Ultrasonic Sonar: ECE 203 Final Project

Noah Avis: na7213@princeton.edu

Tom Wang: tw6600@princeton.edu

Demonstrated on April 25, 2025

Submitted on May 2, 2025

Table of Contents

This paper represents my own work in accordance with University regulations.

/s/ Noah Avis

/s/ Tom Wang

Statement of Objective

The objective of this project is to create an ultrasonic sensor that allows us to detect how far away something is in front of the sensor. To do this, we needed to accomplish the following goals:

1. Correctly wire a circuit such that it utilizes two ultrasonic transducers to detect objects; one will be used to transmit signals, and the other to receive signals
2. Connect the circuit to an ESP32 so that the ESP can control the output of the transmitting ultrasonic transducers and receive the output of the receiving ultrasonic transducers
3. Create a program that is able to determine the time between sending and receiving the signal based on the input of the receiving ultrasonic transducer
4. Write a program that is able to visualize the object in front of the sensor based on the output of the ESP

This sensor can be used everywhere from civilian to industrial usage. It can be used for object detection to open automatic doors. It can be used to help visually impaired people avoid obstacles (if the output of this circuit was sound). It can be put on cars to help with object avoidance and parking assist. The application of this sensor is very broad given its usefulness.

System Overview

The working principle of an ultrasonic distance sensor is simple. A transmitter sends out a pulse of ultrasonic sound. This sound travels outwards until it reaches the object whose distance is being measured. Then the sound reflects off the object and reaches the receiver, where it is converted back into a voltage signal. The time between transmitting and receiving is measured, and one can find the distance using the following formula

$$d = t * 343 / 2$$

where d is the distance in meters, t is the time delay to receiving in seconds, 343 is the speed of sound in meters per second, and the division by two reflects the fact that the sound must travel to and back from the object.

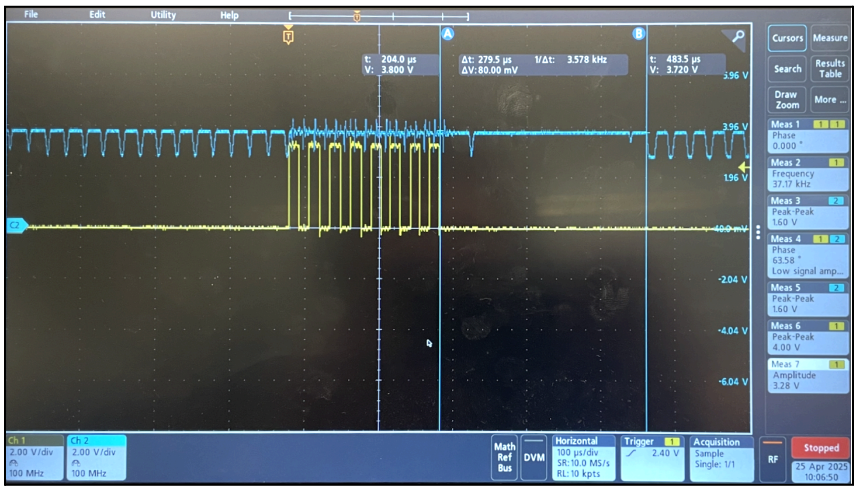

Fig. 1 Oscilloscope Capture of Transmit and Receive Delay

Focusing on our circuit overview, the ultrasonic distance sensor is primarily divided into two sides: transmitter and receiver. As shown in Figure 2, a level shifter is used to convert a 3.3V peak-to-peak 40kHz square wave pulse from the ESP 32 into around 15V. 40kHz is used because it is the manufacturer recommended frequency for the ultrasonic transducers. The receiving end then uses an op amp to amplify the voltage signal for processing by the ESP 32. Distance is measured via sending an 8 peak ultrasonic pulse and counting the elapsed time in microseconds from sending the first pulse to when a pulse is seen on the receiving end. 8 pulses are sent to increase the strength of the echo and open up signal processing features such as pattern recognition, but our system does not implement these. The eight transmit pulses and receive time delay can be seen in the oscilloscope capture in Figure 1. It is important to note that our MAX232 was not set up correctly in this capture, corresponding to the lower peak voltage.

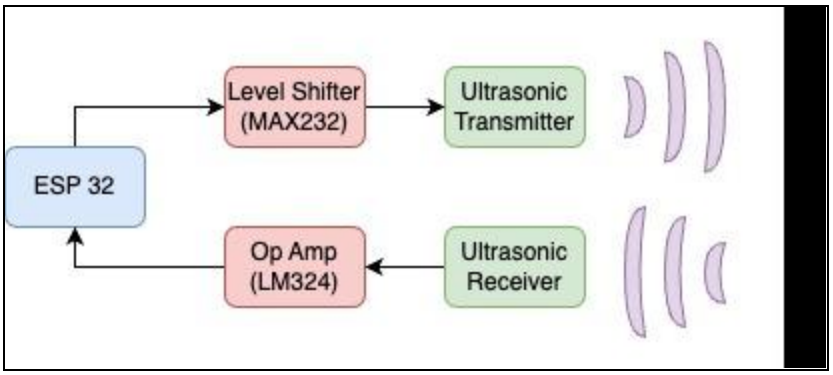

Fig. 2 Hardware Block Diagram for Ultrasonic Distance Sensor

Diving into the transmitting side, operating the MAX232 requires additional capacitors for it to function. A decoupling capacitor is placed between pins 15 and 16 to filter power to the IC. Two more capacitors are required, C1 and C2 as stated by the datasheet, on pins 1, 3, 4, and 6 for the internal charge pump to function. The charge pump takes advantage of the fact that capacitors cannot change their voltage instantly by using fast switches to boost voltage. The other two capacitors are used as storage capacitors to store the boosted voltage needed for the rest of the circuit while the charge pump is switching stages.

