ECE 167 Sensing and Sensor Technologies

Lab #2: Encoder, Ultrasonic, and Capacitive Touch

February 05, 2024

Final Check-off: 01/31/2024, 4:36 pm,

Tim, Caden

Commit ID: cd3631dd0481ba336088eee234715c4421b5c33a

Collaborators: Niki Mobtaker

I. Overview

This lab focused on the exploration of a quadrature encoder, an ultrasonic distance sensor, and a capacitive touch sensor. Using the Quadrature Encoder Interface (QEI), I gained insights into angular displacement measurement and color mapping using RGB LEDs. The ultrasonic distance sensor involved not only capturing accurate distance readings but also calibration using Least Squares. These measurements were then used to modulate speaker tones. Lastly exploring capacitive touch sensors, we used different measurement techniques like RC time constants, capacitive bridges, and relaxation oscillators. Each part developed our skills in sensor interfaces, using ISRs, calibration methods, and signal processing.

Components Required:

- Nucleo-64 + IO shield
- Speaker
- Audio amplifier
- Breadboard
- Rotary encoder
- Ultrasonic distance sensor
- Capacitive touch board
- LM555 timer chip
- Capacitors: 22 pF, 0.01μF,
- Resistors: 10 kΩ,

II. Wiring and Diagrams

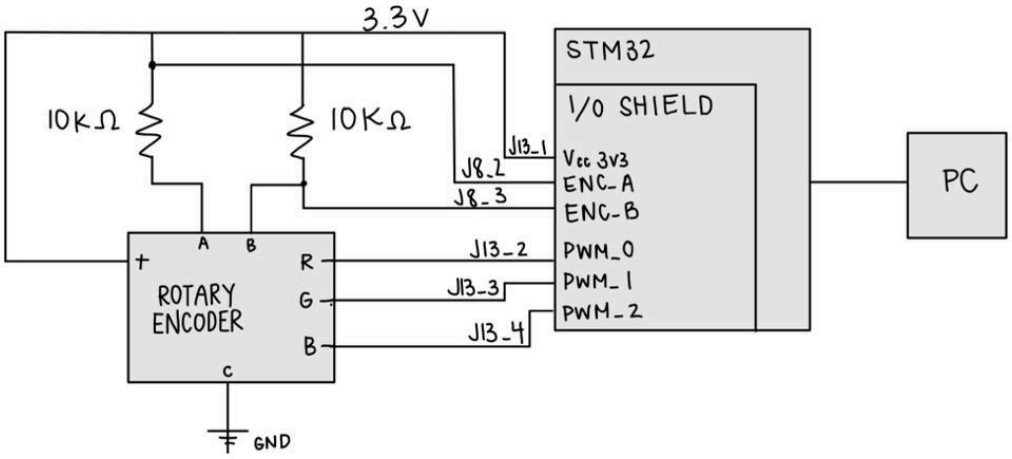


Figure 1. Breadboard diagram for QEI

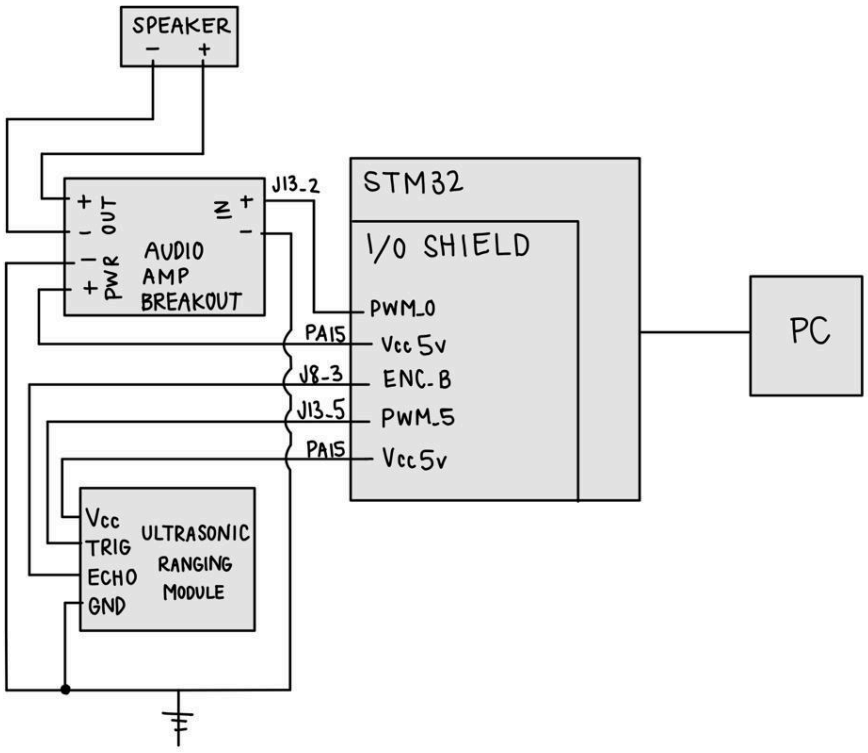


Figure 2. Breadboard diagram for PING

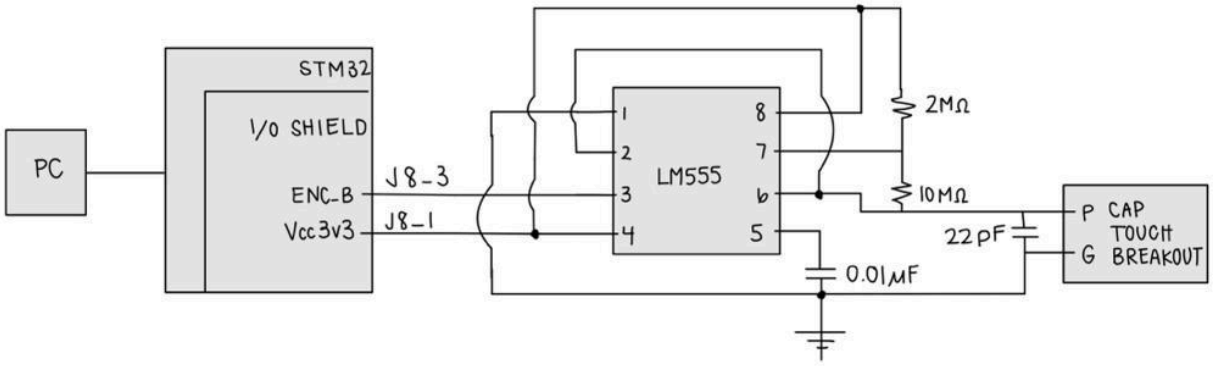


