Lab Group 5
Juliette Mitrovich
Sheila Moroney
Sujani Patel

Lab 2: Full Range Distance Detector

Summary

The purpose of this lab was to use two IR sensors to, through filtering and calibration, accurately detect an object within a 150 cm range. We first learned how to create an analog filter to remove the noise in the raw data from the IR sensors. To improve upon that filter, we learned how to use the digital filters available to us in MATLAB. We also learned through trial and error, the best way to integrate the two sensors' data to work together. One issue we ran into was at distances greater than around 75cm, there would be a "dip" in our distance output. The sensor would read the correct distance value for about three seconds, then the values would dip low for about 1 second. When we averaged the data over five seconds, both the average and the mode were within 5cm of the actual distance. We tried recalibrating the sensors and sampling the data at higher frequencies, but we could never get the dip to go away. One possible explanation for the dip is that the discharge of the capacitor in our analog circuit was causing disparities in the data. Although the IR sensors were finicky to work with, we would not change anything about this lab. It was a great learning experience to deal with filtering real data.

For our setup, we refer to the front sensor as "sensor 1" and the farther sensor as "sensor 2." We used 3D printed brackets to secure the sensors on a flat piece of cardboard. A picture of our setup can be found below in Figure 1.

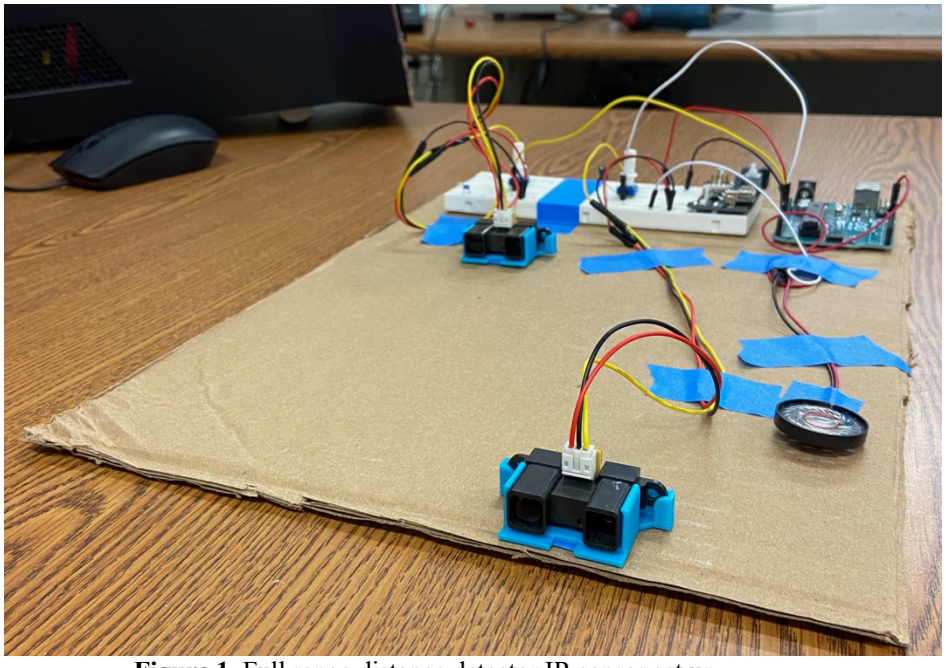

Figure 1. Full range distance detector IR sensor set up

Calibrating the IR Sensor

The calibration of the IR sensors included two parts: filtering the data from the IR sensor to remove noisy data and linearizing that filtered data. We started with filtering the data to ensure the linearization was not affected by outliers in the data. We started off by collecting noisy data from one of the IR sensors to determine at which frequency the noise was occurring. To do this, we placed a white box in front of the sensor and moved it in a wave-like motion. We collected data at a frequency of 125 Hz, wrote the data to the serial monitor, and copy and pasted the data into an Excel sheet to graph it. We repeated these steps for both sensors, individually. We chose a frequency of 125 Hz because we wanted to cancel out the noise coming from the lights in the lab room. The overhead lights have a frequency of 60 Hz. Using Nyquist's theorem, we knew we must sample data at a rate of at least twice that. A graph of the noisy data from sensor 1 and sensor 2 can be found below in Figure 2 and Figure 3, respectively.