The receiving side is slightly more complicated as additional filters and components are required. The LM324 is a quad op amp IC, and we use three of them chained together to get the amplification required for the ESP 32 to process the signal. Because our op amps are powered by a single 5V supply and cannot output a negative voltage and our ultrasonic receiver outputs an AC signal centered around 0V, we need DC offsets to lift the signal into the op-amps range so it can be amplified. These offsets are listed in Figure 3 as V1, V2, and V3 and are determined by potentiometers giving 0-5V in our real circuit. Because op amps amplify the difference between the input terminals, we use these offsets to amplify the signal while also tuning the offset of the final output signal of the op amp chain signal to be the perfect level for the ESP 32 to process. We look for a received pulse by checking if an input pin goes high, and the threshold voltage for the ESP 32 to detect pin high is around 1.8V, and by setting V1, V2, and V3 correctly, the ESP 32 can correctly take measurements when it sees a pulse. Additionally, the op amp chain combines 3 inverting amplifiers each having an amplification of 100k/10k = 10. So, the final

amplification is 1000x, which is necessary since the ultrasonic transducer produces voltages on the order of a few mV.

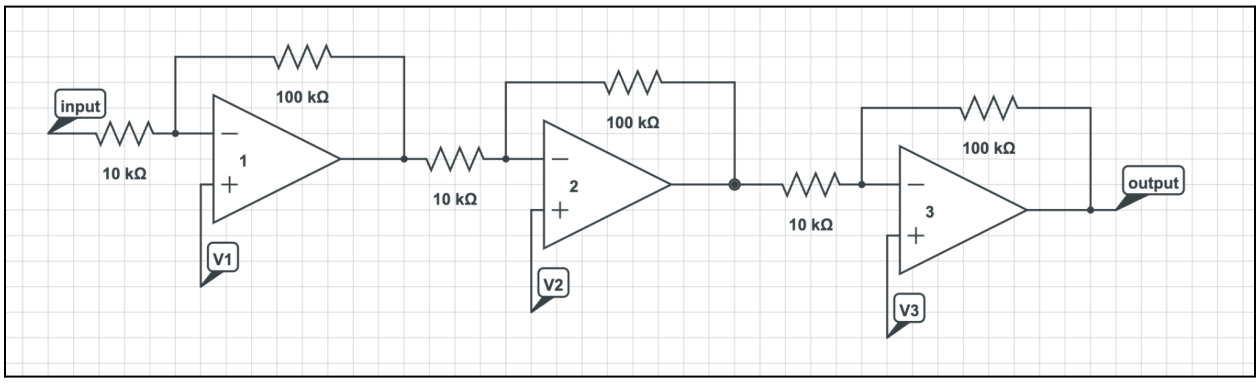

Fig. 3 Op Amp Configuration Schematic from CircuitLab

Additionally, there is an RC filter present on the output of our receiver. This is a high pass filter that serves to attenuate low frequency noise such as mains power or other noise that could throw off our time measurements. At a frequency of 40kHz, the capacitor has an impedance of around 400 Ohms, so it is essentially absent relative to the 10k resistor. Therefore, this filter passes the signal we want to process so we get a cleaner signal to give to the ESP 32.

The other and significantly less complicated portion of our project is a rotating base to scan for objects and a computer output. The base is simply a servo motor, driven by the ESP 32, that rotates the distance sensor to detect close objects in a sweeping area. The sensor takes time measurements every 2 milliseconds and outputs the measurement along with the servo position to the computer via serial communication. Then, using a custom UI written in Processing 4, we read the serial data using Processing's integrated serial library and convert the elapsed time into distance. This is then displayed as a line sweeping according to the servo's position with a red line indicating an object is detected. The distance between the start of the red line and the origin of the line represents the distance from the object to the sensor.