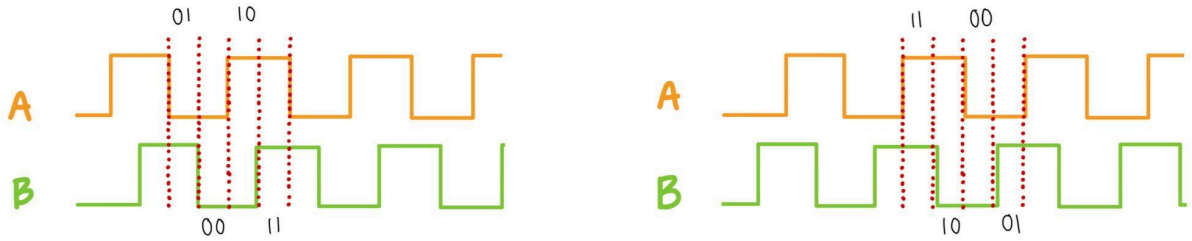
Figure 3. Breadboard diagram for CAPTOUCH

III. Project Design

Part 1: QEI

In this part of the project, I used a rotary encoder to generate different hues of light based on its position and rotation of the. Utilizing quadrature encoding with gray encoding in a state machine, analog readings are converted to rotation angles. By using gray encoding, which has successive values differing in only one bit, I can read the signals A and B and determine which one is leading and the state of the other. The 90˚ phase shift between signals A and B allows us to read the direction of the rotation if the signals are represented by binary bits. I used a state machine with gray encoding to determine rotary encoder direction based on previous and current signal states.

1.1 Read Quadrature Encoder Interface (QEI) and generate angle accumulation



(a) Clockwise (b) Counter-clockwise

Figure 4. Quadrature Encoding Signals A and B Based on Rotation

To generate the state machine first I determined how the signals would behave based on their rotation as shown in Figure 4. Based on the expected behavior I created a state machine that would determine the direction and then increment or decrement the count. The state machine in

Fig. 5 was implemented in `QEI_IRQ()`. The four states are represented by two bits where the first bit represents signal A and the second bit signal B.