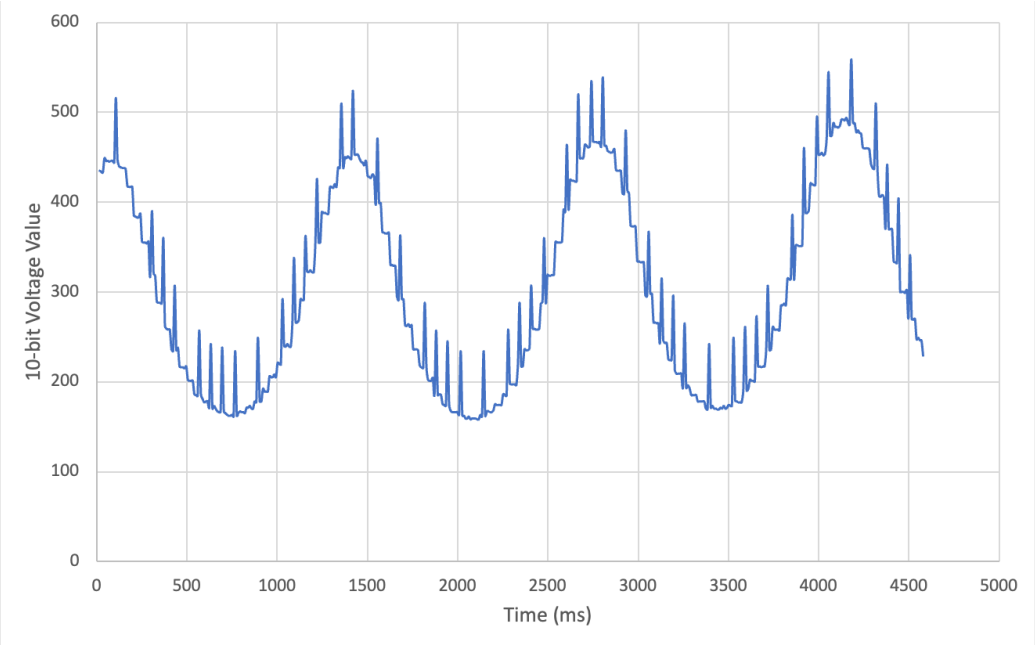

Figure 2. Unfiltered data of IR sensor 1

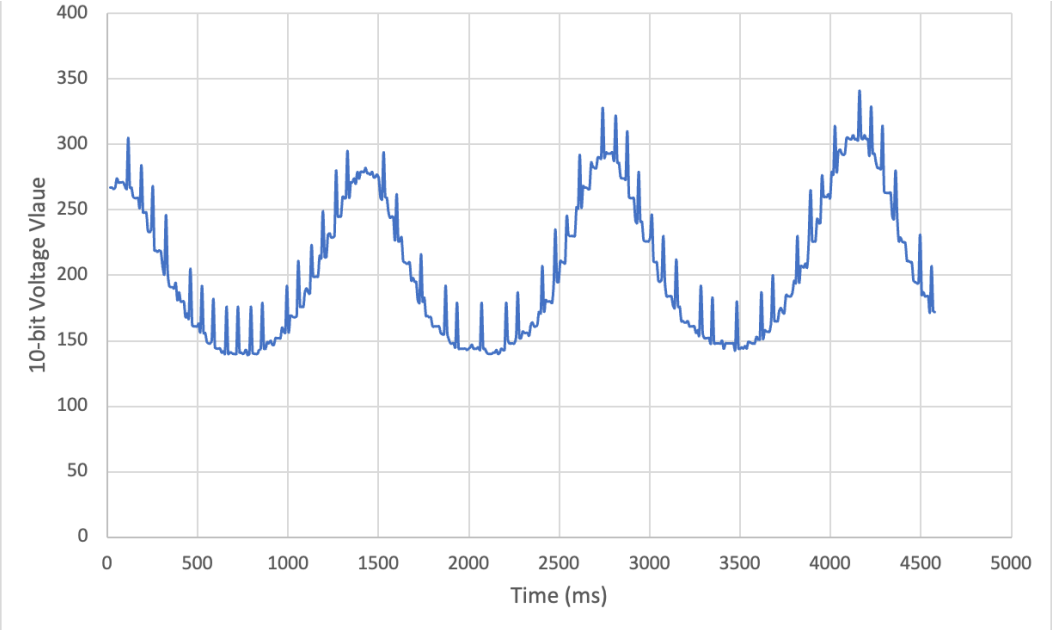

Figure 3. Unfiltered data of IR sensor 2

Once the data was graphed, we determined the frequency at which the noise was occurring. We looked at the spikes in the data and found the amount of time the spikes would occur for. We did this for 10 different spikes at various locations in the data and averaged the times. Then, to find the cut off frequency, we took the inverse of that time. The next step was to determine how to filter the data.

We know of two ways to filter data, an analog filter, or a digital filter. We started off by calculating the values needed for an analog filter. We know that the cutoff frequency, f_c, is equal to:

$$f_c = \frac{1}{2\pi RC}$$

Where R is the resistor value in ohms (Ω) and C is the capacitance value in farads (F). From the supplies we had available to us, we chose to use a capacitor equal to 4.7μF. Therefore, to find the R value needed, we rearranged the equation to be:

$$R = \frac{1}{2\pi f_c C}$$

From there, we found the appropriate R values for each sensor. The cut off frequency, the capacitance and the resistance values for each sensor can be found below in Table 1.

Table 1. Calculated values to create an analog filter

Sensor	Cut-off Frequency (HZ)	Capacitance (μF)	Resistance (Ω)
1	14.28	4.7	2370
2	16.66	4.7	2031

The next step was to implement each analog filter. We followed the same steps for data collection as the unfiltered data and graphed it in excel. The graphs for the analog filtered data for sensor 1 and sensor 2 can be found below in Figure 4 and Figure 5, respectively.