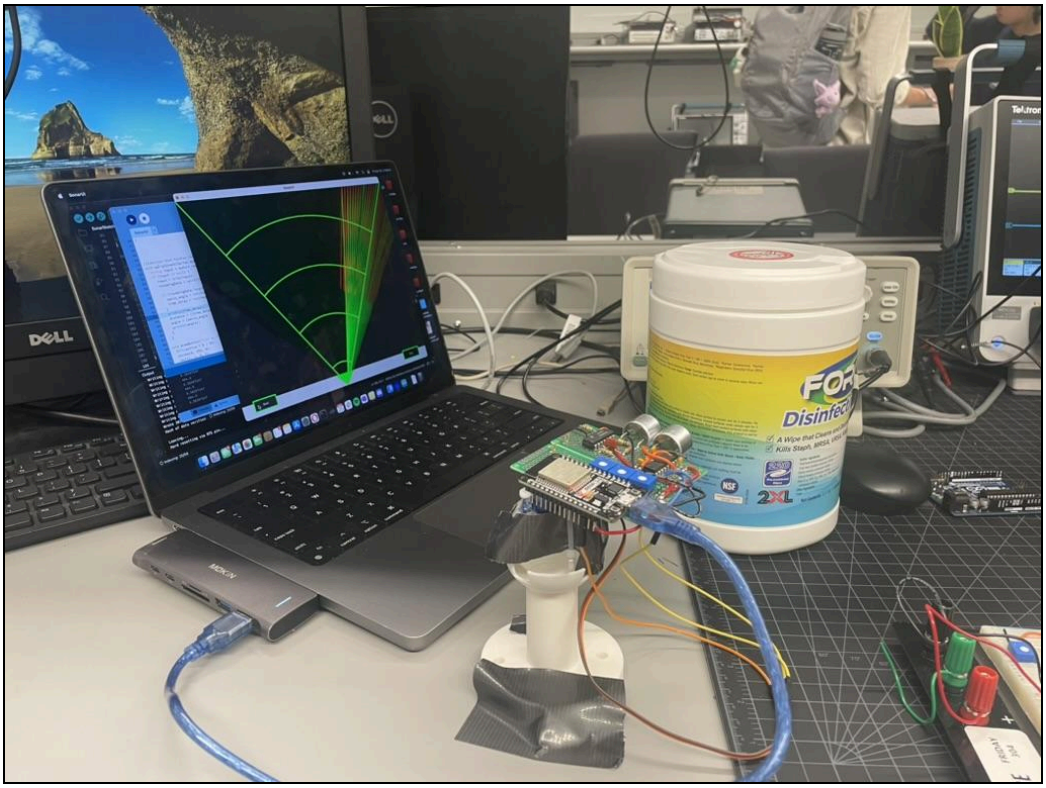

Fig 4. Complete Operation with Computer Output

Performance Analysis

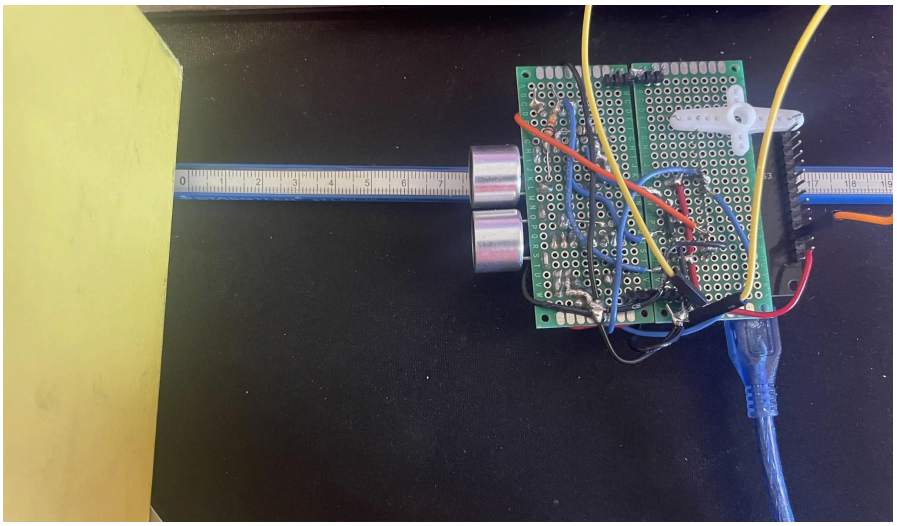

Fig. 5 Performance Evaluation Setup

The primary metric for the performance of our device is the accuracy of measured distances to an object compared to the actual distance. We used a large flat object (a book) to obtain the cleanest sound reflections possible, used a ruler to measure the actual distance between the transducers and the object, and tested distances from 1-12 cm in 1 cm increments. This setup is illustrated in Figure 5. We found that for mid ranged distances, the sensor was extremely accurate, down to fractions of a millimeter. In Figure 6, deviation from the red line shows error in the ultrasonic distance measurement. We can see that from 2.5 - 9 cm the measurement is very accurate.

Looking at the Adafruit HC-SR04, the sensor our project imitates, the datasheet claims accurate distance measurements from 10-250cm. The deviations of our sensor's measurements at longer ranges is most likely due to timing inaccuracies being exacerbated by the longer distance. The HC-SR04 has dedicated timing hardware which enables it to be more accurate at longer distances. Additionally, we tuned a delay constant in the software that corresponds to processing delays of the ESP 32. We tuned this constant to give an accurate distance measurement at 8 cm – midrange for our task. It could be possible that tuned at a longer distance, our sensor could see better accuracy at these ranges.