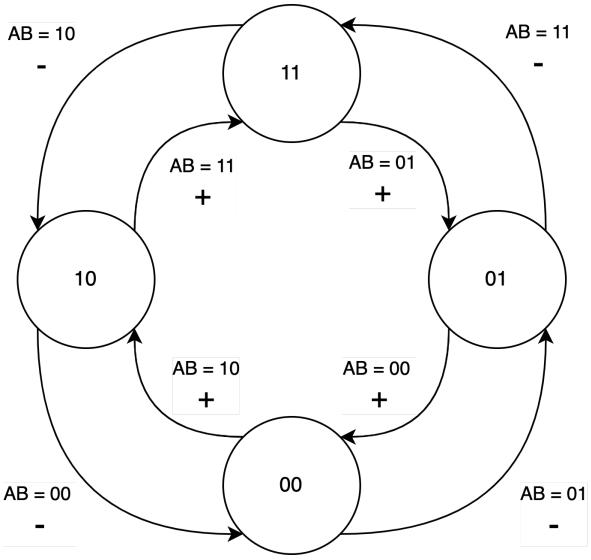


Figure 5. State Machine Implementation for Calculating Direction

Starting at state 11, if the signal for B changed it would move on to state 10 and decrement the count, else if the signal for A changed then it would move on to state 01 and increment the count. For each state the same logic is applied based on which signal changed. This can be visualized in the state machine in Fig. 5.

1.2 QEI Software Implementation

Using two external interrupts for the pins connected to the A and B signal of the quadrature encoder I called the state machine any time there was a change in either signal. This allowed the count to update. This count was stored in a module variable called position.

To make sure that the count returned to zero after its maximum value, I created a function, `circularity()`, that upon detecting a value larger than the maximum or smaller

than the minimum it would set the count to the next value. If at the edge case of 360, it incremented the count to 361, `circularity()`, would reset the value to 0. The same logic would be applied to if the number was below zero, it would reset the count to 360. Initially the raw angle value was returned in `QEI_GetPosition()`.

The function of resetting the position to 0 was taken care of by `QEI_ResetPosition()` so that it would go to 0. After implementation, the circularity function could also be held within the reset position function but its separation was just a matter of coding style.

1.3 Map QEI to color

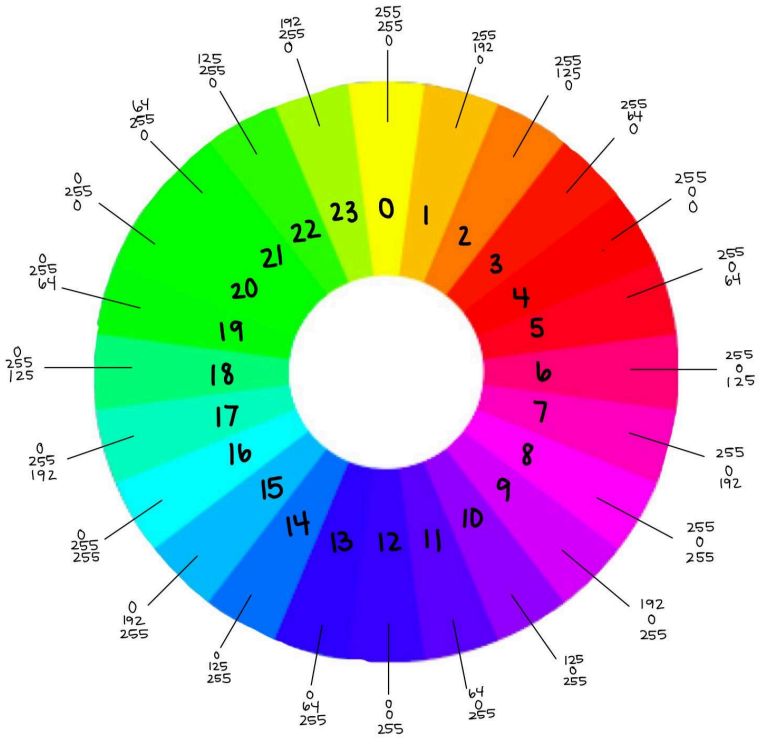


Figure 6. Color Mapping for 24 States State Machine with RGB values

To map the QEI readings to colors, I used another state machine with 24 states. The number of state machines corresponds to the number of pulses the rotary encoder makes in one full cycle. I used the current position values mapped into the 24 to determine what state to go into the color state machine. Once inside the color, the RGB values were set. I initialized the state machine into the first state which made the color yellow. If the position increased, the states would change to reflect the new color. This state machine was called anytime the rotary encoding state machine would detect a change in signals so that the color would update at the same time. The transition between each color was determined by the RGB values on the color wheel and can be seen in Fig. 6, which ensured a smoother transition between each color.

