Ultrasonic Sensor-Based Radar System

Jean A. Miranda Vargas
Department of Electrical and Computer Engineering
Universidad Ana G. Méndez
Caguas, Puerto Rico
jmiranda166@email.uagm.edu

Jean Carlos Gonzalez Lopez
Department of Electrical and Computer Engineering
Universidad Ana G. Méndez
Caguas, Puerto Rico
jgonzalez1207@email.uagm.edu

***Abstract*—This report presents the design, implementation, and testing of an ultrasonic sensor-based radar system as the final project for the microprocessor class and laboratory. The system leverages an ultrasonic sensor and a micro servo motor to emulate radar functionalities by scanning and measuring the distance to objects. The project implementation follows a reference Arduino design [3] and includes prototyping using Tinkercad. This paper outlines the experiment setup, step-by-step assembly procedures, and the tests conducted to verify system performance. Results indicate that the system accurately measures distances and provides a reliable foundation for further enhancements.**

Keywords—Arduino, Radar, Tinkercad, Microprocessor, Servo Motor, Ultrasonic Sensor, Prototyping.

I. Introduction

The aim of this project is to develop a functional radar system employing an ultrasonic sensor for distance measurement and a micro servo motor for scanning in 180˚. Radar systems in modern day are widely used in various applications, including aviation, automotive, robotics and more. Our project introduces a simplified model using readily available components to emulate radar operation, offering practical insights into sensor integration and circuit prototyping. The project is primarily based on the Arduino project detailed in [3].

A. But why this project?

This project was chosen because it was too intuitive to ignore. Essentially, we'll be making our own homemade radar, and although it's miniature, its function is the same. It gives us a perfect idea of what we can expect from a military radar.

B. Project difficulty

Building the circuit isn't that complicated. The only thing that would be tedious and somewhat confusing is getting the servomotor and ultrasonic sensor to work together. Also, connecting the ultrasonic sensor can be annoying considering the different jumpers it requires. However, the most difficult part is the code itself. Getting the parts to work the way you want them to is undoubtedly the part that requires the most time to get the project working. Writing code requires time, patience, and knowledge to write it properly and ensure it works properly.

It can be tedious and annoying to do all of this, but doing work of this caliber requires time, patience, and concentration. Therefore, it's advisable to start every project early to allow enough time to have everything ready and be able to complete the work correctly.

II. System Description, Methodology and Usability

A. Hardware Components

1. Ultrasonic Sensor

The ultrasonic sensor is used to measure distances by emitting sound pulses and calculating the time delay of the reflected waves.

2. Micro-Servo Motor

The micro-servo motor rotates the sensor across an angle of 180˚, allowing the system to cover a range of directions.

3. Breadboard and Mounting Accessories

Breadboard facilitates easy prototyping, while zip ties and tape are used to secure the sensor to the servomotor and the assembly to the breadboard.

4. Microcontroller (Arduino)

The Arduino board is used as the main microcontroller to drive the servomotor and to process the ultrasonic sensor's readings.

B. Software and Control

The system is programmed using the Arduino Integrated Development Environment (IDE). The code first assesses the functionality of the servo motor to ensure accurate positional control. Thereafter, the ultrasonic sensor's readings are processed while the sensor is in motion. The Arduino code (included in this report) manages the PWM signals for motor control and the calculations needed for distance measurement. For our project we also used another software called Processing IDE to display the radar interface and the objects that it detects in real time.

C. Project Usability

An ultrasonic sensor already has many uses, such as detecting the depth of a container and assisting in building/home automation. In this case, it is being used to detect an object in the sensor's sights and highlight that object in the app. Such a circuit could be useful for detecting obstacles in a location one cannot see. Overall, we are simulating a military radar, as it uses methods and technology incredibly like what we are using.

You see, the ultrasonic sensor works by emitting ultrasonic sound waves at a frequency above the audible range. [1] These waves bounce off nearby objects and return to the sensor, which calculates the distance of the echo, which is sent to the Arduino. The Arduino calculates the distance with the formula d = (340 m/s * t) / 2, t being what the sensor calculated. As for a military radar, it consists of an electronic system that contains a rotating antenna that emits a beam of ultra-high frequency electromagnetic waves and when they

collide with an object, they are reflected and when captured by the same antenna of the system, they are processed to appear in the form of light pulses on the air traffic controller's screen. [2]

As you can see, the two methods are almost identical, except the military uses technology of a higher-caliber and higher-quality. However, at their core, they are methods that detect objects and highlight those objects so those in charge see it and calculate the distance between them. This leads us to believe that our project's circuit is extremely important, with its development highly needed, considering that the military uses a method and technology that is too similar to be a coincidence.