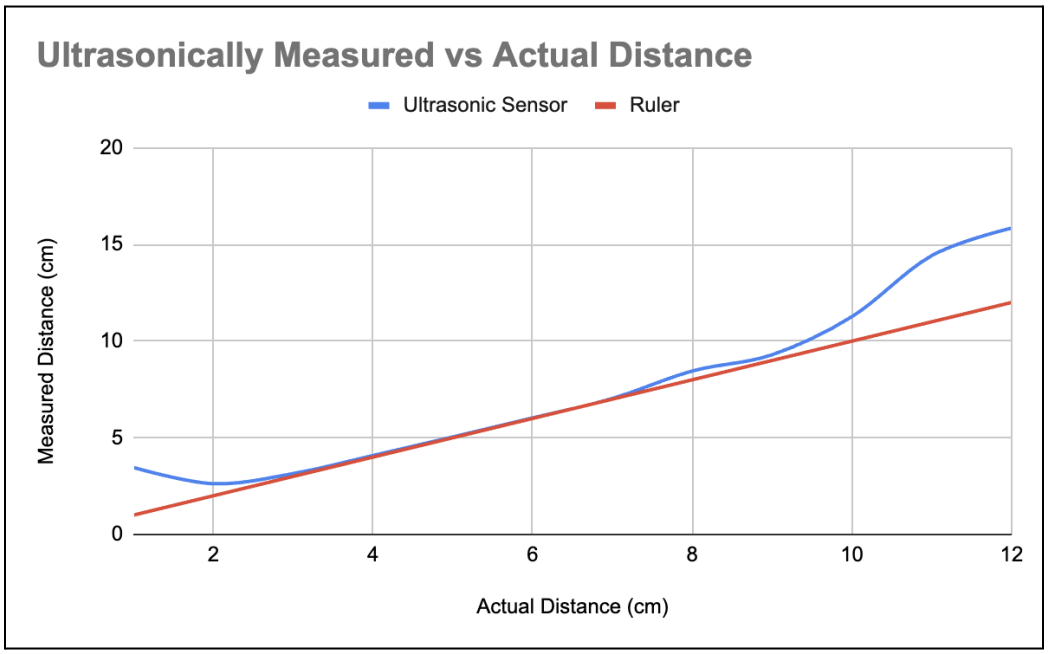

Fig. 6 Evaluation of Accuracy of Length Measurements

Circuit Documentation

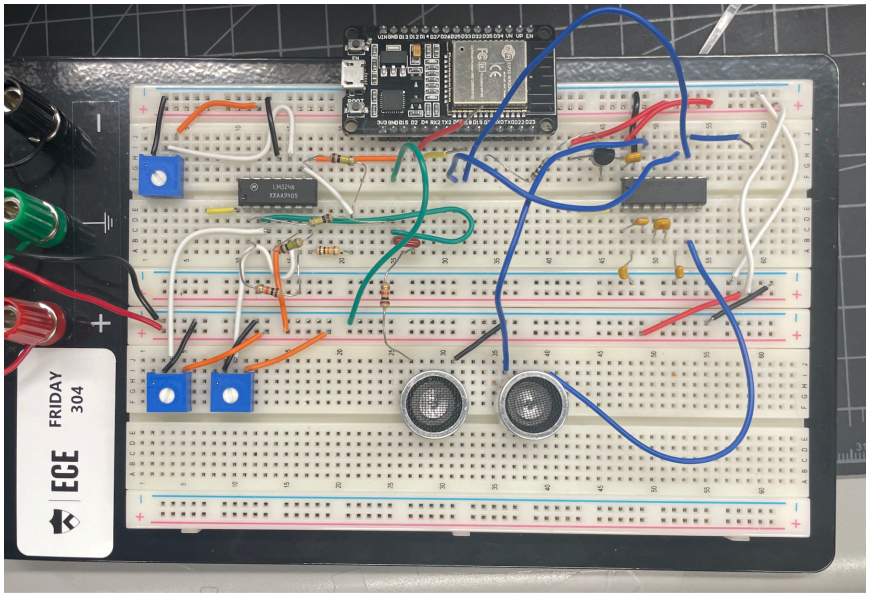

Fig. 7 Breadboard Circuit Layout - Used for Proof of Concept

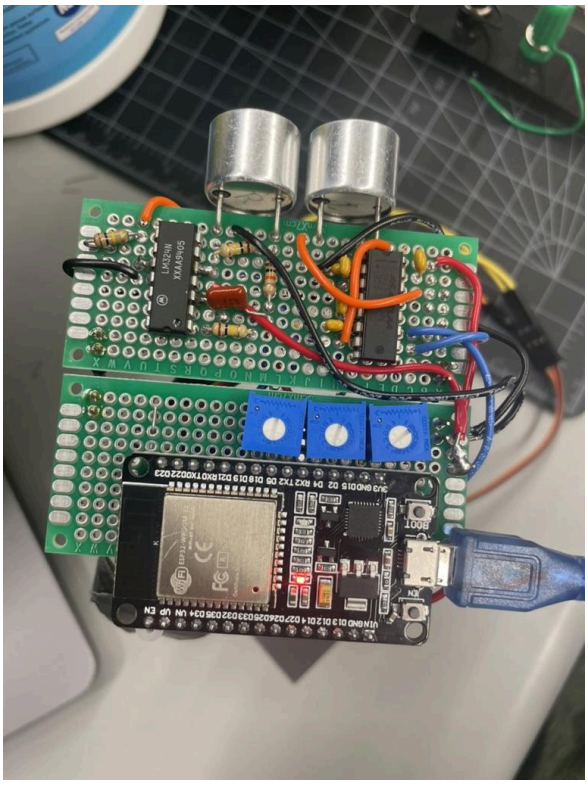

Fig. 8 Final Protoboard Layout (Components on bottom not pictured)