Part 2: Ultrasonic Distance Sensor

The second sensor used in this lab is a time-of-flight ranging sensor using ultrasonic waves. By transmitting a signal into the environment and then measuring the time it takes to reach an object, reflect off it, and return I could determine the distance of an object from the sensor and then use that data to generate a frequency. To determine the time elapsed of the signal, I used two ISRs, one external and one for time. They helped determine what state the echo was in and how long the signal took to travel. Once I obtained a calibrated distance, I would map the distance to set frequencies and generate tones similar to a theremin.

2.1 Capture Ultrasonic Distance Sensor Output

In the time ISR, I implemented a variation of the state machine using two if statements controlled by the variable `switchTime` which is initially set to state 0. The two states are the waiting (0) and trigger state (1). In the first state I set the port pin to high using a function from

the HAL_GPIO library and then using the ARR register to generate an interrupt for 10-1 μs. The state is changed to the trigger state before exiting the if statement.

Once entering the waiting state, the port pin is reset and then I use the ARR register once again to generate an interrupt for 60-1 ms. Just as before, the state is changed to the waiting state so that it will start there in the next iteration of the ISR. Before exiting the IRS the interrupt flag is cleared.

The external ISR is triggered when PB5 (ENC_B) changes state. To determine what type of change occurred I read the state of the pin using the function from the `HAL_GPIO` library that could read the state of PB5. If the value obtained was high, I captured the time in the `startTime` variable, since a high state indicates the echo was sent out. If the state of PB5 returned a low signal then the echo had returned so the time was recorded in endTime was recorded.

To capture the time of flight into `PING_GetTimeofFlight()`, I used a flat surface as the target for the echo to return and returned the difference between the time the echo left and returned to the sensor. These times were captured within the external ISR.

I used the time of flight to calculate the distance in `PING_GetDistance()`. Excluding the negative time of flight values was necessary as they were considered noise in the system and their inclusion would change the final distance value. Instead only when the time of flight was positive, the distance was calculated to be returned. Once I had calculated the distance to the target and time of flight outputs from the serial output, I used that data to develop least squares to estimate the distance to the target.

2.2 Calibrate the Ultrasonic Distance Sensor Using Least Squares

Since the parameters were prone to noise, I used multiple measurements at varying distances to create the least squares. Using matlab software I generated Eq. 1 and used it to set the distance as in Fig. 7.

$$y = 0.0187x + 0.9253 \qquad (1)$$

```
// filter out negative values
  if (flightTime > 0)
  {
      distance = distance = 0.0187 * flightTime + 0.9253;
  }
```

Figure 7. Calibrating distance based on flightTime

The equation was calculated using five measurements at 3 inch intervals so that the multiple measurements would minimize the effect of noise. The variation of time elapsed for every measurement was not very noisy but the occasional negative readings were the only ones needing to be excluded. The measurements were converted into centimeters for the calculation of the equation.

2.3 Tone Out Based on Distance

First I decided on a frequency range of 100-800 Hz which would span a distance of approximately 24 inches. To map the distance into the tones, I converted the current calculated distance into a percentage by dividing it by the maximum distance of 24. I used this percentage to calculate a value within the range of frequencies. The mapping was made using the syntax: `(PING_GetDistance() / 24.0) * 700`. This would yield a value between 0 and 700. Since the minimum frequency for the PWM signal is 100, an offset of 100 was added to the calculated frequency.

Part 3: Capacitive Touch Sensor

For this part of the lab, I explored how to interface to a capacitive touch sensor by measuring the change in capacitance using various methods.

3.1 RC Time Constant Based Measurement

The first method to measure the capacitance was using the RC time constant. The design of the circuit is shown in Fig. 8. To create the low pass filter I tested various resistor values, all greater than 100 kΩ. This was done to observe how the output changed when I touched the capacitor.

