by selecting $R_{10}$ and $R_{11}$ to achieve an overall gain of approximately 11, ensuring that the rectified signal remains within the op-amp linear range.

c) Design and implement a low-pass RC filter using $C_4$ and $R_{12}$ with a cutoff near 1.6 Hz to extract a slowly varying DC voltage proportional to the 1 kHz carrier amplitude.

d) Validate the signal-conditioning chain by driving $V_5$ with a known 1 kHz square wave (about 100 mV peak-to-peak) from the AD2, probing the signal after each stage (high-pass, rectifier, low-pass) and confirming the expected waveforms.

e) Connect the output of the Phase 1 amplifier chain to this rectifier/low-pass block and adjust $R_{10}$ and $R_{11}$ if necessary so that the final output $V_{\text{out}}$ remains within the target 0 V to 2.5 V range over the full LED–photodiode separation.

f) Measure and document the steady-state output voltage versus separation distance, confirming that the circuit neither saturates at close spacing nor loses the signal at the maximum separation.

## Phase 3: Digitization, Software, and Calibration

a) Develop firmware for the MSP430FR5739 to sample $V_{\text{out}}$ using a 10-bit ADC over a 0 V to 3.3 V input range, and format each sample into two bytes (MS5B and LS5B) preceded by a synchronization byte (255).

b) Implement a C# application that opens the serial port, reassembles the MS5B and LS5B fields into a 10-bit digital value, and continuously displays and plots the incoming data stream.

c) Extend the C# user interface to log acquired data to file and provide real-time visualization suitable for tuning the sensor and verifying the dynamic behavior.

d) Perform a calibration experiment by recording ADC output at several known separation distances across the usable range; fit an appropriate function relating ADC code to distance using numerical tools.

e) Incorporate the calibration function into the C# program so that it converts each ADC reading into a distance estimate, displays both raw and converted values, and flags out-of-range conditions.

f) Quantify the sensor resolution by logging the converted distance at a fixed nominal position for an extended period, computing the standard deviation of the measured distance, and repeating near the ends of the operating range to compare noise performance.

# 3 Phase 1: Analog Front-End

## 3.1 Exercise 1: Optical Transmitter

### 3.1.1 Question 1: Assemble and Power the LED

The LED transmitter was mounted on the linear rail and connected to the Analog Discovery 2 (AD2) waveform generator. Vin was configured as a 5 V amplitude square wave with a 2.5 V DC offset so the LED could be pulsed at the desired current.

### 3.1.2 Question 2: Verify 1 Hz Modulation

With the generator at 1 Hz the LED visibly flashed, demonstrating that the wiring and mechanical alignment were correct and that the LED could traverse the slider smoothly.

### 3.1.3 Question 3: Verify 1 kHz Modulation

if you choose 1 kHz waveform, the LED appear continuously on, indicating that the chosen frequency exceeds the flicker fusion threshold and is suitable as the optical carrier for later exercises.

## 3.2 Exercise 2: Photodiode Amplifier and High-Pass Filter

### 3.2.1 Question 1: Choose $R_2$

Using the constraint $\Delta V_2 = 100\,\mathrm{mV}$ for a $1\,\mu\mathrm{A}$ photocurrent,

$$R_2 = \frac{\Delta V_2}{I_d} = \frac{0.1}{1 \times 10^{-6}} = 100\,\mathrm{k\Omega}.$$

### 3.2.2 Question 2: Choose $R_3$ and $R_4$

Bias $V_1$ at $0.5\,\mathrm{V}$ using a divider referenced to $5\,\mathrm{V}$:

$$V_1 = \frac{R_4}{R_3 + R_4} V_{\mathrm{ref}} = 0.5.$$

Selecting $R_3 = 200\,\mathrm{k\Omega}$ and $R_4 = 22\,\mathrm{k\Omega}$ yields

$$V_1 = \frac{22}{200 + 22} \times 5 \approx 0.495\,\mathrm{V},$$

which is sufficiently close to the target bias.

### 3.2.3 Question 3: Choose $R_5$ and $C_1$

For the filter design, the transfer function amplitude is equal to $\frac{1}{\sqrt{2}}$ at the cutoff frequency:

$$\left| \frac{R}{R + \frac{1}{\omega C}} \right| = \frac{1}{\sqrt{2}}$$

$$\left| \frac{\omega RCj}{\omega RCj + 1} \right| = \frac{1}{\sqrt{2}}$$

$$\frac{(\omega RC)^2}{1 + (\omega RC)^2} = \frac{1}{2}$$

$$2(\omega RC)^2 = 1 + (\omega RC)^2$$

$$(\omega RC)^2 = 1$$

$$\omega RC = 1$$

$$RC = \frac{1}{\omega} = \frac{1}{2\pi \cdot 100}$$

$$RC \approx \boxed{1.59155 \times 10^{-3} \text{ s}}$$

$\mathrm{R}_5 = 1.6\ \mathrm{k}\Omega, \quad \mathrm{C}_1 = 1\ \mu\mathrm{F}$ are good enough

### 3.2.4 Question 4: Demonstrate Ambient-Light Impact

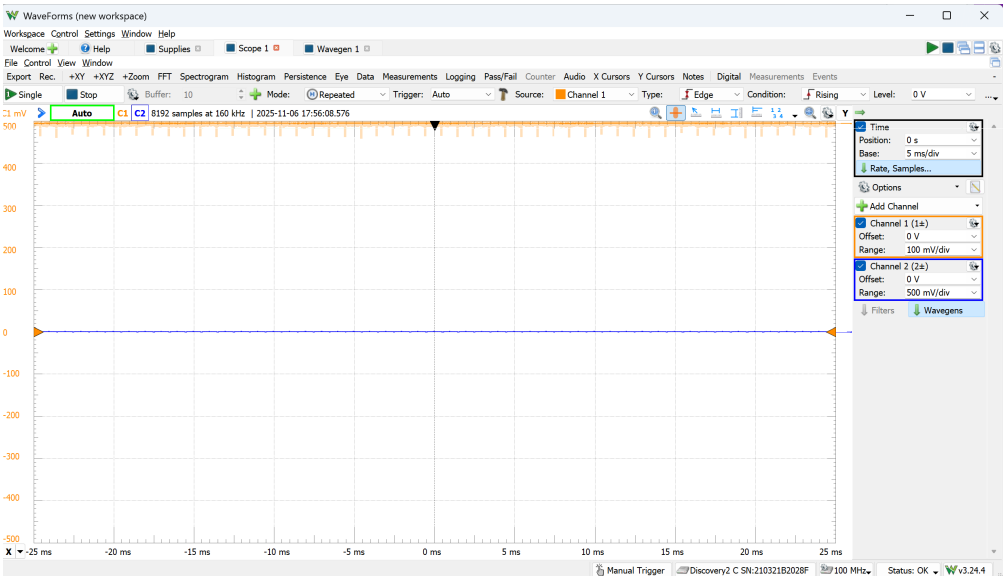


Figure 1: Waveform when photodiode is exposed to ambient light.

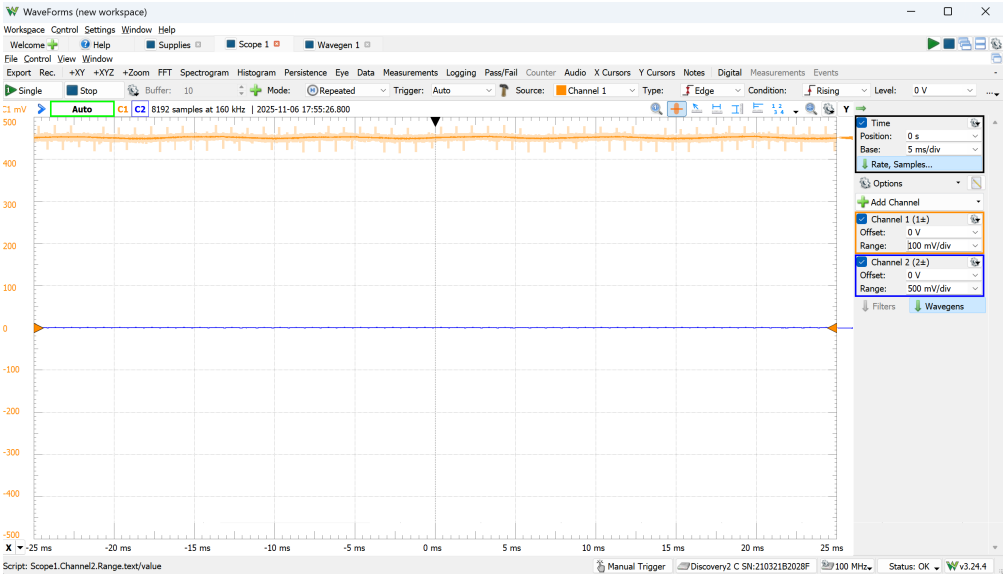


Figure 2: Waveform when photodiode is covered.