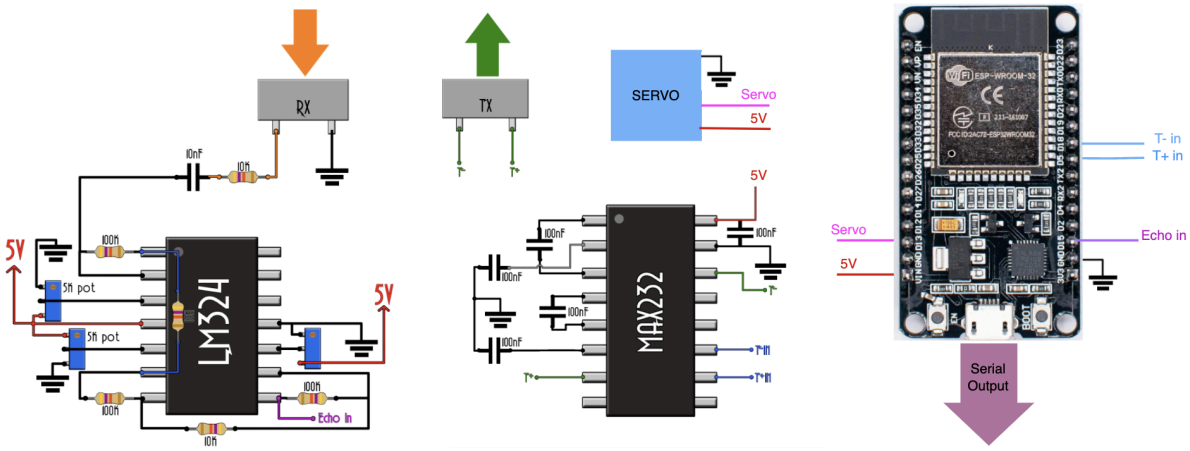

Fig. 9 Full Circuit Schematic – Adapted Version of Electro Noobs Tutorial

Design Challenges and Solutions

We had challenges tuning the op-amp to the correct sensitivity. If it was too sensitive, the voltage would go to Vss, and if it was not sensitive enough it wouldn't pick up the voltage from the receiving transducer. We ended up tuning it first by using the function generator to input a current into the circuit, then tuning the potentiometer to get the op amp to the correct sensitivity. We then tuned it by running the ultrasonic transducer to make sure that we got the right signal.

Another issue we had was software. In order to correctly display the visual for the radar, we needed the position of the servo motor, which meant we needed to use the same ESP that we control the transducers to control the servo. Our code for the radar was to time the difference between sending out the signal and receiving the signal to find the distance, but if we add the loop that controls the servo, it messes up the time. To correct this, we tried to do parallel processing on the ESP 32 so the servo could run and not mess up the time of the ultrasonic transducers. However, our attempt at implementing it significantly decreased the accuracy of the radar. After several iterations on the code, the accuracy was better, but the visualizer would pick up random "artifacts" that weren't actually there. We weren't able to fix the artifacts that were appearing on the screen in the end but managed to get the timing correct while also running the servo by building our own PWM servo control function instead of using the library.

Our last challenge was finding a suitable base that would hold the servo and attach the servo onto the circuit. To attach the servo onto the circuit, we had to opt for a less optimal

placement because the circuitry was in the way. As for the base, we used an old part that was from another project, which we hot-glued and duck-taped the servo onto.

Future Improvements

The circuit was very noisy despite the high pass filter at the output of the receiving ultrasonic transducer. The obvious improvement that we could make is to make a band pass filter (an RLC circuit in series) which would reduce the high frequency noise that we were getting.

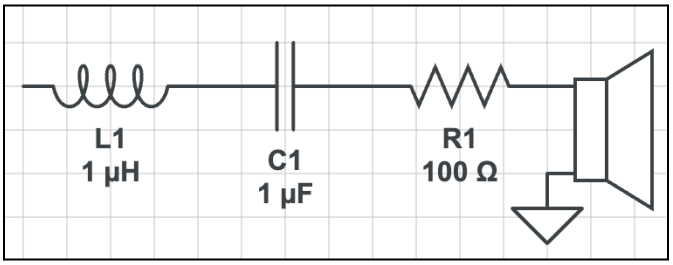

Fig. 10 Example Band Pass Filter From CircuitLab

To further increase the accuracy of our sensor and display of output, we can use two ESP32, one to control the ultrasonic transducers and one to control the motor. This would allow us to simultaneously control both without interference with one another. Using two processors would solve the issue of artifacts appearing on the display discussed in the challenges section. It would also simplify the code significantly and allow for easy readability.