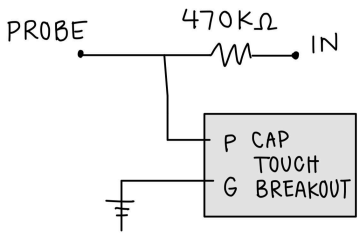
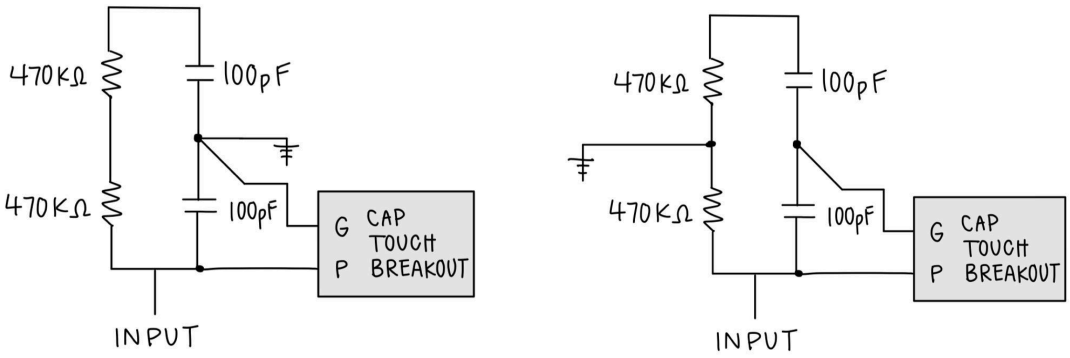


Figure 8. RC Circuit for Capacitance Sensing

3.2 Capacitive Bridge and Differential Amp

In this part I tested two capacitive bridges in parallel with the capacitive touch sensor as is shown in Fig, 9. The two bridges used the same capacitor and resistor values. This design was placed into the breadboard so that I could determine which configuration yielded the best signal on the oscilloscope.



(a) Low-pass (b) High-Pass

Figure 9. Capacitive Bridges for Capacitance Sensing

3.3 Relaxation Oscillator

For this portion of the lab, I used the LM555 in astable mode, the circuit in Fig. 3, shows its implementation. To calculate the resistors to be used I picked a reasonable frequency and then used the relationship in Eq. 2 to determine the best resistor values

$$f = \frac{1}{T} = \frac{1.44}{(R_A + 2R_B)C} \quad (2)$$

When solving this equation for the resistors A and B while using a 22 pF capacitor and a desired frequency of 3kHz we get the relationship between the two resistors to be Eq. 3

$$R_A + 2R_B = 21.8\,M\Omega \quad (3)$$

Based on that calculation, I decided to have R_A to be 2MΩ and R_B to be 10MΩ. The 2MΩ resistor was replaced by two 1MΩ resistors.

3.4 Capacitive Touch Software Interface

Once again using an external ISR to detect a change in the signal from PB5, we can determine the frequency of the signal. In this interrupt I determined the period of the change by taking the difference of the current and last time. Then calculating the average with that reading and the last 5 previous values to determine if there was a change in capacitance. If the change was larger than the threshold obtained from the relaxation oscillator then I could determine that the capacitive touch sensor was touched.

IV. Testing and Results

Part 1: QEI

1.1 Read Quadrature Encoder Interface (QEI) and generate angle accumulation

After testing the functionality of the state machine, the serial output would count infinitely positive for clockwise rotations and infinitely negative for counterclockwise rotations. This was expected as it was outputting the raw data every time it changed. To improve the count and map the values from 0-360, I created a helper function that would take care of the transition when the values reached the end.

```
circularity()
{
   If the position greater than 360:
       Reset position to 0
       // expected: ... 359, 360, 0, 1 ...
   else if the position is less than 0
       Reset position to 360
       // expected ... 1, 0, 360, 359 ...
}
```

Fig. 10. Implementation of circularity

The general pseudocode I implemented is Fig. 10. This was the last version of the circularity function. Other edge comparison values were initially 359 and -1 and other variations of the same but they wouldn't always work. In general I noticed the circularity numbers could be changed to any ranges

1.2 QEI Software Implementation

To make sure that the count would update every time there was an external event detected the state machine was called within the ISR for each pin.