III. Circuit Design

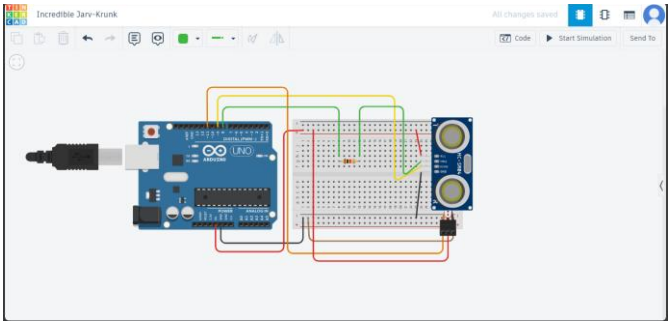

Figure 1: Simulation of our Radar Circuit.

As you can see, the Ultrasonic Sensor is placed on top of the Micro-Servo Motor. How it's been placed like this is decided later on. What's important is that, in the simulation, both components work correctly. For example, in the simulation, without placing the resistor, we were warned that the TRIG connection we placed would damage the Arduino without placing a resistor in it.

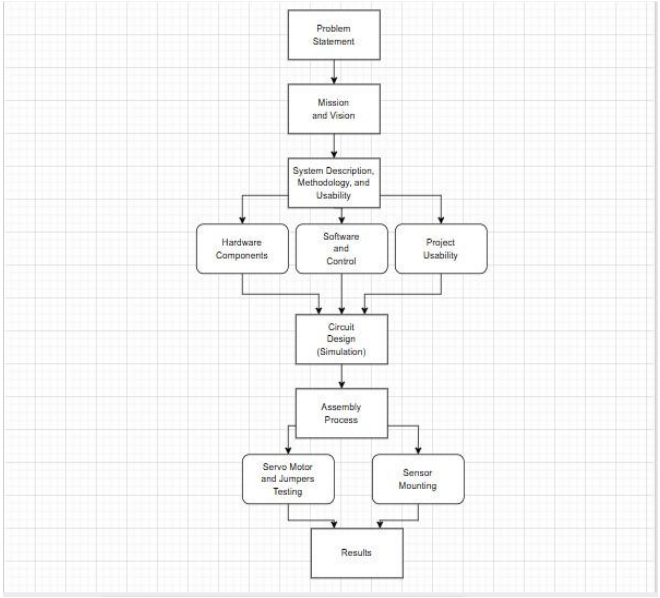

Figure 2: Project Flowchart.

```
#include<Servo.h>

#define trigPin 8
#define echoPin 9

long duration;
int distance ;

Servo myservo;

int calculateDistance()
{
  digitalWrite(trigPin,LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin,HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin,LOW);
  duration = pulseIn(echoPin, HIGH);
  distance = duration*0.034/2;
  return distance;
}

void setup()
{
  pinMode(trigPin , OUTPUT);
  pinMode(echoPin, INPUT);
  myservo.attach(11);
  Serial.begin(9600);
}

void loop()
{
 int i ;
 for (i=15; i<=165; i++)
 {
  myservo.write(i);
  delay(15);
  calculateDistance();
  Serial.print(i);
  Serial.print(",");
  Serial.print(distance);
  Serial.print(".");
 }
 for(i=165; i>=15; i--)
 {
  myservo.write(i);
  delay(15);
  calculateDistance();
  Serial.print(i);
  Serial.print(",");
  Serial.print(distance);
  Serial.print(".");

 }
}
```

Figure 3: Code for the Simulation.

IV. Assembly Process

A. Servo Motor and Jumper Testing

We started off our project by testing out the micro-servo motor to see that it worked properly and that the motor spun the way we needed it to. This is because in Arduino packages, there are times when a few components come damaged, particularly jumpers and servo motors. As such, we need to test the servo motors and jumpers to see if they work.

Specifically for the jumpers, we used a multimeter, A multimeter is an instrument designed to measure electric

current, voltage, and resistance, typically over several ranges of values.

Figure 4: The Multimeter.