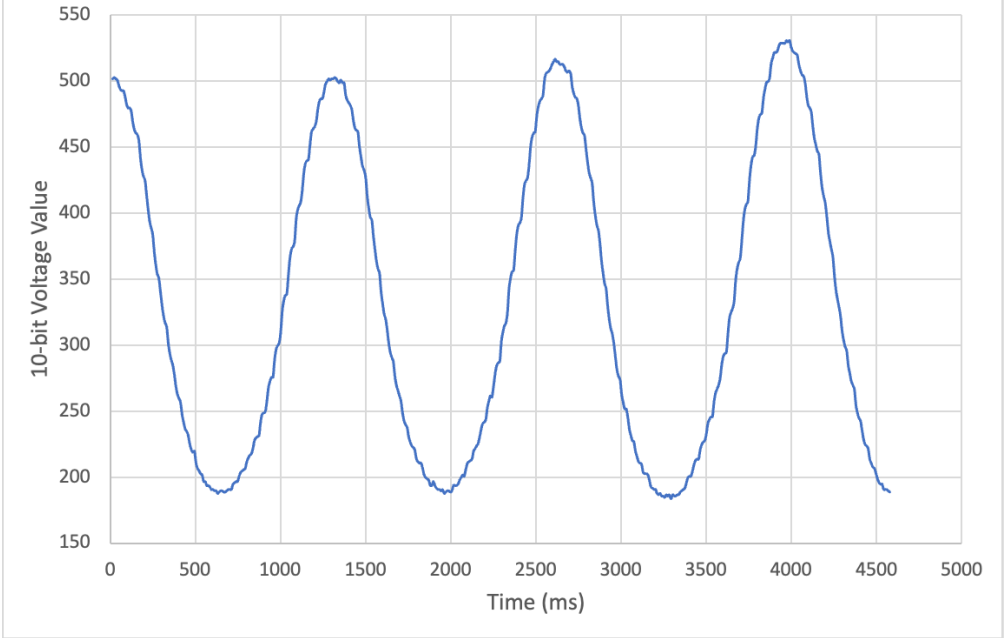

Figure 4. Analog filtered data of IR sensor 1

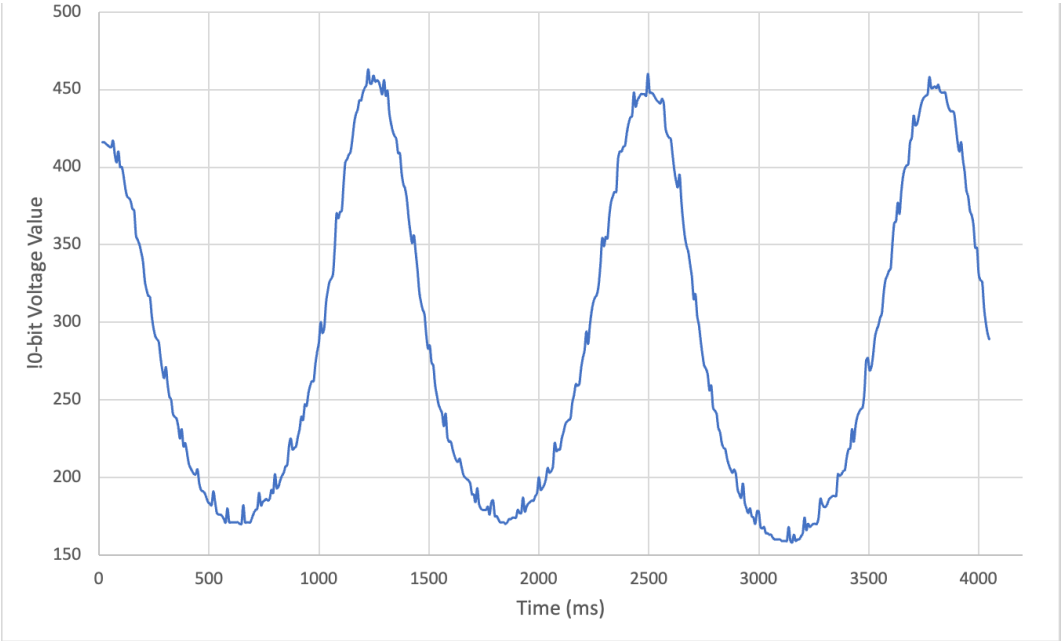

Figure 5. Analog filtered data of IR sensor 2

As you can see from the two graphs, there is significantly less noise, meaning the analog filters did what we expected. However, they were still not perfect. To further eliminate the remaining noise, we decided to implement a digital filter on top of the analog filter. In other words, we took the analog filtered graph, found the cut-off frequency of the noise, and used that to design a digital filter. The cut-off frequencies of the filtered noise can be found below in Table 2.

Table 2. Cut-off frequencies of the analog filtered data for each sensor

Sensor	Cut-Off Frequency (Hz)
1	27.77
2	22.22

We chose to design a Butterworth filter, using the MATLAB command "butter" to find the coefficients. Once we had the coefficients, we implemented the digital filter in our code and collected the sensor data in the same manner as previously mentioned. See Figures 6 and 7 below for a visual of the digital and analog filtered data.