1.3 Map QEI to color

Just like the state machine was called anytime there was an external signal change detected. I assumed the color would be changed anytime the distance changed which was in the position state machine. To implement the color mapping I used a state machine based on Fig. 6 where each state sets the modular r, g, b values. One of the important facts to remember is how the values are translated by the function to be used to set the colors. For a color value of 255, which is the highest, the equivalent duty cycle would be 0. For the minimum color value of 0, the equivalent duty cycle is 100. This was discovered after various testing. Once I determined the correct distance values and made sure they output the corresponding color I could change the RGB to get a better match.

Part 2: Ultrasonic Distance Sensor

2.1 Capture Ultrasonic Distance Sensor Output

Initially, I started with a state machine for the external interrupt but realized that it was necessary that the state immediately change once the time constraints were met. I achieved this by setting two if loops next to each other. To test if it was changing between each “state” I used print statements. These cause a delay when testing the program so it was essential that I removed them before continuing. Once I knew the time was being captured I could measure the distance based on the output time differences and mapped the time to predicted distance. This turned out to be inaccurate especially when noise would affect the sensor and negative elapsed times would appear as seen in Fig. 11.

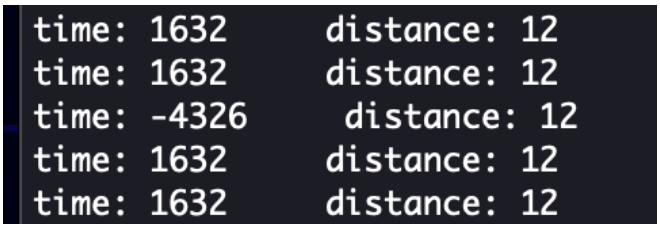


Figure 11. Exclusion of

The solution was excluding any negative time elapsed into the calculation of the distance and instead taking the previous value of the distance.

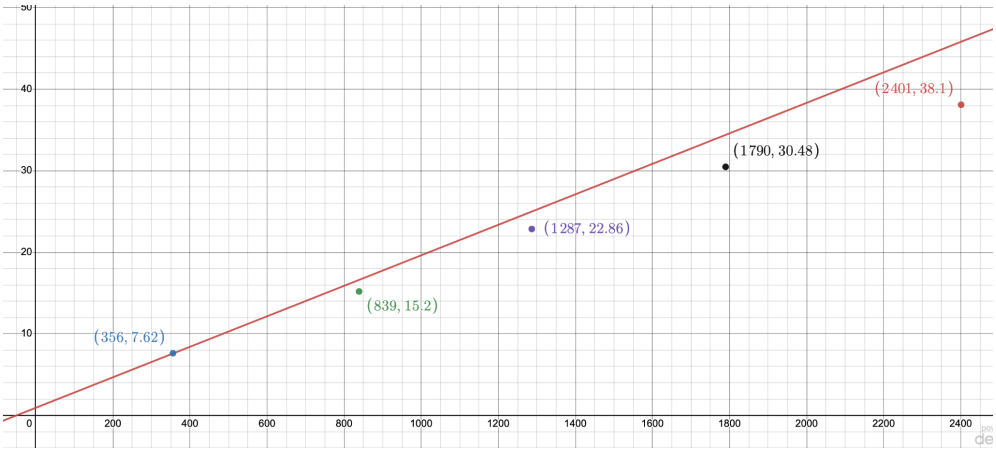
2.2 Calibrate the Ultrasonic Distance Sensor Using Least Squares

By implementing the least squares model from Eq. 1, I was able to obtain a much better result on distance calculations. Initially I had measured distances for the equation in inches but it yielded a much less precise result. Eventually I changed to centimeters by changing the units of

the distance measurements. The new relationship yielded better and more accurate results which only varied as the object got further from the radar. While testing the radar I also noticed that the size of the measurement object will vary the results, objects with smaller surface area will create more noise in the sensor as the distance returns at different intervals.

The recorded measurements compared to the least squares equation shows this variation. As the time elapsed was longer the equation yields a distance that is greater than what the measured distance was. This is observed towards the right of Fig. 12 where the red line represents Eq. 1 and each point is a measurement taken to derive Eq. 1.