Figure 1 shows $V_2$ with the photodiode exposed to room light while Figure 2 shows the effect of covering it. As you can see, covering the photodiode reduces the DC offset, which is what we expected.

### 3.2.5 Question 5: Detect the 1 kHz Carrier at $V_2$

First, setup the LED by using the AD2 waveform generator, connect the positive of the waveform to the anode of the LED and the negative to the cathode of the LED. Set the waveform generator to output a 2.5 V amplitude square wave with a 2.5 V DC offset at 1 kHz. Figure 3 shows the setup inside the WaveForm software.

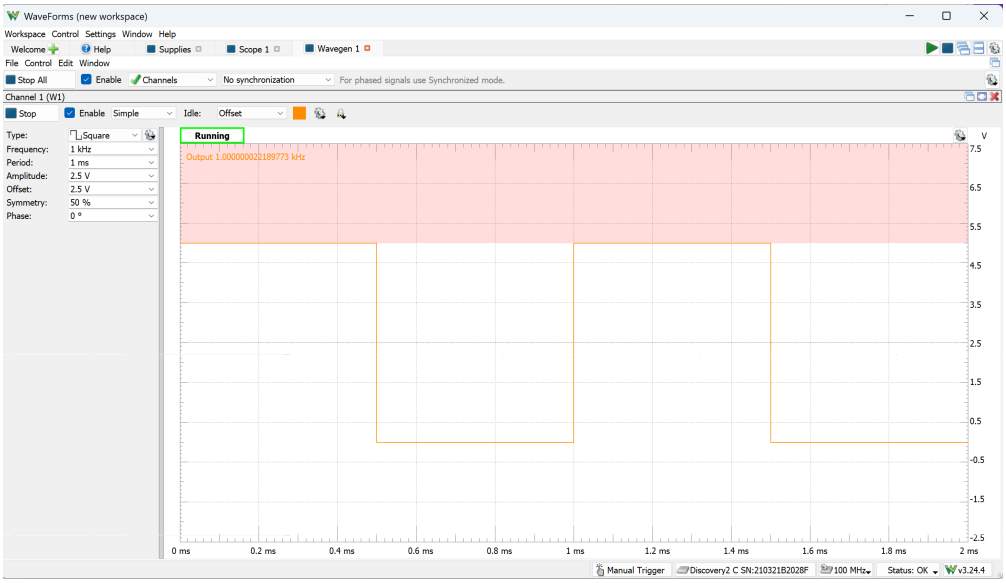


Figure 3: Waveform generator setup for LED modulation

After setting up the LED, bring the LED close to the photodiode, and observe the voltage using the AD2 oscilloscope at $V_2$.

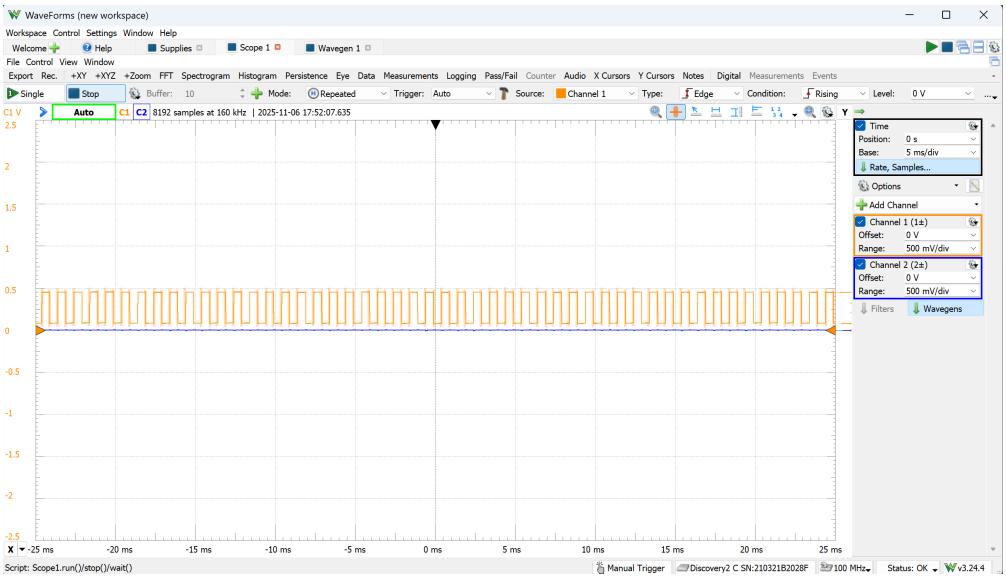


Figure 4: Waveform at $V_2$ with LED close to photodiode.

The resulting waveform is shown in Figure 4. Notice that this is a 1khz square wave signal (just count that there is 5 cycles in 5ms)

This is what we should expect, here is the math for our result:

Let the photodiode current be written as the sum of a DC component due to ambient light

and an AC component due to the modulated LED,

$$i_{\mathrm{ph}}(t) = I_{\mathrm{amb}} + \Delta I\, q(t),$$

where $q(t)$ is a unit-amplitude, zero-mean square wave of frequency $f_0 = 1$ kHz (i.e. $q(t) = \pm 1$ with 50% duty cycle), and $\Delta I$ is the amplitude of the photocurrent variation when the LED is driven.

The op-amp is configured as a transimpedance amplifier referenced to $V_1$. Because of the virtual short between the inputs, the inverting node is held at

$$V_- \approx V_+ = V_1.$$

Applying KCL at the inverting node gives

$$\frac{V_2(t) - V_1}{R_2} = -i_{\mathrm{ph}}(t),$$

so the output voltage of the transimpedance stage is

$$V_2(t) = V_1 - R_2\, i_{\mathrm{ph}}(t) = \big(V_1 - R_2 I_{\mathrm{amb}}\big) - R_2 \Delta I\, q(t).$$

Thus $V_2(t)$ consists of a DC offset

$$V_{2,\mathrm{DC}} = V_1 - R_2 I_{\mathrm{amb}}$$

plus a 1 kHz square-wave component of peak amplitude

$$V_{2,\mathrm{AC}}(t) = -R_2 \Delta I\, q(t),$$

so the peak-to-peak value of the 1 kHz component at $V_2$ is

$$V_{2,\mathrm{pp}} = 2 R_2 \Delta I.$$

As the LED–photodiode distance changes, $\Delta I$ changes, and the square-wave amplitude at $V_2$ scales linearly with the photocurrent variation.

### 3.2.6 Question 6: Observe the High-Pass Output

The result at $V_3$ is shown in Figure 5.