Lasly, to determine if there was an object in front of the sensor, we programmed the ESP32 to look out for any peak in voltage from the receiving ultrasonic transducer. This worked fine, but the accuracy could be improved. The transmitting transducer is programmed to output 8 pulses every cycle. If we were to look for 8 consecutive pulses, or 8 pulses that are similar in timing of the output pulses, it would significantly reduce false positives from the sensor and increase accuracy.

These improvements would be cheap to implement but effective. The materials required to implement the improvements are cheap and easy to obtain (only an inductor and additional ESP32) but would help the accuracy of the sensor.

Appendices

References:

All About Charge Pumps:
https://www.allaboutcircuits.com/technical-articles/switched-capacitor-circuits-charge-pump-circuit-basics/

MAX232 Datasheet:
https://www.ti.com/lit/ds/symlink/max232.pdf?ts=1746128384296&ref_url=https%253A%252F%252Fwww.google.com%252F

LM324 Datasheet:
https://www.ti.com/lit/ds/symlink/lm324.pdf

HC-SR04 Datasheet:
https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/692/3942_Web.pdf

Electronoobs Tutorial:
https://electronoobs.com/eng_arduino_tut36.php

BOM:

Name / Part no.	Quantity	Price (Quantity * unit price)	Seller	Link	Notes
Ultrasonic Transducers	1	15.99	Amazon	https://www.amazon.com/FainWan-TCT40-16R-Ultrasonic-Transceiver-Transmitters/dp/B09PN5G8N4/140-7744924-5833016?pd_rd_w=wlYri&content-id=amzn1.sym.4e570b94-6aa5-4986-a730-b6fb915405	20 pcs. 10 transmitters and 10 receivers

				fc&pf_rd_p=4e570b94-6aa5-4986-a730-b6fb915405fc&pf_rd_r=54MPPE6VS051SQQCZA1K&pd_rd_wg=XkvjR&pd_rd_r=1c51ece8-bea6-4ef4-8e1f-5d4a0104216f&pd_rd_i=B09PN5G8N4&psc=1	
LM324	2	7.99	Amazon	https://www.amazon.com/ALLECIN-LM324N-Quadruple-Operational-Amplifier/dp/B0CBKKQV3J?crid=7Z1PL7HOUCTP&dib=eyJ2IjoiMSJ9.FagR7wV8IBgogmDvm-okdf5sWhsOiqmwjtp8V3fCV0g_pOmR3veQlgZOC0ZUgMnHj49vxNbt7NSa-llaSHYuzZ6rkw_Lu8w9QRJiUgUP_4o5f8yRjL3iZXIwDff0QupyyO13Nb7RBNa2BB48rZ2KGzW2g1ofMmbMzzbpf3qPGQZ5MMFyNbWqgPVzVSPnUVh9BW9HltvYkFy1HDUONkyb1zLUrMnEqDOYwjdvzwdeFSk.RBW5lBSS5bw8gLMtK_C5MF2b1TXHSgMohBlZjJBPv1E&dib_tag=se&keywords=lm324&qid=1742225682&sprefix=lm324%2Caps%2C98&sr=8-3&th=1	Pack of 30. We don't need that many. They probably have these already.
MAX232	3	$5.43	Digikey	https://www.digikey.com.br/en/products/detail/texas-instruments/MAX232N/277048?gQT=2	
ESP 32	1	16.99	Amazon	https://www.amazon.com/ATmega328P-Arduino-Compatible-Arduino-Voltage-Compatible/dp/B0D83J2TJJ/ref=asc_df_B0D83J2TJJ?mcid=c98ec68fa765345b9a5c06a810b5b073&hvocijid=12377042114938183112-B0D83J2TJJ-&hvexpln=73&tag=hyprod-20&linkCode=df0&hvadid=721245378154&hvpos=&hvnetw=g&hvrand=12377042114938183112&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9003941&hvtargid=pla-22	

				81435178818&th=1	
Capacitors	1		Amazon	https://www.amazon.com/BOJACK-Ceramic-Capacitor-Assortment-Capacitors/dp/B07P8N8BW9?crid=YU2B55A0M4KG&dib=eyJ2IjoiMSJ9.tq8WpqSLISopjnCWXK8uYl-8u-4ffyYpYtY8sl1ARfrOG1Ofd2H4NWXEvZAUWGLa4IYM-KJvhKD-9sWrEdX5JsSrmCiVAxtgX9wN_qFgp-Hd2AP8vtq9t7ipeTW8dd5S7VqYXnf9FnZdbEffy9_B69lqwVWwWH55mhqHHNEnEZJy_0AKVgQDrZV2kNHnF1w6GPJwB0UtdtKUndJ550Xdd5rlKGQWYxjLFBOpKSer1Tc.PvccVgwLfpOMuL3kQ-hQH5qS5BSg6lcN7piw5fX4bxA&dib_tag=se&keywords=100+nf+ceramic+capacitor&qid=1742399000&sprefix=%2Caps%2C77&sr=8-6	Assorted Capacitors. We need 100nF and 10nF
Servos	1	6.99	Amazon	https://www.amazon.com/WWZMDiB-SG90-Control-Servos-Arduino/dp/B0BKPL2Y21?crid=2006ACLCW2DPC&dib=eyJ2IjoiMSJ9.w0Xv0V9ezw2OMZIJmWpQPgnAICMVOFootLhXD3Wsvgw-wjTRsEjMeF11P3qTB3NPAC5NQggpeOHSVRu4XRelAnQzdk5IypJy4klrVexErrfyTTBcvRaA_PPjlzzBmIUBVARsESlYu4ybd3morujca3_5_DpWd9Lwu88lsQ8r6j_ZqyKJ9Qen1fslZQuDUga8xzPAOmfMVA5FbDj7pyDDqWaKNcz-0MZXSMfB70E1jn3I6NT3vf2zfMrivO62H1yFky1oJw6nsRizOdJQWji-kcd4eUuzsJCpTxjgfSb-3_R4iloZj3LQyWpwA6ZCdrMY3RhdKi4-SraP52KyHANHdsFKCBlbfub7urlhbPP3PresQdp0zFMlIEDtdksEmyGNRPswWi5WKt19jRVL3V42xrnbk8AiduEPXu9Z3loR1hLlTK0UJqDfmalbwNFdQUuM.vMoIvsxuTitxgpYqyFOd-HUaibKyytKj8LRb1DXSC6c&di	3 pack Servo motors. Rotate the ultrasonic sensor to form a scanning SONAR