Distance (cm)



Time Elapsed (µs)

Figure 12. Time elapsed (µs) vs Distance (cm)

2.3 Tone Out Based on Distance

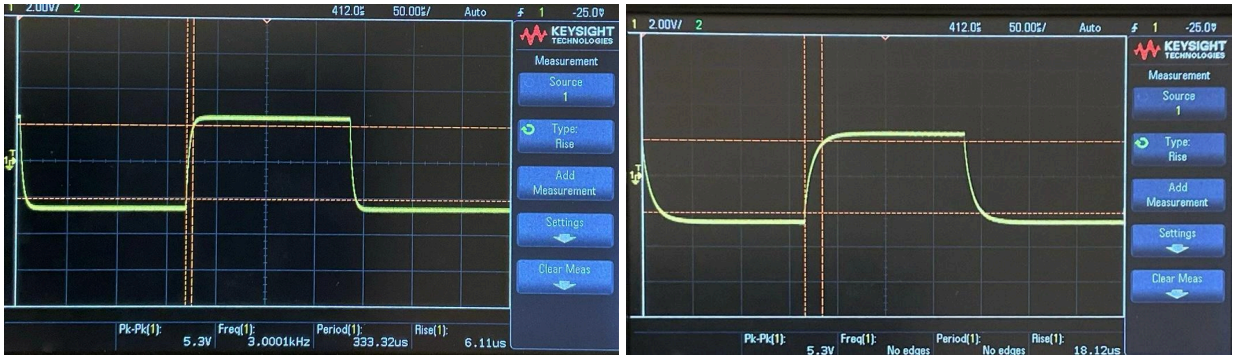
Once the distance was calculated and mapped, the range was not changed. This was picked based on previous usages of the tone. The frequency is not high enough to be painful to listen to which was essential during the testing portion.

Part 3: Capacitive Touch Sensor

3.1 RC Time Constant Based Measurement

To determine which resistor yielded a better change I tested various resistors in series to the touch sensor to create a low-pass filter as in Fig. 8 where 470kΩ had the largest change. Initially the driving wave was at 2kHz but driving it higher gave clearer wave forms so it was increased. After driving a 3kHz square wave from the signal generator, I observed which resistor created a better capacitor difference using the oscilloscope by placing a probe between the capacitor and resistor

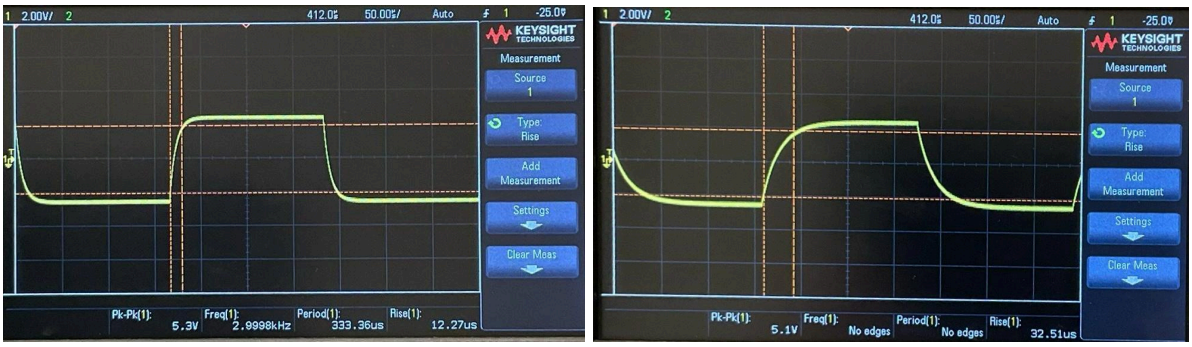
First I started with 100 kΩ resistors which only yielded a rise difference of 12.01 µs as is seen in the non touch rise of 6.11 µs, Fig. 13(a), and the touch rise difference of 18.12 µs, Fig, 13(b).



(a) Non-touched (b) Touched

Figure 13. RC Circuit for Capacitance Sensing with 100kΩ

The next resistor I tested was 220 kΩ and it yielded a rise difference of 20.24 µs as shown in the non-touch rise of 12.27 µs, Fig, 14(a), and the touch rise difference of 32.51 µs, Fig, 14(b).



(a) Non-touched (b) Touched

Figure 14. RC Circuit for Capacitance Sensing with 220kΩ

The last resistor I tested was 470 kΩ. This last resistor yielded the best results with the largest rise difference of 33.01 µs as can be observed by the non-touched rise of 26.59 µs, Fig. 15(a), and the touched rise of 59.60 µs, Fig. 15(b).