The node $V_2$ is now applied to the high-pass filter formed by $C_1$ and $R_5$. Taking $V_2(t)$ as the input and $V_3(t)$ as the output, the transfer function of this first-order high-pass filter is

$$H(s) = \frac{V_3(s)}{V_2(s)} = \frac{s R_5 C_1}{1 + s R_5 C_1}.$$

For a sinusoidal component of angular frequency $\omega$,

$$H(j\omega) = \frac{j\omega R_5 C_1}{1 + j\omega R_5 C_1},$$

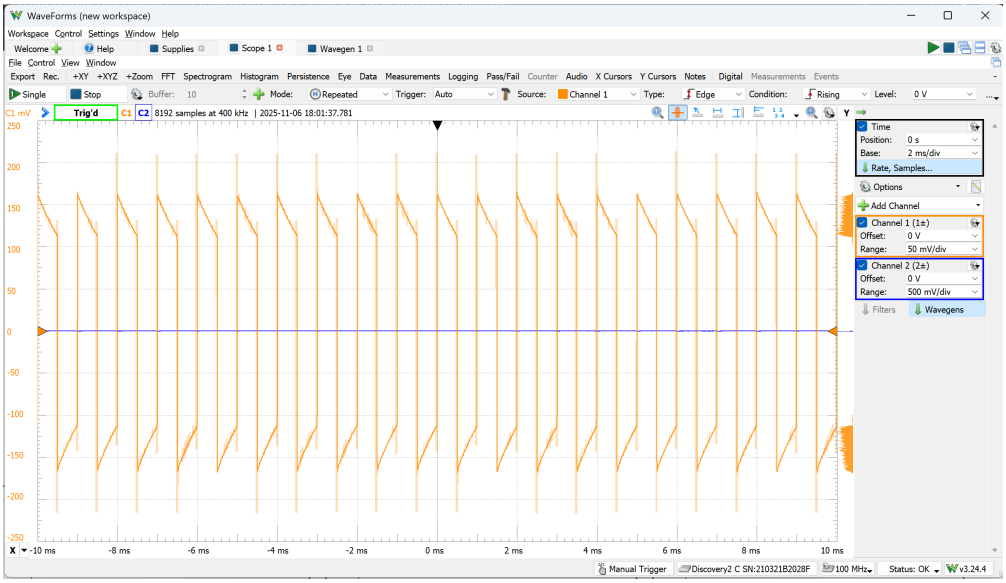


Figure 5: Waveform at $V_3$ with LED close to photodiode.

and its magnitude is

$$\left|H(j\omega)\right| = \frac{\omega R_5 C_1}{\sqrt{1+(\omega R_5 C_1)^2}}.$$

The cut-off angular frequency is chosen as

$$\omega_c = \frac{1}{R_5 C_1} \approx 500 \text{ rad/s} \quad \text{(about 100 Hz)},$$

so at the LED modulation frequency $f_0 = 1$ kHz, $\omega_0 = 2\pi f_0$, we have

$$\omega_0 R_5 C_1 = \frac{\omega_0}{\omega_c} = \frac{2\pi \cdot 1000}{500} \approx 12.6,$$

and therefore

$$\left|H(j\omega_0)\right| = \frac{12.6}{\sqrt{1+12.6^2}} \approx 0.997 \approx 1.$$

The DC term $V_{2,\mathrm{DC}}$ is completely blocked by the high-pass filter (its gain at $\omega = 0$ is zero), while the 1 kHz square-wave component passes essentially unchanged in amplitude. Therefore the output at $V_3$ is approximately

$$V_3(t) \approx \left|H(j\omega_0)\right| V_{2,\mathrm{AC}}(t) \approx -R_2 \Delta I \, q(t),$$

with a peak-to-peak value

$$V_{3,\mathrm{pp}} \approx \left|H(j\omega_0)\right| V_{2,\mathrm{pp}} \approx V_{2,\mathrm{pp}} = 2R_2 \Delta I.$$

Hence, at $V_3$ we observe a square wave at 1 kHz whose peak-to-peak amplitude is essentially the same as the AC component of $V_2$, but now centered around 0 V rather than around the bias voltage $V_1$. As the distance between the LED and photodiode is changed, $\Delta I$ changes, and the peak-to-peak amplitude at $V_3$ varies proportionally, which is what I found experimentally (and hopefully you too).

### 3.2.7 Question 7: Hardware Arrangement

Follow the instruction and you'll end up with the setup shown in the lab manual. Note that this is very crucial for later experiments. Because you need to align the photodiode and LED properly, make sure that the LED is facing the photodiode directly. I experienced a lot of issues because of misalignment, particularly I couldn't get the full range of distance measurement because the photodiode couldn't "see" the LED properly. So future mechatronics student, please pay attention to this. If you're not convinced, I have derivation once we build the full circuit.

## 3.3 Exercise 3: High-Pass Filter and AC Amplifier

### 3.3.1 Question 1: Bias Network

$$\frac{R_8}{R_8 + R_7} = \frac{1}{2} \tag{1}$$

$$2R_8 = R_8 + R_7 \tag{2}$$

$$R_8 = R_7 \tag{3}$$

We can choose, for example,

$$R_8 = R_7 = \boxed{10\ \text{k}\Omega}. \tag{4}$$

The input and feedback impedances of the op-amp stage are

$$Z_1 = R_5 + \frac{1}{sC_1} = \frac{1 + sR_5C_1}{sC_1}, \tag{5}$$

$$Z_2 = R_6 \parallel \frac{1}{sC_2} = \frac{R_6}{R_6 + \dfrac{1}{sC_2}} = \frac{R_6}{1 + sR_6C_2}. \tag{6}$$

For the non-inverting configuration, with $V_4$ at the non-inverting input and $V_2$ feeding the inverting node through $Z_1$, the output is

$$V_0 = V_4\left(1 + \frac{Z_2}{Z_1}\right) - V_2\left(\frac{Z_2}{Z_1}\right) \tag{7}$$

$$= V_4 + \frac{Z_2}{Z_1}\left(V_4 - V_2\right). \tag{8}$$

We want the magnitude of the transfer ratio to be about 10:

$$\left|\frac{Z_2}{Z_1}\right| \approx 10. \tag{9}$$

Compute the ratio:

$$\frac{Z_2}{Z_1} = \frac{\dfrac{R_6}{1+sR_6C_2}}{R_5 + \dfrac{1}{sC_1}} \tag{10}$$

$$= \frac{R_6}{1+sR_6C_2} \cdot \frac{sC_1}{1+sR_5C_1} \tag{11}$$

$$= \frac{R_6}{R_5}\,\frac{sR_5C_1}{1+sR_5C_1} \cdot \frac{1}{1+sR_6C_2}. \tag{12}$$

In the midband of interest (where the frequency-dependent factors are near unity), the dominant term is $R_6/R_5$, so we choose

$$\frac{R_6}{R_5} \approx \boxed{10}. \tag{13}$$

Then we can choose our resistor, which in this case I choose to be:

$$R_5 = \boxed{1.6\ \text{k}\Omega}, \tag{14}$$

$$R_6 = \boxed{16\ \text{k}\Omega}. \tag{15}$$

### 3.3.2 Question 2: Choose $C_2$

The low-pass pole formed by $R_6$ and $C_2$ should be above 16 kHz:

$$\omega_{c,\text{LP}} = \frac{1}{R_6C_2} \geq 10^5 \text{ rad/s}.$$

With $R_6 = 16\,\text{k}\Omega$ this gives $C_2 \leq 0.622\,\text{nF}$. The closest available capacitor was 620 pF, yielding a calculated cutoff near 99.5 kHz.

### 3.3.3 Question 3: Verification

Applying a 100 mV sinusoid at 1 kHz produced the expected tenfold gain without saturation when the LED-photodiode spacing was swept between 3 cm and 25 cm. The high-pass chain maintained a detectable waveform across the entire travel range.

## 3.4 Exercise 4: High-Pass Filter, Rectifier, and Low-Pass Filter

### 3.4.1 Question 1: Duplicate High-Pass

This stage reused the Exercise 2 sizing so $R_9 = 1.6\,\text{k}\Omega$ and $C_3 = 1\,\mu\text{F}$, keeping the 100 Hz cutoff.