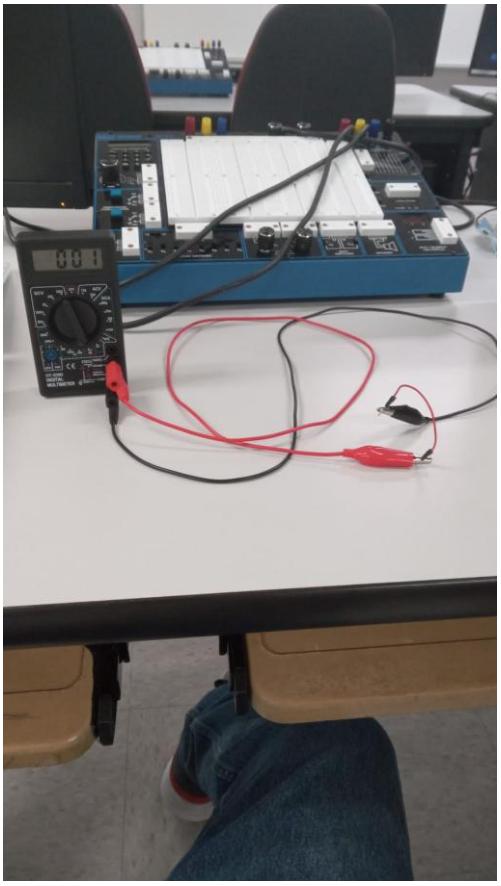

Figure 5: Using the Multimeter to test the jumpers.

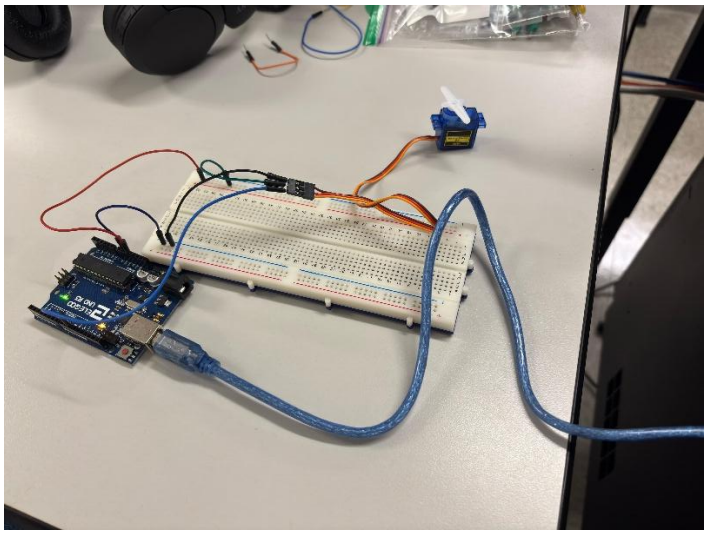

Figure 6: Micro-Servo Motor Testing.

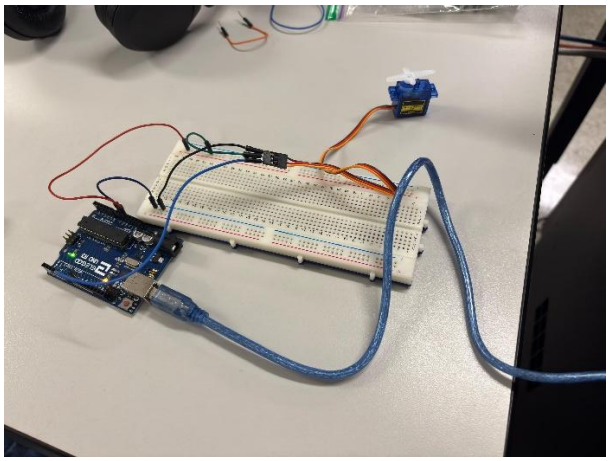

Figure 7: Servo Motor Spinning Accurately.

```
#include <Servo.h>

Servo myServo;

void setup() {
  myServo.attach(9);  // signal pin
on D9
}

void loop() {
  // Fast sweep from 0 to 180
  for (int pos = 0; pos <= 180; pos
+= 5) {
    myServo.write(pos);
    delay(5);  // faster motion
  }

  // Fast sweep from 180 to 0
  for (int pos = 180; pos >= 0; pos
-= 5) {
    myServo.write(pos);
    delay(5);
  }
}
```

Figure 8: Code Used to Test Micro-Servo Motor.