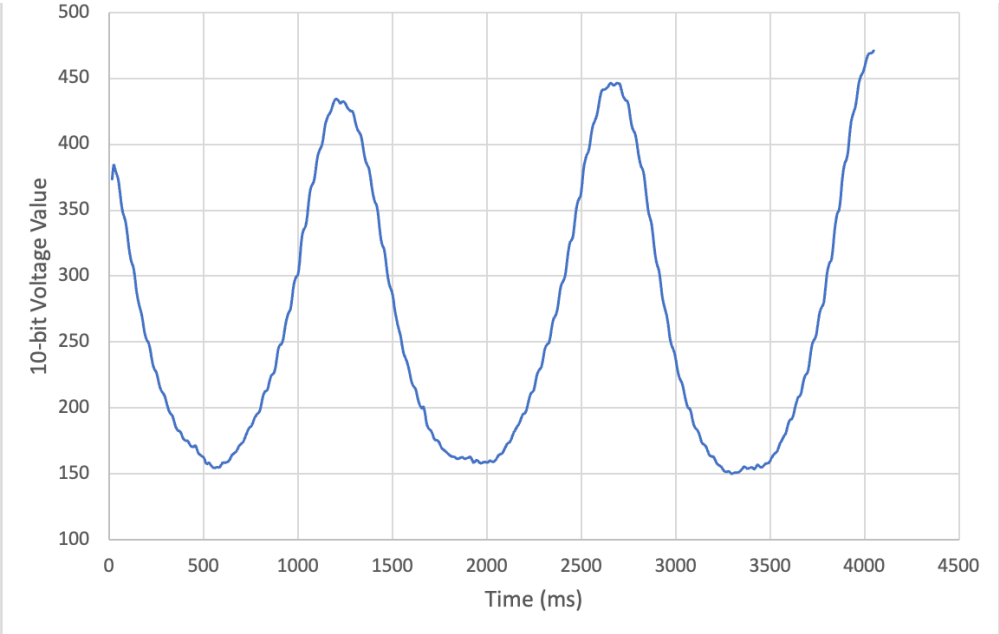

Figure 6. Digital and analog filtered data of IR sensor 1

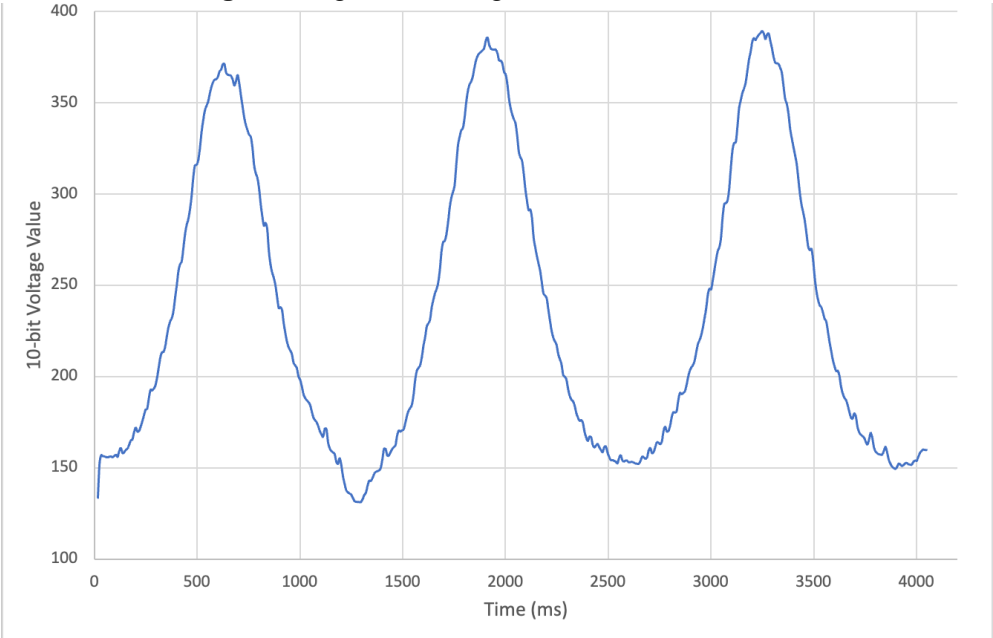

Figure 7. Digital and analog filtered data of IR sensor 2

As you can see from the two graphs, the digital filter and analog filter combined improved upon the amount of noise still present. Even though the data was still not perfectly smooth, we determined that this would be sufficient to detect the distance correctly.

Now that we had a satisfactory filter for our sensor, the last step was to linearize the sensor. Because the Sharp IR sensor is such a commonly used device for distance detection, we did research to see if there was linearization data readily available to us, and there was. We used this linearization equation [1]:

$$D = \frac{16569}{V_{a2d} + 25} - 11$$

Where D is the distance in cm and V_{a2d} is the 10-bit voltage value from the sensor. We implemented this equation into our code and placed a white box at various distances to see the accuracy of the data. We found that both sensors were able to read the correct distance value within a few centimeters. At this point, we finished the sensor calibration.

Staggering the Two Sensors

The two sensors were offset by 20cm, with the sensor 2 placed behind sensor 1. We chose to do this because the sensors only have an accurate range from 20cm to 150cm. However, for this task, we needed the ability to read a distance closer than 20cm. With this set up, when an object gets too close to the front sensor, we can always rely on sensor 2's reading since the box will never be closer than 20 cm to it. Sensor 2 was also placed at a horizontal distance of 5 cm to the right of sensor 1 so that the two sensors would not interfere.