### 3.4.2 Question 2: Rectifier Gain

The feedback network $R_6$–$C_2$ implements a first–order low-pass term of the form

$$H_{\mathrm{LP}}(j\omega) = \frac{1}{1 + j\omega R_6 C_2}.$$

The magnitude is

$$|H_{\mathrm{LP}}(j\omega)| = \left| \frac{1}{1 + j\omega R_6 C_2} \right| = \frac{1}{\sqrt{1 + (\omega R_6 C_2)^2}}.$$

The $-3\,\mathrm{dB}$ cut-off occurs when $|H_{\mathrm{LP}}(j\omega_c)| = 1/\sqrt{2}$:

$$\begin{aligned} \frac{1}{\sqrt{1 + (\omega_c R_6 C_2)^2}} &= \frac{1}{\sqrt{2}} \\ 1 + (\omega_c R_6 C_2)^2 &= 2 \\ (\omega_c R_6 C_2)^2 &= 1 \\ \omega_c R_6 C_2 &= 1 \\ C_2 &= \frac{1}{\omega_c R_6}. \end{aligned}$$

With $R_6 = 16\,\mathrm{k\Omega}$ and the desired cut-off $f_c = 16\,\mathrm{kHz}$ (so $\omega_c = 2\pi f_c \approx 1.01 \times 10^5$ rad/s),

$$\begin{aligned} C_2 &= \frac{1}{(2\pi f_c) R_6} \\ &= \frac{1}{(2\pi)(16 \times 10^3)(16 \times 10^3)} \\ &\approx 6.22 \times 10^{-10}\ \mathrm{F} = 0.622\ \mathrm{nF} \approx 620\ \mathrm{pF}. \end{aligned}$$

Thus the ideal design value is

$$\boxed{C_2 \approx 0.622\,\mathrm{nF} \approx 620\,\mathrm{pF}}.$$

In the lab, the largest available capacitor not exceeding this value was $C_2 = 100\,\mathrm{pF}$. Using this component shifts the pole to a higher frequency:

$$\begin{aligned} \omega_{c,\mathrm{actual}} &= \frac{1}{R_6 C_2} = \frac{1}{(16 \times 10^3)(100 \times 10^{-12})} = 6.25 \times 10^5\ \mathrm{rad/s}, \\ f_{c,\mathrm{actual}} &= \frac{\omega_{c,\mathrm{actual}}}{2\pi} \approx 9.95 \times 10^4\ \mathrm{Hz} \approx 100\,\mathrm{kHz}. \end{aligned}$$

So the implemented filter still satisfies the specification ($\omega_c \geq 10^5$ rad/s) while providing even stronger attenuation of higher-frequency interference:

$$\boxed{C_2 = 100\,\mathrm{pF}, \quad f_{c,\mathrm{actual}} \approx 100\,\mathrm{kHz}.}$$

so what is this circuit doing? below are the derivation to see what is happening.

For small AC signals, the bias node $V_4 \approx 2.5$ V is a DC level, so its AC value is effectively zero. Thus, the non-inverting input is at AC ground and the op-amp works as an *inverting* amplifier with:

$$V_2 \xrightarrow{C_1, R_5} \text{inverting node}, \qquad V_5 \xrightarrow{R_6 \| C_2} \text{feedback}.$$

Define

$$Z_1 = R_5 + \frac{1}{sC_1} \qquad \text{(series input impedance)},$$

$$Z_2 = R_6 \parallel \frac{1}{sC_2} = \frac{R_6}{1 + sR_6C_2} \qquad \text{(feedback impedance)}.$$

With the inverting-node voltage $\approx 0$ (ideal op-amp), Kirchhoff's current law gives

$$\frac{V_2(s) - 0}{Z_1} + \frac{V_5(s) - 0}{Z_2} = 0,$$

$$\Rightarrow \quad \boxed{\frac{V_5(s)}{V_2(s)} = -\frac{Z_2}{Z_1}}.$$

Substitute $Z_1$ and $Z_2$:

$$\frac{V_5(s)}{V_2(s)} = -\frac{\dfrac{R_6}{1 + sR_6C_2}}{R_5 + \dfrac{1}{sC_1}}$$

$$= -\frac{R_6}{1 + sR_6C_2} \cdot \frac{sC_1}{1 + sR_5C_1}$$

$$= \boxed{\frac{R_6}{R_5}\left(\frac{sR_5C_1}{1 + sR_5C_1}\right)\left(\frac{1}{1 + sR_6C_2}\right)}$$

This shows:

- $C_1$ and $R_5$ form a **high-pass** with cutoff $\omega_{\text{HP}} = \dfrac{1}{R_5C_1}$.
- $R_6$ and $C_2$ form a **low-pass** with cutoff $\omega_{\text{LP}} = \dfrac{1}{R_6C_2}$.
- In between these two cutoffs, the gain tends to the constant $\dfrac{R_6}{R_5}$ (inverting amplifier).

### 3.4.3 Question 3: Low-Pass Filter

Configure your waveform generator to 100 mV amplitude 1 kHz sine wave using the AD2 signal generator and connect it to V2. Then use your oscilloscope to measure the output at V5. You should see a sine wave at V5. Measure the amplitude of the output sine wave at V5. Vary the frequency to see it's effect on the output amplitude. Here are my result for 1khz, 100khz, and 1mhz:

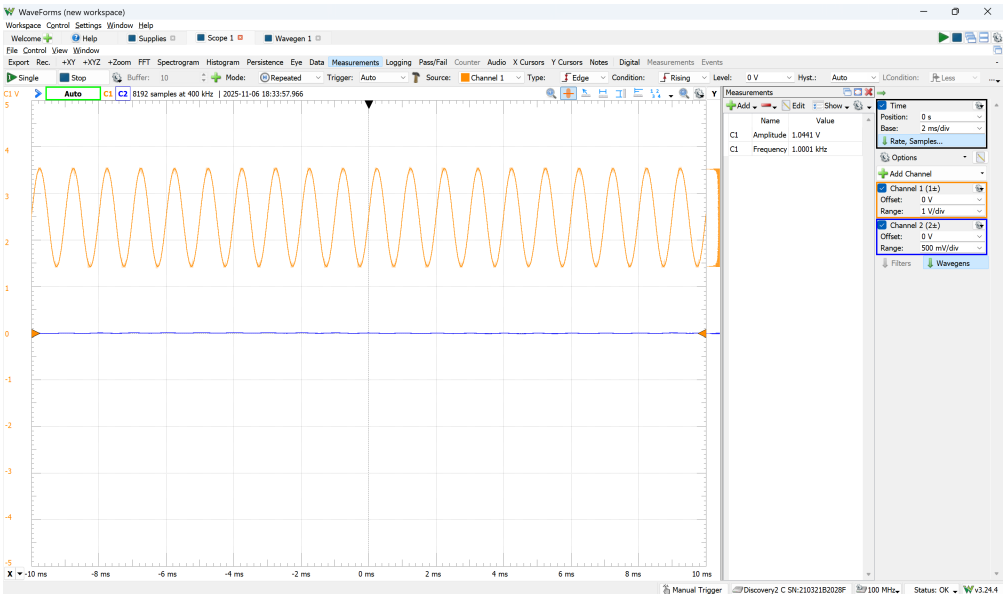


Figure 6: Waveform at $V_5$ with 1 kHz input.