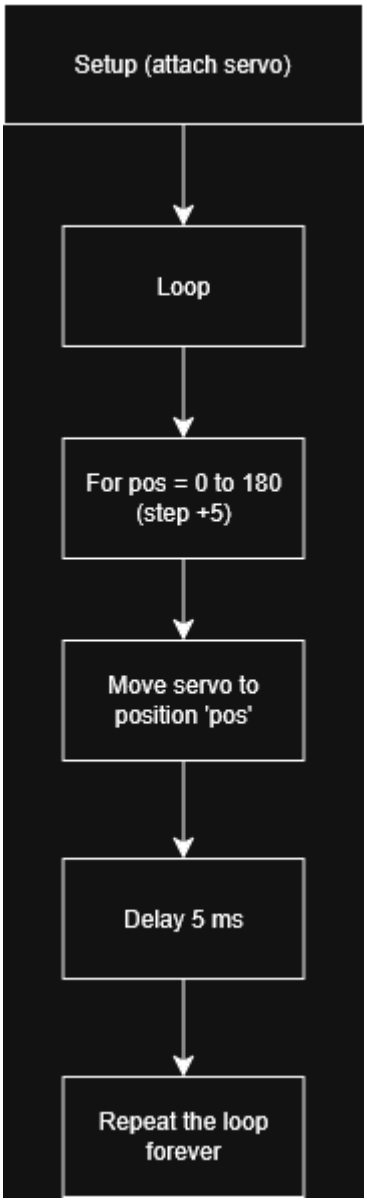

Figure 9: Servo Motor Test Flowchart.

The servo motor was programmed to rotate through a series of angles. The motor spun accurately, and we determined that it was working properly after we finished the testing.

B. Sensor Mounting

After we finished evaluating the servo motor, we needed to find a way to mount the ultrasonic sensor on top of the motor in a secure manner. We opted to secure it to the breadboard by using tape on the base of the servo motor and use zip ties to secure both the sensor and the motor together. This way, they can work in tandem and stay in place. After the sensor was mounted, we did tests with the motor to see that the sensor was in place securely.

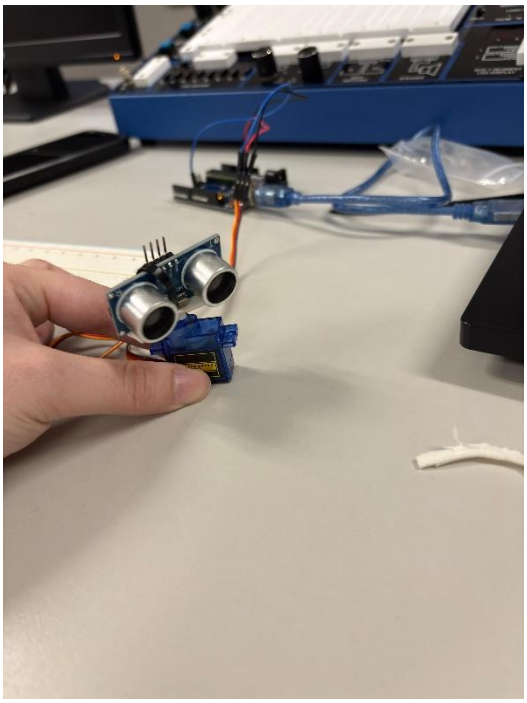

Figure 10: Ultrasonic Sensor Mounted on Micro-Servo Motor.

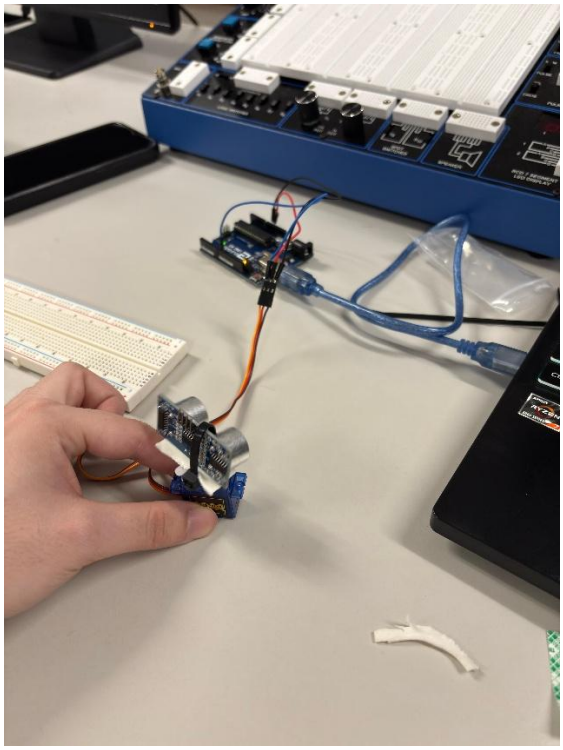

Figure 11: Servo Motor Spinning.