				b_tag=se&keywords=servo%2Bmotor&qid=1742399137&sprefix=servo%2Bmotor%2Caps%2C99&sr=8-6&th=1	
Resistors	1		Amazon	https://www.amazon.com/ALLECIN-4W-Film-Resistor-Kit/dp/B0BTP88DZN?crid=1MIRTN9QUBUN7&dib=eyJ2IjoiMSJ9.zwvaZrTLU3vn-R-k3rdlaKAn5BeFG4MkYcisPVC0oaTQeoZQvAWYI2KgAMUOQZKiLBfh-amJmSLm83LmIEJXiyzY_tcBPai-6ZloJU2ZRdsf3spaV_F8CsCf43ujuohwS9GMmpzmcZpWnfQ236ePNe5A8F93SmPydmxb6O2wLCacTn-KBQ7QjW2MIiUUj6qWpRO8letpukgSNh1bJy_HmiVnaABIvA7tL0CHeH16CBA.cLLGuNHM3X893Zp4W1i9PPecCSAOVQnllNjrRNJx1D0&dib_tag=se&keywords=assorted%2Bresistors&qid=1742399593&sprefix=assorted%2Bresistors%2Caps%2C94&sr=8-4&th=1	Assorted Pack. We need 10K, 100K, and 100
5k Potentiometers	1	8.31	Amazon	https://www.amazon.com/Projects-Variable-Resistor-Adjustment-Potentiometer/dp/B0CX5NPJ8F?crid=1A7IPYXW4B6HA&dib=eyJ2IjoiMSJ9.M5K0fkiBq17DbxRivaUntBmCJjeLg9Rrhd1FM7Epp26Orr6MbIOPnRYKf_HkZFg-6Ligykrq-nQIHl9DcBUdX13qGmBjoPRdp2djdbfPR1pCsqP90AggnbK9N7SxnukNR44VIYVBqH9tWXZYlHDUCZnIQsTLZj_KgJZS_HbdU4TRReytYPMWL229Sk7Zc1EUA7CcM5QlIQu-nwB_Pb0cTYWErewM3aVqBqsuOIGRvu8.2QpRmtKow4u9tz53hvIxYUpW80tPzIlcIKsJ5WcAPnc&dib_tag=se&keywords=5k+potentiometers&qid=1742399724&sprefix=5k+potentiometers%2Caps%2C73&sr=8-18	50 Pack. 5K Trimming Pots