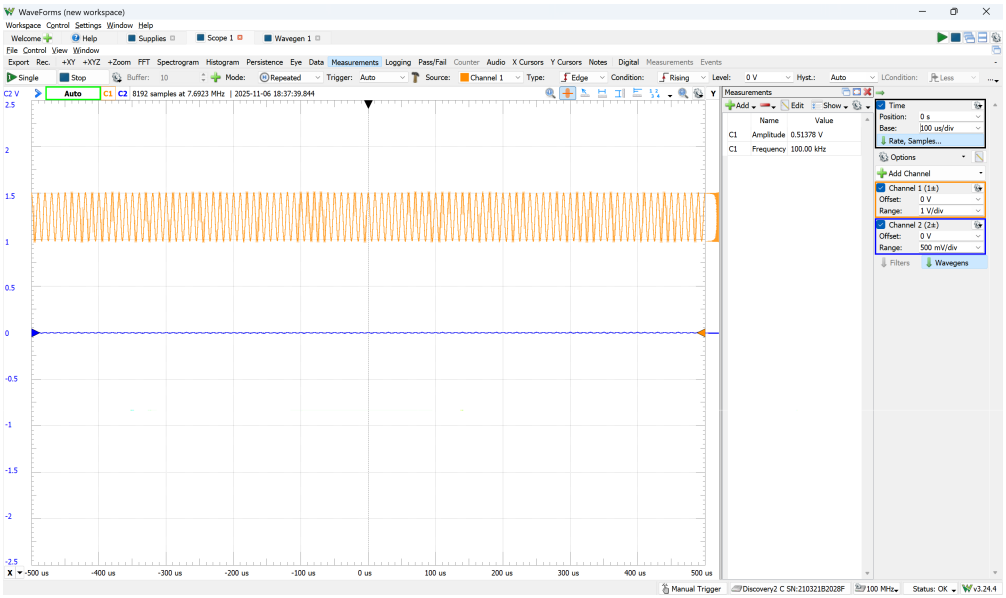


Figure 7: Waveform at $V_5$ with 100 kHz input.

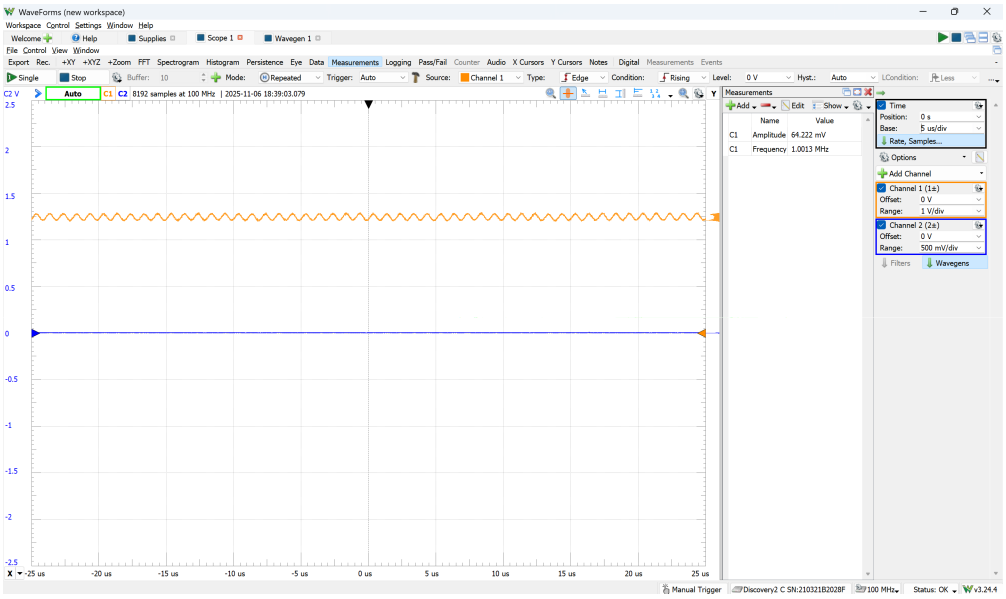


Figure 8: Waveform at $V_5$ with 1 MHz input.

## Frequency response and output amplitudes

Let the input at $V_2$ be a sine wave

$$v_2(t) = \hat{V}_2 \sin(\omega t), \qquad \hat{V}_2 = 0.1 \text{ V } (100 \text{ mV}).$$

The complex gain at frequency $\omega$ is

$$A(j\omega) = \frac{V_5(j\omega)}{V_2(j\omega)} = \frac{R_6}{R_5}\left(\frac{sR_5C_1}{1+sR_5C_1}\right)\left(\frac{1}{1+sR_6C_2}\right).$$

The output amplitude is

$$\hat{V}_5 = |A(j\omega)|\,\hat{V}_2.$$

If we go through the calculations, these are the amplitude modulations at different frequencies for our circuit (assuming R6/R5 = 10):

$$\hat{V}_5 \approx \begin{cases} 9.99999 \text{ V}, & f = 1 \text{ kHz}, \\ 5.3 \text{ V}, & f = 100 \text{ kHz}, \\ 0.62 \text{ V}, & f = 1 \text{ MHz}. \end{cases}$$

Our result are similar to our calculation (1.044 V, 0.514 V, 64.2 mV) are of the same order, so everything seems to check out. I believe this is reasonable given tolerances in the actual resistor/capacitor values, and can also be caused by the bandwidth and slew-rate limits of the real op-amp.

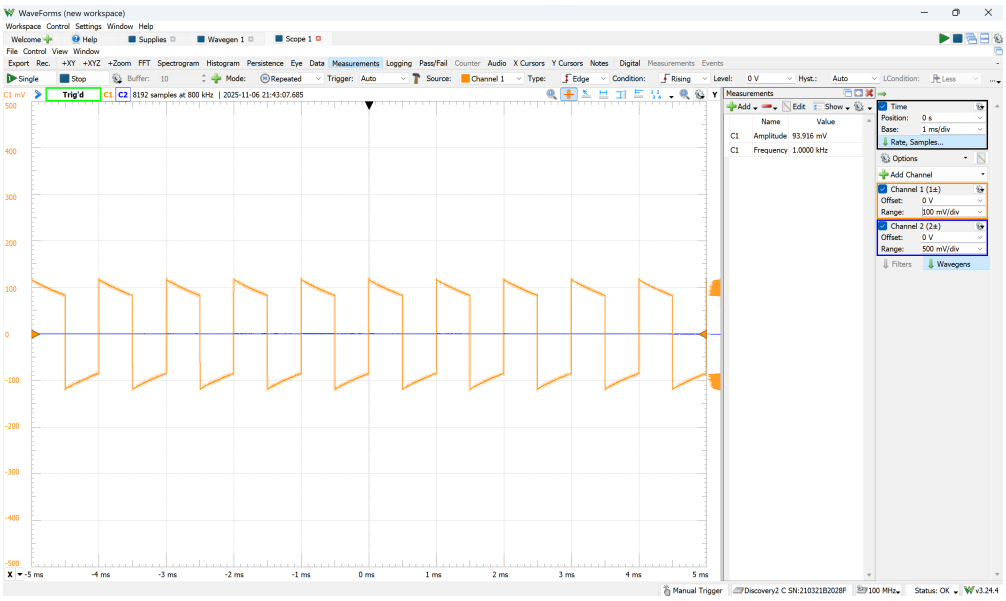

Figure 9: Waveform at $V_6$.

### 3.4.4 Question 4: Response Checks

The next step is to check if your circuit works for Exericse 3 number 4, 5, 6. This is the perfect time for you to test if your circuit has the full 2.5V range. If the range is bad, please first try aligning your photodiode, because that might be the number one cause of this issue. Below are my results for V6, V7, and V8:

Notice that V8 is a constant line, which is great for our data acquisition system. This is the outline on the output of V8: The last stage is a simple RC low–pass with input $V_7$ and output $V_8$: a resistor $R_{12}$ in series, a capacitor $C_4$ to ground at the output node. Its transfer function is

$$H(s) = \frac{V_8(s)}{V_7(s)} = \frac{1}{1 + sR_{12}C_4},$$

so in the frequency domain

$$H(j\omega) = \frac{1}{1 + j\omega R_{12}C_4},$$
$$\left|H(j\omega)\right| = \frac{1}{\sqrt{1 + (\omega R_{12}C_4)^2}}.$$

The cut–off frequency is

$$\omega_c = \frac{1}{R_{12}C_4} = 10 \text{ rad/s} \quad \Rightarrow \quad f_c = \frac{\omega_c}{2\pi} \approx 1.6 \text{ Hz},$$

so we can write

$$\left|H(j\omega)\right| = \frac{1}{\sqrt{1 + (\omega/\omega_c)^2}} = \boxed{\frac{1}{\sqrt{1 + (\omega/10)^2}}}.$$