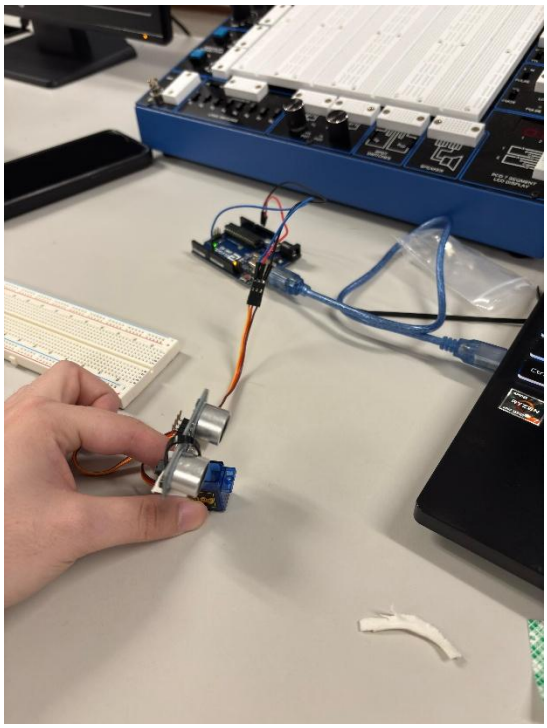

Figure 12: Servo Motor Spinning Test with Sensor.

We used the same code as shown in Figure 7 to test that the ultrasonic sensor was mounted securely and that it would spin without any problems, we concluded that it indeed was working properly and that it was secured in place.

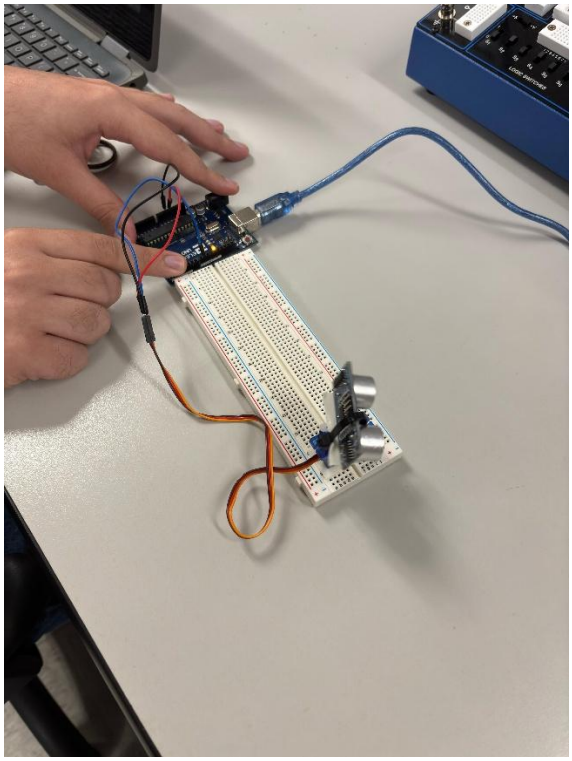

Figure 13: Ultrasonic Sensor Attached to Servomotor and Servomotor Mounted on Breadboard.

Afterwards we then mounted the micro servo motor on the breadboard by using tape to make sure it stayed in place, we proceeded to test if our current system worked without any problems and the servomotor along with the sensor were spinning in place without any type of issues, with this we were certain that we were ready to start wiring the sensor to the Arduino and start the coding process for the radar system.

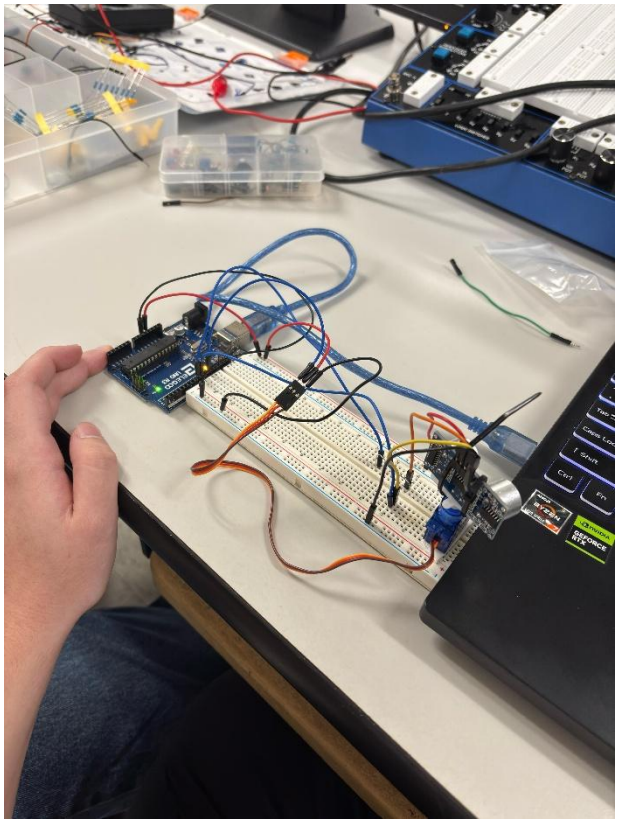

Figure 14: Circuit Fully Wired and Implemented.