Arduino Code:

```
// Pin Definitions
const int Dpos = 18;     // Positive side of ultrasonic driver
const int Dneg = 5;      // Negative side of ultrasonic driver
const int ECHO = 15;     // Echo input pin
const int servoPin = 13; // Servo PWM control pin

// Servo Control Variables
unsigned long lastPulseTime = 0;
unsigned long lastAngleUpdateTime = 0;
int servoPos = 0;          // Current angle (0-90)
bool movingForward = true; // Sweep direction

// Ultrasonic Measurement Variables
unsigned long startTime;
const unsigned long TIMEOUT_US = 30000; // Timeout for echo wait

void setup() {
 // Set up pins
 pinMode(Dpos, OUTPUT);
 pinMode(Dneg, OUTPUT);
 pinMode(ECHO, INPUT);
 pinMode(servoPin, OUTPUT);

 digitalWrite(Dpos, LOW);
 digitalWrite(Dneg, LOW);

 // Start Serial for output
 Serial.begin(115200);
 Serial.println("Program is starting");
}

void loop() {
 unsigned long now = millis();

 // Servo Angle Update
 if (now - lastAngleUpdateTime >= 30) { // Change angle every 30ms
   lastAngleUpdateTime = now;

   if (movingForward) {
     servoPos++;
```

```
    if (servoPos >= 90) {
      movingForward = false;
    }
  } else {
    servoPos--;
    if (servoPos <= 0) {
      movingForward = true;
    }
  }
}

//Manual Servo Pulse
if (now - lastPulseTime >= 20) { // Send servo pulse every 20ms
  lastPulseTime = now;
  sendManualServoPulse(servoPos);
}

// Ultrasonic Stuff
delay(1);
send_pulse();
measure_echo();
delay(1);
}


// Send 8 cycles of 40kHz ultrasonic signal
void send_pulse() {
startTime = micros();
for (int i = 0; i < 8; i++) {
  digitalWrite(Dpos, HIGH);
  digitalWrite(Dneg, LOW);
  delayMicroseconds(13); // Half period
  digitalWrite(Dpos, LOW);
  digitalWrite(Dneg, HIGH);
  delayMicroseconds(13);
}
}

// Measure echo width and output distance
void measure_echo() {
while (digitalRead(ECHO) == LOW) {
  if (micros() - startTime > TIMEOUT_US) {
```

```
    return; // Timeout if no echo
   }
 }
  unsigned long endTime = micros();
 unsigned long width = endTime - startTime - 205; // Account for ESP32 processing
delays

 // Print servo position and distance
 if (width > 80 && width < 1500) { // Valid echo window
   Serial.print(servoPos);
   Serial.print(",");
   Serial.println(width);
 } else {
   Serial.print(servoPos);
   Serial.print(",");
   Serial.println(1000000000); // Invalid signal
 }
}

// Generate a manual servo pulse without using PWM.
// Had to do it this way to avoid timing issues
void sendManualServoPulse(int angle) {
 int pulseWidth = map(angle, 0, 180, 1000, 2000); // Map 0-180° to 1-2ms pulse
 digitalWrite(servoPin, HIGH);
 delayMicroseconds(pulseWidth);
 digitalWrite(servoPin, LOW);
}
```

Processing Code:

```
import processing.serial.*;

Serial myPort;
String[] incomingData;

float time_delay;
int servo_pos;

float angle = 0;
boolean sweeping = false;
int sweep_x = 0;

float distance = 0;

float increment = PI/300;

int servo_angle;

void setup() {
  size(1000, 800);
  smooth();
  ellipseMode(RADIUS);

  //Setup Serial Port
  String portName = Serial.list()[1]; // Change index as needed
  myPort = new Serial(this, portName, 115200);
  myPort.bufferUntil('\n'); // Wait for full line. Will also
automatically call SerialEvent when it sees data
}
```

```
void draw() {
  fill(98, 245, 31);
  // simulating motion blur and slow fade of the moving line
  noStroke();
  fill(0, 4);
  rect(0, 0, width, height-height*0.065);

  fill(98, 245, 31); // green color

  // Buttons
  drawButton(50, height - 60, 100, 40, "Start", !sweeping);
  drawButton(width - 150, height - 60, 100, 40, "Stop", sweeping);

  drawUI();
  drawSweep();
  drawObject();
}




//Function that handles incoming Serial input
void serialEvent(Serial myPort) {
  String input = myPort.readStringUntil('\n');
  if (input != null) {
    input = trim(input);
    incomingData = split(input, ',');
```

```
    if (incomingData.length == 2) { // make sure you have exactly 2
values
      servo_angle = int(incomingData[0]);
      time_delay = int(incomingData[1]);
    }
    println(time_delay);
    distance = ((time_delay/ 1000000) / 2) * 34000;
    angle = (servo_angle - 45) * PI / 180 ;
    println(angle);
   }
  }

void drawButton(float x, float y, float w, float h, String label,
boolean active) {
  fill(active ? 0 : 50);
  stroke(0, 255, 0);
  rect(x, y, w, h);
  fill(255);
  textAlign(CENTER, CENTER);
  text(label, x + w/2, y + h/2);
}

void mousePressed() {
  if (overButton(50, height - 60, 100, 40)) {
    sweeping = true;
  } else if (overButton(width - 150, height - 60, 100, 40)) {
    sweeping = false;
  }
}
```

```
boolean overButton(float x, float y, float w, float h) {
  return mouseX > x && mouseX < x + w && mouseY > y && mouseY < y +
h;
}

void drawUI() {
// Radar image
  stroke(98, 245, 31);
  noFill();
  line(width/2, height, 0,0);
  line(width/2, height, width,0);
  arc(width/2, height, 500, 500, -PI/2-PI/5.65, -PI/3.1);
  arc(width/2, height, 700, 700, -PI/2-PI/5.65, -PI/3.1);
  arc(width/2, height, 300, 300, -PI/2-PI/5.65, -PI/3.1);
  arc(width/2, height, 100, 100, -PI/2-PI/5.65, -PI/3.1);
}

void drawSweep() {
 // Radar sweep
  stroke(0, 255, 0);
  noFill();
  sweep_x = int(sin(angle)*1000) + (width/2);
  if (sweeping) {
    line(width/2, height, sweep_x,0);

    if(angle < -PI/4) {
      increment = PI/300;
    }
```

```
    if(angle > PI/4) {
      increment = -PI/300;
    }

    angle+= increment;
  }
}

void drawObject() {
  strokeWeight(4);
  stroke(255, 10, 10); // red color

  line(int(sin(angle)*distance*50) + (width/2),
height-cos(angle)*distance*50, sweep_x,0);
}
```