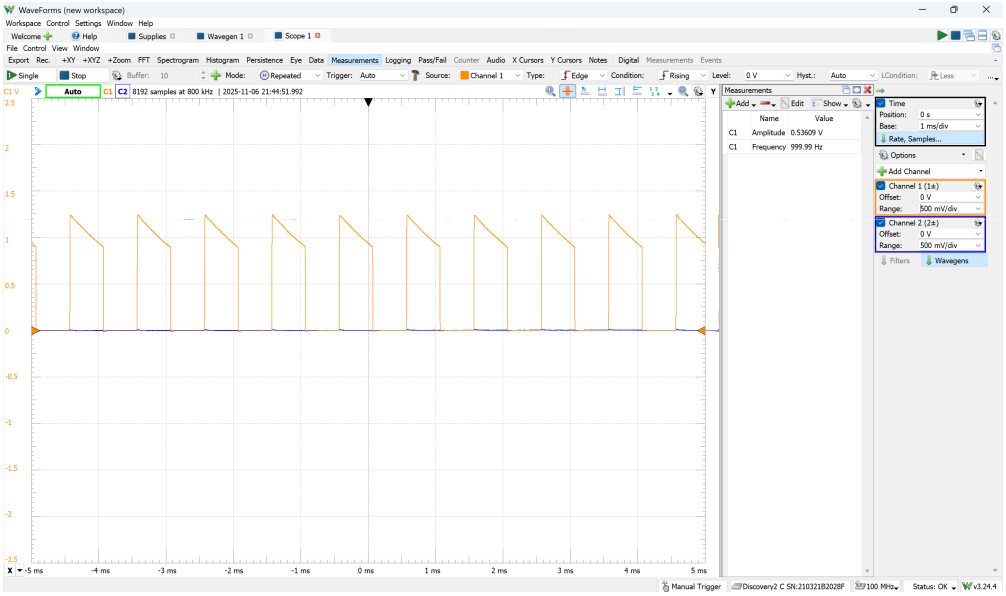


Figure 10: Waveform at $V_7$.

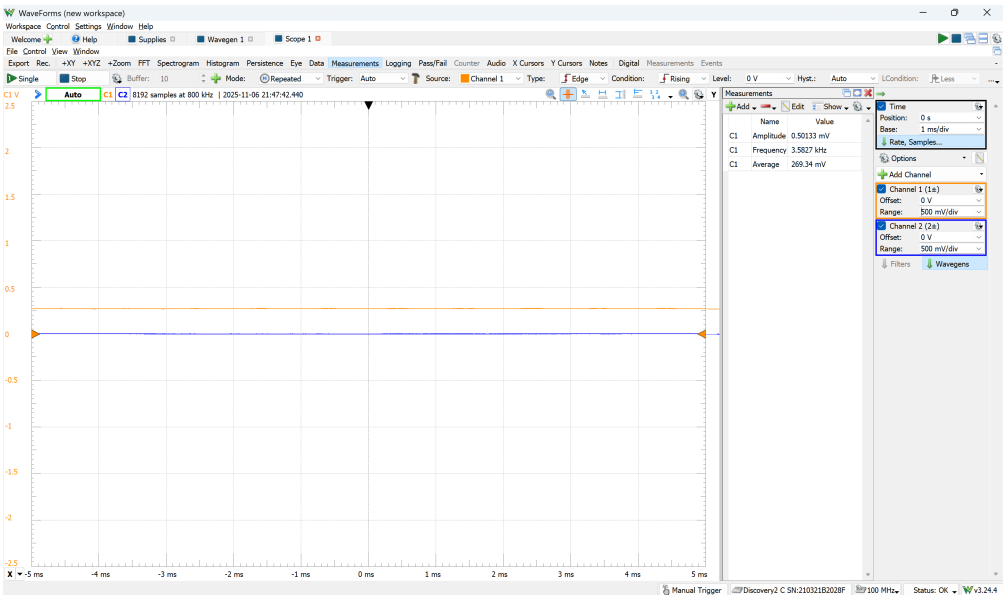


Figure 11: Waveform at $V_8$.

## Effect on the rectified 1 kHz waveform

Let the rectifier output $V_7(t)$ be decomposed into a DC component plus an AC component with fundamental frequency $f_0 = 1$ kHz:

$$V_7(t) = V_{\mathrm{DC}} + v_{\mathrm{AC}}(t),$$
$$v_{\mathrm{AC}}(t) = \sum_{n=1}^{\infty} A_n \cos\bigl(n\omega_0 t + \phi_n\bigr), \qquad \omega_0 = 2\pi f_0.$$

(For a rectified square wave this is a Fourier series with harmonics of 1 kHz.)

The DC term is at $\omega = 0$, so

$$\bigl|H(j0)\bigr| = 1,$$

i.e. the DC level passes unchanged.

For the AC harmonics, each component at frequency $n\omega_0$ is multiplied by $\bigl|H(jn\omega_0)\bigr|$:

$$v_{\mathrm{AC,out}}(t) = \sum_{n=1}^{\infty} A_n \bigl|H(jn\omega_0)\bigr| \cos\bigl(n\omega_0 t + \phi_n + \angle H(jn\omega_0)\bigr).$$

Now compute the attenuation for the fundamental at 1 kHz:

$$\omega_0 = 2\pi \times 10^3 \approx 6.283 \times 10^3 \text{ rad/s},$$
$$\frac{\omega_0}{\omega_c} = \frac{6.283 \times 10^3}{10} \approx 628.3,$$
$$\bigl|H(j\omega_0)\bigr| = \frac{1}{\sqrt{1 + (628.3)^2}} \approx \boxed{1.6 \times 10^{-3}}.$$

So the 1 kHz ripple at the output is only about 0.16% of the ripple at $V_7$. Higher harmonics ($n \geq 2$) are at even larger frequencies $n\omega_0$, so

$$\bigl|H(jn\omega_0)\bigr| = \frac{1}{\sqrt{1 + (n\omega_0/\omega_c)^2}} \ll 1.6 \times 10^{-3} \qquad (n \geq 2),$$

and are suppressed even more strongly.

Therefore the output is

$$V_8(t) = V_{\mathrm{DC}} \cdot 1 \; + \; v_{\mathrm{AC,out}}(t)$$
$$\approx V_{\mathrm{DC}} \; + \; (\text{very small ripple}).$$

On the oscilloscope, this tiny residual ripple is far below the vertical scale and you see essentially only the DC term:

$$\boxed{V_8(t) \approx V_{\mathrm{DC}} = \text{constant line.}}$$

Intuitively, the time constant of the low–pass filter is $\tau = R_{12}C_4 = 1/\omega_c = 0.1$ s, whereas the period of the 1 kHz waveform is $T = 1$ ms $\ll \tau$. The capacitor voltage cannot change much within one period of the waveform, so the filter effectively averages the rectified signal and outputs an almost constant DC value.

## 3.5 Exercise 5: Assemble Complete Circuit

### 3.5.1 Question 1: Integrate Stages

Finally, we are done with our circuit, the next step is to wire together the transimpedance amplifier (Exercise 2), AC amplifier (Exercise 3), and rectifier/low-pass chain (Exercise 4) to form the complete sensor. After this check all the connections again to make sure everything is correct. Once you are sure everything is correct, power up the circuit and test if it works as expected.

### 3.5.2 Question 2: Constrain Output Range

If everything is correct, you should be able to see the output voltage $V_{\text{out}}$ vary between 0 V and 2.5 V as you sweep the LED along the rail. It's okay if you don't get the full range, as long as it's close to 2.5V that's good enough.

# 4 Phase 3: Data Acquisition and Calibration

## 4.1 Exercise 6: Firmware and C# Acquisition

### 4.1.1 Question 1: MSP430 Firmware

The firmware on the MSP430FR5739 was written to sample the low-pass output with the on-chip 10-bit ADC referenced to the 3.3 V rail. Each conversion was formatted into a three-byte packet `[255, MS5B, LS5B]` where the most- and least-significant five bits occupy separate bytes to simplify parsing on the PC.