We wired the entire circuit and fully integrated all of the components.

- **Jumpers:** The cables themselves, used to connect all the components together. They are the connections that'll allow the project to work.
- **Breadboard:** The backbone of the circuit. It is what allowed us to connect all the other components and build the circuit literally on it.
- **Arduino Uno:** The microprocessor that will enact and run the code we've written. Through the code, we will have the circuit work the way it'll be designed for.
- **Power Rails:** We connected a cable from the Arduino 5V **VCC** to the breadboards positive rail. A cable was also connected to the Arduinos **GND** and was connected to the negative rail on the breadboard.
- **Ultrasonic Sensor:** The ultrasonic sensor was wired in the following manner. The **VCC** on the sensor was connected to the breadboard's positive rail with a cable, the **GND** was connected to the negative rail on the breadboard. The **TRIG** from the sensor was connected with a cable to the Arduino's digital pin **D2** through a 1KΩ resistor to limit current and to protect it. Finally, the **ECHO** was linked to a cable to Arduino's digital pin **D3**.
- **Micro Servo Motor (SG90):** The micro servo motor was connected in the following manner.

The **VCC (red)** was connected to the breadboards positive rail, the **GND (brown)** was connected to the breadboards negative rail, finally the **SIGNAL (orange)** was connected to the Arduino's digital pin **D4.**

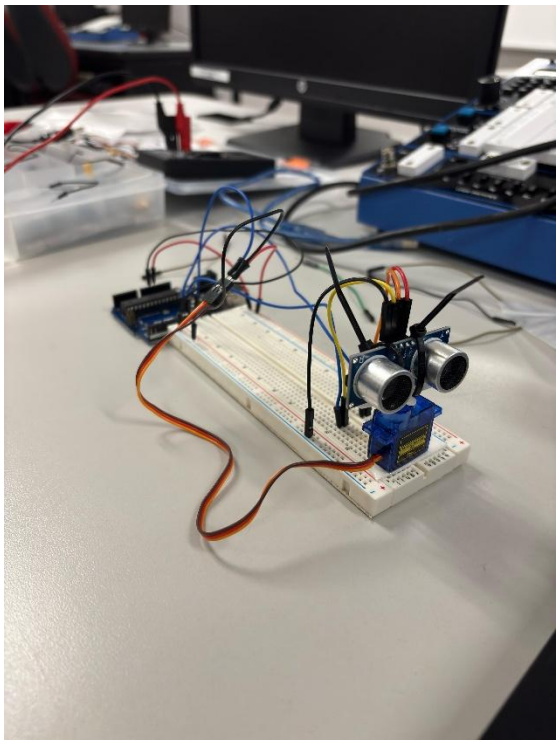

Figure 15: Radar Circuit Wired Up.

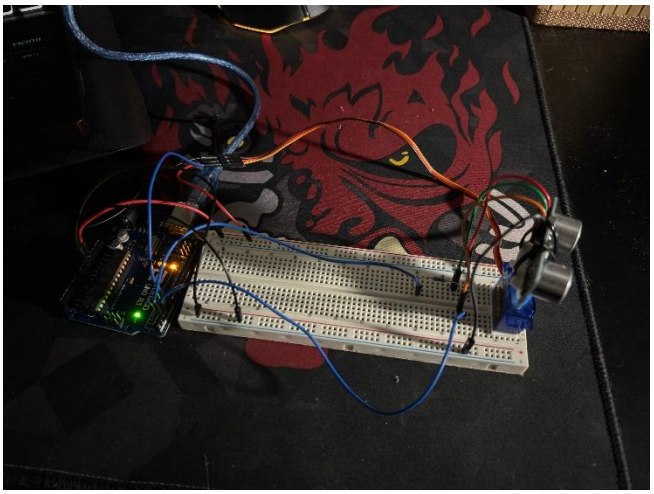

Figure 16: The Circuits Wiring.