Fusing the Sensor Signals

There were many ways to approach this part of the lab, we decided to use a series of if statements that changed the weights of each sensor's distance within three different ranges. Due to the limits of the sensors, we used only sensor 2 for the first 20 cm of the range. Then from the 20-100cm range, we used the average distances from both sensors. And after 100cm, we only used the distance from sensor 1. We found that sensor 1 was reading lower values in the 100-150cm range so we increased the weight multiplier to 1.05.

Mapping The Distance to a Range of Notes

To map the distance to a range of notes we used the "map()" function in Arduino. This function uses the syntax: map(value, fromLow, fromHigh, toLow, toHigh). Our "value" was the distance output from the two IR sensors. The "fromLow, fromHigh" values were our distance range (0 to 150 cm). The "toLow, toHigh" values were the min and max frequencies of the range we wanted our speaker to play.

References

1. “Sharp IR distance sensor linearization.” limulo, 6 August 2017, limulo.net/website/coding/physical-computing/sharp-linearization.html.

Appendix A: Source code used in Arduino

This appendix presents the source code used on the Arduino. We had two separate programs, one to run the sensor calibration and one to run the final program for the lab check off. The sensor calibration code can be found below in Figure A-1 and the final code for lab check-off can be found in Figure A-2.

Figure A-1. Sensor Calibration Code

```
/*
  ME 545 Lab 2
  IR sensor measurement
  Filtering - raw data, analog filter, analog and digital filter
  Juliette Mitrovich, Sujani, Sheila Moroney
*/

// Polling loop variables
unsigned long startTime = 0; // variable for the start time
unsigned long currentTime = 0; // variable for the current time
unsigned long elapsedTime; // variable to store the elapsed time (currentTime - startTime)
int count = 0; // initialize a counter

// IR sensor, Raw data pins and variables
float IRsensorRaw = A0; // set IRsensor as analog pin 0
float sensorValueRaw; // variable to store the output of IRsensor

// IR sensor, Analog filtered (AF) data pins and variables
float IRsensorAF = A1;
float sensorValueAF;

// First order digital filter coefficients
// values found at different sampling and cut off frequencies, change as needed

// fc = 22.22Hz, fs = 125Hz
float a1 = -0.2309;
float b0 = 0.3846;
float b1 = 0.3846;

// fc = 27.77Hz, fs = 125Hz
//float a1 = -0.0875;
//float b0 = 0.4563;
//float b1 = 0.4563;

// Digital filter variables
float Vold = 0; // initialize the first input
float yold = 0; // initialize the first output
float yfilt_f; // initialize variable for the filter equation
```

```
float digitalFilter; // initialize variable to store first order digital filter values

void setup() {
  Serial.begin(115200); // initialize the serial monitor
  currentTime = millis(); // set the currentTime equal to millis function
}

void loop() {
  currentTime = millis();
  elapsedTime = currentTime - startTime; // calculate the elapsed time

  // collect 500 sampling points at a frequency of 125 Hz
  if (8 <= elapsedTime && 500 >= count) {

    // Collect raw voltage data
    sensorValueRaw = analogRead(IRsensorRaw);

    // Collect analog filter voltage data
    sensorValueAF = analogRead(IRsensorAF);

    // collect digitally filtered data with analog filtered data as the input
    digitalFilter = (b0 * sensorValueAF) + (b1 * Vold) - (a1 * yold);
    yold = digitalFilter;
    Vold = sensorValueAF;

    // Print important data to serial monitor
    Serial.print(currentTime);
    //    Serial.print(",");
    //    Serial.print(sensorValueRaw);
    //    Serial.print(", ");
    //    Serial.println(sensorValueAF);
    Serial.print(", ");
    Serial.println(digitalFilter);

    startTime = millis();
    count += 1; // add 1 to variable ‘count’
  }
}
```