(a) Non-touched (b) Touched

Figure 15. RC Circuit for Capacitance Sensing with 470kΩ

For each resistor their change in capacitance is shown in Table 1. The RC time constant relationship can be used to calculate the capacitance by changing the relationship as in Eq. 4. From the calculations, the kΩ resistor is associated with a greater capacity.

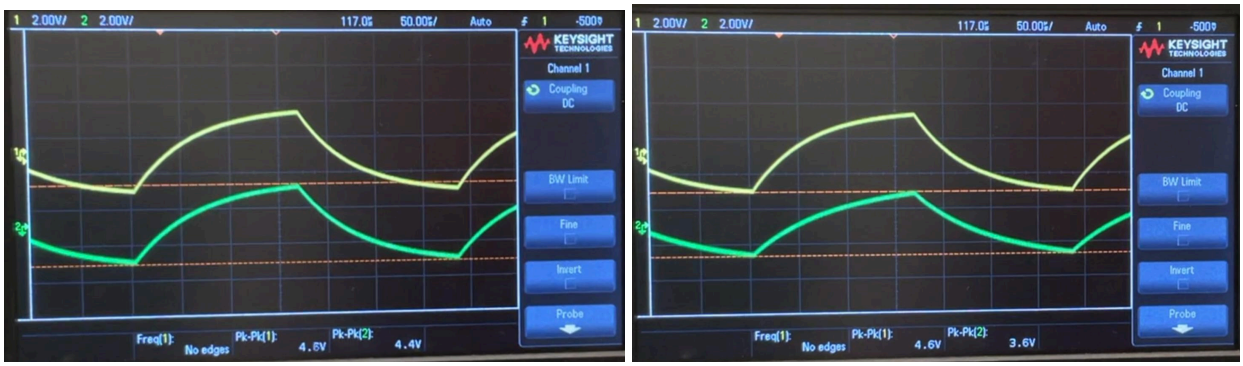
$$\tau = RC \rightarrow C = \frac{\tau}{R} \qquad (4)$$

Resistor	Non-Touched	Touched
100 kΩ	$C = \frac{6.11\ \mu s}{100\ k} = 61.1\ nF$	$C = \frac{18.12\ \mu s}{100\ k} = 181.2\ nF$
220 kΩ	$C = \frac{12.27\ \mu s}{220\ k} = 55.7\ nF$	$C = \frac{32.51\ \mu s}{220\ k} = 147.7\ nF$
470 kΩ	$C = \frac{26.59\ \mu s}{470\ k} = 56.6\ nF$	$C = \frac{59.60\ \mu s}{220\ k} = 126.8\ nF$

Table 1. Calculated capacitance from various resistor values

3.2 Capacitive Bridge and Differential Amp

Working with the configuration in Fig. 9 for the low and high pass filters we compared the difference between the two sides of the bridge. For the low pass bridge from Fig 9. (a) the peak to peak voltage between non touched and touched differs by 1.2 V. For the high pass bridge from Fig. 9 (b) the peak to peak voltage between non touched and touched differs by 1 V. The low pass bridge yielded better results. The waves generated by the low pass bridge are in Fig. 16, and the high pass are in Fig. 17.



(a) No-touch (b) Touch

Figure 16. “Low-Pass” Capacitive Bridge

(a) No-touch (b) Touch

Figure 17. “High-Pass” Capacitive Bridge

3.3 Relaxation Oscillator

Based on the readings from the oscilloscope, when the capacitive touch sensor is undisturbed, the difference is about 635 µs. Once the sensor is touched this difference changes to 1.9 ms. The first value can be used to determine the threshold value for the software implementation

(a) Non-touched (b) Touched

Figure 18. LM555 Oscillator Results

3.4 Capacitive Touch Software Interface

For the softwas portion, the value that was tested was how many readings would the average be calculated with. After testing three values (3, 4, 5) the last average yielded the best results and generated less noise.

V. Conclusion

In conclusion, this lab served as an exploration of sensor interfacing and signal processing, while providing an understanding of quadrature encoders, ultrasonic distance sensors, and capacitive touch sensors. Through hands-on implementation, I worked on Quadrature Encoder Interface (QEI) to output to RGB LED colors based on measured rotation. The ultrasonic distance sensor allowed me to practice using Least Squares for calibration. The last portion also gave me insights into RC time constants, capacitive bridges, and relaxation oscillators. This lab focused on building a foundation in sensor technologies, preparing us for more intricate applications.