# 5 The Importance of Alignment with Photodiode and LED

I was struggling to get the full range of 2.5V output for a while, but thanks to our TA, he informed me that the alignment between the photodiode and LED is very crucial. To convince you, here is a simple math behind it:

### 5.0.1 Question 2: C# Application

The companion C# program:

a) opened the appropriate serial port and maintained continuous communication,

b) recombined the MSBs and LSBs into a single 10-bit code,

c) displayed and plotted the live data stream while logging to disk, and

d) implemented a basic UI showing instantaneous voltage and providing controls for calibration capture.

## 5.1 Exercise 7: Calibration and Resolution

### 5.1.1 Question 1: Distance Sweep

Measurements were recorded at a minimum of five LED-photodiode separations across the full mechanical travel. Each point stored both the ADC code and the physical distance measured with a ruler on the extrusion.

### 5.1.2 Question 2: Curve Fit

The voltage-to-distance data were fitted with a smooth function (e.g., inverse power). Plotting both the raw scatter and the fitted curve in the report allowed visual confirmation that the model captured the sensor response with minimal residual error.

### 5.1.3 Question 3: Conversion to Position

The inverse of the fitted function was implemented in software so that each ADC code could be converted in real time to an estimated separation distance.

### 5.1.4 Noise Characterization

With the slider set near mid-range, the converted position was logged for roughly 10 s and its standard deviation was interpreted as RMS noise. The experiment was repeated near both extremes of travel, and differences in noise level were attributed to the varying sensitivity (slope of the calibration curve) at those points.

# 6 Appendix

## 6.1 Explanation for Firmware

## 6.2 MSP430 Firmware for Distance Sensor Interface

The goal of this firmware is to periodically sample the analog output of a distance sensor using the `ADC10_B` on the MSP430FR5739, quantize it to a 10-bit value in the range $[0, 1023]$ corresponding to $[0, 3.3]$V, and transmit the result to a host PC over UART as a 3-byte frame consisting of: a start byte with value 255, a second byte holding the most significant five bits (MS5B) of the ADC code, and a third byte holding the least significant five bits (LS5B). This packing format is what the C# program expects when reconstructing the original 10-bit value. A TimerA interrupt every 10 ms defines the sampling period; all measurement and communication work is done inside the timer ISR, while the main loop simply initializes hardware and then idles.

### Main Program Flow

The main function stops the watchdog, initializes all peripherals, enables global interrupts, and then leaves the CPU in an idle loop. All periodic sampling is driven by the TimerA interrupt.

Listing 1: Main firmware entry point

```c
#include <msp430.h>
#include <stdint.h>

#define START_BYTE 0xFF
#define TICK_10MS  10000u      // 10 ms @ SMCLK = 1 MHz

static void gpio_init(void);
static void uart_init_9600_smclk1M(void);
static void adc_init_A2_AVCC(void);
static void timer_init_10ms(void);
static inline void uart_tx(uint8_t b);
static uint16_t adc_sample_blocking(void);

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;           // stop watchdog

    gpio_init();
    uart_init_9600_smclk1M();
    adc_init_A2_AVCC();
    timer_init_10ms();

    PM5CTL0 &= ~LOCKLPM5;               // unlock I/Os on FRAM parts
    __bis_SR_register(GIE);             // global IRQs on

    while (1) { __no_operation(); }     // all work done in timer ISR
}
```

The watchdog timer is disabled to prevent unwanted resets, and the helper functions `gpio_init()`, `uart_init_9600_smclk1M()`, `adc_init_A2_AVCC()`, and `timer_init_10ms()` configure the GPIO, UART, ADC, and TimerA respectively. The line `PM5CTL0 &= ~LOCKLPM5;` unlocks the FRAM device I/O pins, and `__bis_SR_register(GIE);` enables global interrupts, allowing the TimerA ISR to execute every 10 ms. The `while(1)` loop does nothing; it simply keeps the CPU running while the interrupt-based acquisition proceeds in the background.

### GPIO Configuration

The GPIO configuration routes `P1.2` to the analog input channel `A2` for the distance sensor and maps `P2.0`/`P2.1` to the UART TX/RX pins for communication with the PC.

Listing 2: GPIO setup for ADC input and UART pins

```c
static void gpio_init(void)
{
    // P1.2 as analog input (A2)
    P1DIR  &= ~BIT2;
    P1SEL0 |=  BIT2;
```

```
    P1SEL1 |=  BIT2;                    // selects analog function on
        FR57xx
    P1REN  &= ~BIT2;                    // disable pull resistor

    // P2.0=TXD, P2.1=RXD for eUSCI_A0
    P2SEL1 |= (BIT0 | BIT1);
    P2SEL0 &= ~(BIT0 | BIT1);
}
```

On P1.2, the direction bit is cleared to make it an input, and the special function select bits P1SEL0/P1SEL1 are set so that the pin is connected to the internal ADC input channel A2. Pull-up and pull-down resistors are disabled to avoid disturbing the analog signal. Pins P2.0 and P2.1 are switched from general-purpose I/O to the UART peripheral (eUSCI_A0) for serial transmission and reception.

### UART Configuration and Transmission

The UART is configured to run at 9600 baud using SMCLK at approximately 1 MHz. The code uses oversampling mode for better baud rate accuracy. A small helper function uart_tx() blocks until the transmit buffer is ready and then writes a single byte, guaranteeing that the 3-byte frame is transmitted in order.

Listing 3: UART configuration at 9600 baud

```
static void uart_init_9600_smclk1M(void)
{
    UCA0CTLW0 = UCSWRST | UCSSEL__SMCLK;      // hold reset, select
       SMCLK
    UCA0BRW   = 6;                            // 1 MHz / (16*9600)
           6.51
    UCA0MCTLW = 0x2081;                       // UCOS16 + BRF=8 +
        B R S 0x20
    UCA0CTLW0 &= ~UCSWRST;                    // release reset,
       enable UART
}

static inline void uart_tx(uint8_t b)
{
    while (!(UCA0IFG & UCTXIFG));             // wait until TX buffer
        empty
    UCA0TXBUF = b;                            // send one byte
}
```

In summary, this block ensures that the microcontroller can reliably send each sample as a sequence of bytes over the serial port to the C# application.

### ADC Configuration and Sampling

The ADC is configured to perform single-channel, single-conversion measurements on input channel `A2` with reference voltage `AVCC` (3.3 V). The conversion result is 10 bits, which directly matches the exercise requirement.

Listing 4: ADC setup on channel A2, 10-bit resolution

```
static void adc_init_A2_AVCC(void)
{
    ADC10CTL0 &= ~ADC10ENC;                    // allow configuration
    ADC10CTL0  = ADC10SHT_2 | ADC10ON;         // 16-cycle S/H, ADC on
    ADC10CTL1  = ADC10SHP;                     // use sampling timer
    ADC10CTL2  = ADC10RES;                     // 10-bit result
    ADC10MCTL0 = ADC10SREF_0 | ADC10INCH_2;    // AVCC/AVSS, channel
       A2 (P1.2)
    ADC10CTL0 |= ADC10ENC;                     // enable conversions
}
```

Listing 5: Blocking ADC sampling routine

```
static uint16_t adc_sample_blocking(void)
{
    ADC10CTL0 |= ADC10SC;                      // start conversion
    while (ADC10CTL1 & ADC10BUSY) ;            // wait until
       conversion complete
    return ADC10MEM0 & 0x03FF;                 // 10-bit value
       (0..1023)
}
```

The function `adc_sample_blocking()` triggers a conversion by setting `ADC10SC`, then busy-waits until `ADC10BUSY` clears. Finally, it masks the 10-bit result with `0x03FF` to ensure that only the lower ten bits are returned, implementing exactly the $[0, 1023]$ range required.

### Timer Configuration and 10 ms Period

To obtain a sampling period of approximately 10 ms, TimerA0 is configured in up mode using `SMCLK` at 1 MHz. The value `TICK_10MS` is chosen such that the timer reaches the compare value every 10 ms.