After we finished wiring the entire circuit, we then proceeded to write the code for the Arduino to work with the ultrasonic sensor and to work in conjunction with the servo motor.

```
/**
 * Ultrasonic Radar with Servo Sweep
 *
 * This sketch controls an HC-SR04 ultrasonic sensor
mounted on a servo motor.
 * It performs a continuous 0°→180°→0° sweep, measures
distance at each angle,
 * and sends "angle,distance." strings over Serial for
visualization.
 *
 * Connections:
 *   - trigPin (D2)   → HC-SR04 Trig (with 1K Ω series
resistor)
 *   - echoPin (D3)   → HC-SR04 Echo
 *   - servo signal   → Servo signal line on D4
 *   - 5 V and GND    → common power/ground rails
 */

#include <Servo.h>            // Include Servo library
for PWM control

// Pin definitions
#define trigPin 2              // Ultrasonic sensor
trigger pin
#define echoPin 3              // Ultrasonic sensor
echo pin

// Variables for distance calculation
long duration;                 // Time (µs) for echo
return
int distance;                  // Calculated distance
in cm

Servo myservo;                 // Servo object for
controlling the sweep

/**
 * calculateDistance()
 *
 * Sends a 10 µs pulse on trigPin, measures the time
until echoPin goes HIGH,
 * converts that duration into a distance (cm), and
returns it.
 */
int calculateDistance() {
  // Ensure trigger is LOW for at least 2 µs
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  // Send a 10 µs HIGH pulse to trigger ultrasonic
burst
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read the duration of the HIGH pulse on echoPin
  duration = pulseIn(echoPin, HIGH);

  // Convert time (µs) to distance (cm): sound speed ≈
0.034 cm/µs, divide by 2 for round trip
  distance = duration * 0.034 / 2;
  return distance;
}

void setup() {
  // Configure sensor pins
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  // Attach servo to pin 4
  myservo.attach(4);

  // Begin serial output for Processing visualization
(9600 baud)
  Serial.begin(9600);
}

void loop() {
  // --- Sweep from 0° to 180° ---
  for (int pos = 0; pos <= 180; pos += 5) {
    myservo.write(pos);        // Move servo to current
angle
    delay(30);                 // Short delay for
smooth movement

    calculateDistance();       // Update `distance`

    // Send formatted data: "angle,distance."
    Serial.print(pos);
    Serial.print(",");
    Serial.print(distance);
    Serial.print(".");
  }

  // --- Sweep from 180° back to 0° ---
  for (int pos = 180; pos >= 0; pos -= 5) {
    myservo.write(pos);
    delay(30);

    calculateDistance();

    Serial.print(pos);
    Serial.print(",");
    Serial.print(distance);
    Serial.print(".");
  }
}
```

Figure 17: Code for Arduino, Ultrasonic Sensor, and Micro Servo Motor.

Once the Arduino code was completed, we proceeded to edit the Processing IDE code in the radar reference project [3]. This code and software will allow us to display a radar matrix on the computer to physically view the radar in action.