Figure A-2. Final code for the lab check-off

```
*
  ME 545 - Mechatronics
  Lab 2 - IR Sensors final code
  Lab Group 5
  Juliette Mitrovich, Sijani, Sheila Moroney
*/
// Polling loop variables
unsigned long startTime = 0; // variable for the start time
unsigned long currentTime = 0; // variable for the current time
unsigned long elapsedTime; // variable to store the elapsed time (currentTime - startTime)

// IR sensor 1 variables
float IRsensor1 = A0; // set IRsensor as analog pin 0
float sensorValue1; // variable to store the output of IR sensor 1

// IR sensor 2 variables
float IRsensor2 = A1; // set IRsensor as analog pin 0
float sensorValue2; // variable to store the output of IR sensor 1

// digital filter coefficients, sensor 1 (fc = 27.77Hz, fs = 125Hz )
float a11 = -0.0875;
float b01 = 0.4563;
float b11 = 0.4563;

// digital filter coefficients, sensor 2 (fc = 22.22Hz, fs = 125Hz)
float a12 = -0.2309;
float b02 = 0.3846;
float b12 = 0.3846;

// digital filter, IR sensor 1 values
float Vold1 = 0; // initialize the first input
float yold1 = 0; // initialize the first output
float digitalFilter1; // initialize variable to store first order digital filter values

// digital filter, IR sensor 2 values
float Vold2 = 0; // initialize the first input
float yold2 = 0; // initialize the first output
float digitalFilter2; // initialize variable to store first order digital filter values

// variables to store distance calculations
float distance1; // store distance calculation from IR sensor 1
float distance2; // store distance calculation from IR sensor 2
float distance2_adj; // adjusted distance from sensor 2, set 20cm back
float distance; // final distance calculated from both IR sensors
```

```
// weight variables for sensor fusion
float w1;
float w2;

// speaker variables for mapping tone
int speakerOut = 9;
int pitch;

void setup() {
  Serial.begin(115200); // initialize the serial monitor
  pinMode(speakerOut, OUTPUT);
  currentTime = millis(); // set the currentTime equal to millis function
}

void loop() {
  currentTime = millis();
  elapsedTime = currentTime - startTime; // calculate the elapsed time

  // collect data at a fequency of 125 Hz (every 8 milliseconds)
  if (0 == (elapsedTime % 100)) {

    // Collect IR sensor 1 data
    sensorValue1 = analogRead(IRsensor1);

    // Collect IR sensor 2 data
    sensorValue2 = analogRead(IRsensor2);

    // digitially filter IR sensor 1 data
    digitalFilter1 = (b01 * sensorValue1) + (b11 * Vold1) - (a11 * yold1);
    yold1 = digitalFilter1;
    Vold1 = sensorValue1;

    // digitally filter IR sensor 2 data
    digitalFilter2 = (b02 * sensorValue2) + (b12 * Vold2) - (a12 * yold2);
    yold2 = digitalFilter2;
    Vold2 = sensorValue2;

    // Convert filtered IR sensor data to distance
    distance1 = (16569.0 / (digitalFilter1 + 25)) - 11;
    distance2 = (16569.0 / (digitalFilter2 + 25)) - 11;
    distance2_adj = distance2 - 21;

    // Use two distances to calculate the final distance
    // if IR sensor 2 reads a distance of 120cm or more, only use IR sensor 1 data
    if (120 < distance2) {
      w1 = 1.05;
```

```
    w2 = 0;
    distance = (w1 * distance1);
   }

   // if IR sensor 2 reads a distance of 40cm or closer, only use IR sensor 2 data
   if (40 >= distance2) {
    w1 = 0;
    w2 = 1;
    distance = ((w1 * distance1) + (w2 * distance2_adj)) / (w1 + w2);
   }

   // if IR sensor 2 reads a distance between 40cm and 120 cm, average the two sensor distances
   if (120 >= distance2 && 40 < distance2) {
    w1 = 1;
    w2 = 1;
    distance = ((w1 * distance1) + (w2 * distance2_adj)) / (w1 + w2);
   }

   // generate varying pitch from the varying distance
   pitch = map(distance, 0, 150, 120, 1500);
   tone(9, pitch);

   // Print data to the serial monitor
   //      Serial.print(distance1); // print when you want to test sensor 1 distance
   //      Serial.print(", ");
   //      Serial.print(distance2); // print when you want to test sensor 2 distance
   //      Serial.print(", ");
   Serial.println(distance); // print when you want to test sensor fusion distance

   startTime = millis(); // reset startTime to allow for 125Hz of polling
  }
}
```