Listing 6: TimerA configuration for 10 ms period

```
static void timer_init_10ms(void)
{
    TA0CCR0  = TICK_10MS - 1;                  // period: 10000 ticks
    TA0CCTL0 = CCIE;                           // enable CCR0
       interrupt
    TA0CTL   = TASSEL__SMCLK | MC__UP | TACLR;  // SMCLK, up mode,
       clear
}
```

Every time TimerA0 reaches `TA0CCR0`, the `TIMER0_A0_VECTOR` interrupt is triggered. This interrupt defines the acquisition rate of approximately 100 Hz.

### Core Logic: Sampling, Packing, and Transmitting

The most important code block is the TimerA ISR. Every 10 ms, it performs an ADC conversion, splits the resulting 10-bit value into two 5-bit fields (MS5B and LS5B), and sends them to the PC, preceded by a fixed start byte `0xFF`.

Listing 7: TimerA ISR: sample ADC and transmit 3-byte frame

```
/* ---- 10 ms ISR: sample, split to 5+5 bits, send 3 bytes ---- */
#pragma vector = TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR(void)
{
    uint16_t adc = adc_sample_blocking();      // 0..1023 for 0..3.3V
        (AVCC)
    uint8_t  ms5 = (adc >> 5) & 0x1F;          // bits [9:5]  (MS5B)
    uint8_t  ls5 =  adc       & 0x1F;          // bits [4:0]  (LS5B)

    uart_tx(START_BYTE);                       // Out byte 1: start
       marker (255)
    uart_tx(ms5);                              // Out byte 2: MS5B
    uart_tx(ls5);                              // Out byte 3: LS5B
}
```

The key operations inside the ISR are the bit manipulations `ms5 = (adc >> 5) & 0x1F;` and `ls5 = adc & 0x1F;`. The first line shifts the 10-bit ADC value $D$ right by five positions and masks with `0x1F` to keep only bits $b_9$ through $b_5$ (the most significant five bits). The second line masks the original value with `0x1F` to keep bits $b_4$ through $b_0$ (the least significant five bits). On the PC side, the C# application reconstructs the original 10-bit value using the inverse operation $D = (D_{\text{MS}} \ll 5) + D_{\text{LS}}$.

Together, the TimerA ISR and the UART transmission realize the required data format: out byte 1 is the fixed start marker (255), out byte 2 encodes MS5B, and out byte 3 encodes LS5B. This 3-byte frame is sent every 10 ms, providing a continuous stream of distance measurements for the C# application to decode, display, graph, and store.

## 6.3 Explanation for C#

The C# application was updated to display both raw ADC values and converted position simultaneously. Additional logic signaled when the sensor moved outside the characterized range, satisfying the lab requirement for an out-of-range indicator. The most important components of this program are:

- a serial interface that reconstructs 10-bit ADC codes from the MSP430 data stream,
- a calibration module that fits a curve and converts each ADC code to physical distance, and

- a UI/update loop that shows distance, flags out-of-range conditions, and logs data for the noise analysis.

**Serial data acquisition and ADC reconstruction** On the firmware side, the MSP430 sends a 3-byte packet composed of a start byte and two 5-bit fields containing the most and least significant bits of the 10-bit ADC result. On the PC side, the `SerialPortService` class implements a small state machine that watches the incoming serial stream, waits for the start byte `0xFF`, and then assembles the full ADC code. This behaviour is captured in **??** 8.

Listing 8: Reassembling 10-bit ADC samples from 3-byte serial packets in `SerialPortService`.

```
private void ProcessByte(byte data)
{
    if (_bufferIndex == 0)
    {
        // Look for start byte 0xFF
        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;

        int ms5b = _buffer[1] & 0x1F;    // bits 9..5
        int ls5b = _buffer[2] & 0x1F;    // bits 4..0
        int adcValue = (ms5b << 5) | ls5b;

        AdcDataReceived?.Invoke(
            this,
            new AdcDataReceivedEventArgs(adcValue));

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

?? 8 shows that every time a complete 3-byte packet is received, the code masks the two 5-bit fields, shifts the most significant portion, and ORs them together into a 10-bit integer. The resulting `adcValue` is wrapped in an `AdcDataReceivedEventArgs` object and broadcast via the `AdcDataReceived` event so that the rest of the application can treat ADC samples as a clean, time-stamped stream.

**Calibration, ADC-to-distance conversion, and range checking** The calibration step from Exercise 7 is implemented in the `CalibrationService` and `CalibrationData` classes. `CalibrationService` takes the user-entered distance–ADC pairs, calls the MathNet.Numerics fitting routines (linear, polynomial, power-law, or inverse), and stores the resulting coefficients and $R^2$ value. Once a calibration is available, the `CalibrationData` class applies the fitted function to convert every ADC sample into an estimated distance and also decides whether a reading lies inside the valid region of the calibration. The core logic is illustrated in ?? 9.

Listing 9: Using the fitted calibration curve to convert ADC codes to distance and check the valid range.

```
public double ConvertAdcToDistance(int adcValue)
{
    if (Coefficients == null || Coefficients.Length == 0)
        return 0;

    double x = adcValue;

    switch (FitType)
    {
        case FitType.Linear:
            // y = m x + b
            return Coefficients[0] * x + Coefficients[1];

        case FitType.Polynomial2:
            // y = a x^2 + b x + c
            return Coefficients[0] * x * x
                 + Coefficients[1] * x
                 + Coefficients[2];

        case FitType.Power:
            // y = a x^b
            return (x > 0)
                ? Coefficients[0] * Math.Pow(x, Coefficients[1])
                : 0;

        // Other fit types (polynomial3, inverse)
        // are implemented analogously.
    }

    return 0;
```

```
}

public bool IsInRange(int adcValue)
{
    return adcValue >= MinAdcValue &&
           adcValue <= MaxAdcValue;
}
```

In **??** 9 the chosen model (for example a quadratic polynomial) and its coefficients are determined once during the calibration step. During normal operation the function `ConvertAdcToDistance` is then evaluated in real time for each incoming `adcValue` to produce a physical distance in centimetres, while `IsInRange` uses the user-selected minimum and maximum ADC thresholds to flag when the sensor is being operated outside the calibrated region.

**Real-time display, logging, and out-of-range indicator** The `MainForm` class subscribes to the `AdcDataReceived` event and uses the calibration object to keep the UI and data logger in sync with the incoming samples. The event handler shown in **??** 10 converts the ADC code, updates the numerical readouts, sets the in-range or out-of-range label, appends the data to the plotting buffers, and optionally logs the sample to disk and to the noise-analysis buffer.

Listing 10: Event handler that updates the UI, plots, and logging structures for each new sample.

```
private void SerialPort_AdcDataReceived(
    object? sender, AdcDataReceivedEventArgs e)
{
    int adcValue = e.AdcValue;
    double distance = 0;
    bool isInRange = true;

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

    // Update numeric readouts
    lblAdcValue.Text =
        $"ADC Value: {adcValue} / 1023";
    lblDistance.Text =
        $"Distance: {distance:F2} cm";

    // Indicate whether the reading is inside
    // the calibrated operating range
```

```
25      if (isInRange)
26      {
27          lblRangeStatus.Text = "In Range";
28          lblRangeStatus.ForeColor = Color.Green;
29      }
30      else
31      {
32          lblRangeStatus.Text = "Out of Range";
33          lblRangeStatus.ForeColor = Color.Red;
34      }
35
36      // Append to time-series buffers for plotting
37      double t = (DateTime.Now - _startTime).TotalSeconds;
38      _adcTimeData.Add(t);
39      _adcValueData.Add(adcValue);
40      _distanceTimeData.Add(t);
41      _distanceValueData.Add(distance);
42
43      // Log data if logging is active
44      if (_isLogging)
45      {
46          var reading = new SensorReading(
47              adcValue, distance, isInRange);
48          _dataLogger.AddReading(reading);
49          lblSampleCount.Text =
50              $"Samples: {_dataLogger.Count}";
51      }
52
53      // When noise recording is enabled, the same
54      // distance values are accumulated for later
55      // RMS noise and standard deviation analysis.
56  }
```