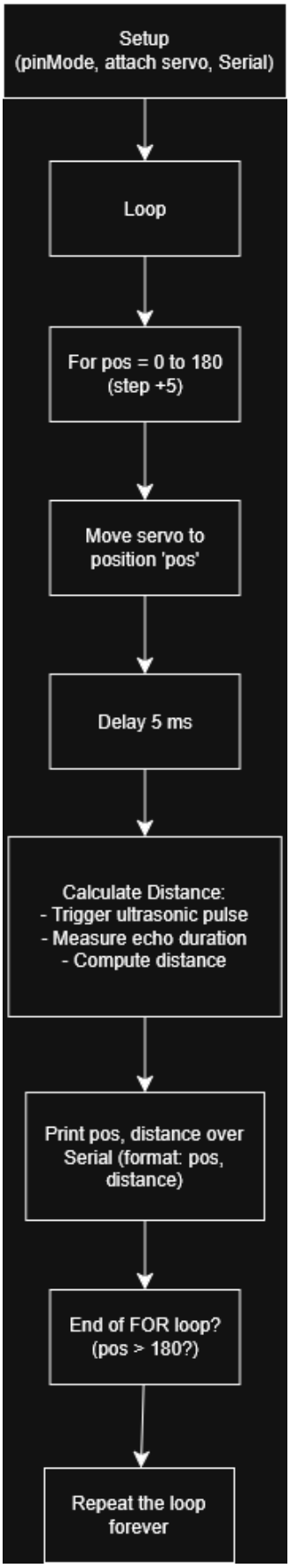

Figure 18: Flowchart for Arduino Code.

```
import processing.serial.*;              // import Serial library

Serial myPort;                           // Serial port object

// Variables for incoming data
String angle = "";                       // raw angle substring
String distance = "";                    // raw distance substring
String data = "";                        // full "angle,distance" string
float pixsDistance;                      // scaled pixel distance for drawing
int iAngle, iDistance;                   // parsed integer angle and distance
int index1 = 0;                          // index of comma separator

void setup() {
  fullScreen();                          // occupy entire display at native
resolution
  smooth();                              // enable anti-aliasing for shapes
  myPort = new Serial(this, "COM9", 9600);// open Serial port COM9 at 9600 baud
  myPort.bufferUntil('.');               // buffer characters until '.' then fire
serialEvent()
}

void draw() {
  // Fade the background slightly to create motion blur effect
  noStroke();
  fill(0, 4);
  rect(0, 0, width, height - height * 0.065);

  // Draw radar grid (arcs + spokes)
  strokeWeight(2);
  stroke(98, 245, 31);
  noFill();
  drawRadar();

  // Draw the sweeping line at current angle
  drawLine();
  // Draw detected object blip if within range
  drawObject();

  // Draw the bottom overlay text panel
  drawText();
}

/**
 * serialEvent()
 *
 * Called automatically when Serial bufferUntil('.') condition is met.
 * Reads the incoming string (e.g. "45,20."), strips the '.', splits at the comma,
 * and converts to integer values iAngle and iDistance.
 */
void serialEvent(Serial myPort) {
  data = myPort.readStringUntil('.');
  if (data != null && data.length() > 1) {
    data = data.substring(0, data.length() - 1);  // remove trailing '.'
    index1 = data.indexOf(",");                   // find comma position
    angle = data.substring(0, index1);            // extract angle substring
    distance = data.substring(index1 + 1);        // extract distance substring
    iAngle = int(angle);                          // parse to integer
    iDistance = int(distance);
  }
}

/**
 * drawRadar()
 *
 * Renders four concentric half-circles (PI*TWO_PI) scaled to the window width,
 * and radial spokes every 30° from 0° to 180°.
 */
void drawRadar() {
  pushMatrix();
  translate(width/2, height - height*0.074); // move origin to bottom-center
  // concentric arcs
  arc(0, 0, width*0.94, width*0.94, PI, TWO_PI);
  arc(0, 0, width*0.73, width*0.73, PI, TWO_PI);
  arc(0, 0, width*0.52, width*0.52, PI, TWO_PI);
  arc(0, 0, width*0.31, width*0.31, PI, TWO_PI);
  // horizontal baseline
  line(-width/2, 0, width/2, 0);
  // angled spokes at 30° increments
  for (int a = 30; a <= 150; a += 30) {
    line(0, 0,
         -width/2 * cos(radians(a)),
         -width/2 * sin(radians(a)));
  }
  popMatrix();
}

/**
 * drawLine()
 *
 * Draws the current scanning line in bright green, length scaled by window height.
 */
void drawLine() {
  pushMatrix();
  strokeWeight(9);
  stroke(30, 250, 60);
  translate(width/2, height - height*0.074);
  float lineLen = (height - height*0.12);
  line(0, 0,
       lineLen * cos(radians(iAngle)),
       -lineLen * sin(radians(iAngle)));
  popMatrix();
}

/**
 * drawObject()
 *
 * If an object is within 40 cm, draw a red blip at the corresponding polar
coordinate.
 */
void drawObject() {
  if (iDistance < 40) {
    pushMatrix();
    translate(width/2, height - height*0.074);
    strokeWeight(9);
    stroke(255, 10, 10);
    // convert cm to pixels relative to chart size
    pixsDistance = iDistance * ((height - height*0.1666) * 0.025);
    line(pixsDistance * cos(radians(iAngle)), -pixsDistance * sin(radians(iAngle)),
         (width - width*0.505) * cos(radians(iAngle)),
         -(width - width*0.505) * sin(radians(iAngle)));
    popMatrix();
  }
}

/**
 * drawText()
 *
 * Draws a semi-opaque black panel at the bottom for text:
 *  - Static range markers (10–40 cm).
 *  - Dynamic labels: N_Tech title, current angle, and distance/status.
 */
void drawText() {
  // panel background
  fill(0);
  noStroke();
  rect(0, height - height*0.08, width, height*0.08);

  fill(98, 245, 31);
  textSize(height * 0.02);
  // static range markers along bottom
  text("10cm", width * 0.60, height * 0.92);
  text("20cm", width * 0.69, height * 0.92);
  text("30cm", width * 0.78, height * 0.92);
  text("40cm", width * 0.87, height * 0.92);

  // dynamic info
  textSize(height * 0.03);
  text("N_Tech", width * 0.10, height * 0.97);
  text("Angle: " + iAngle + "°", width * 0.40, height * 0.97);

  String distStr = iDistance < 40
    ? "Distance: " + iDistance + " cm"
    : "Distance: Out of Range";
  text(distStr, width * 0.65, height * 0.97);
}
```

Figure 19: Processing IDE Code.

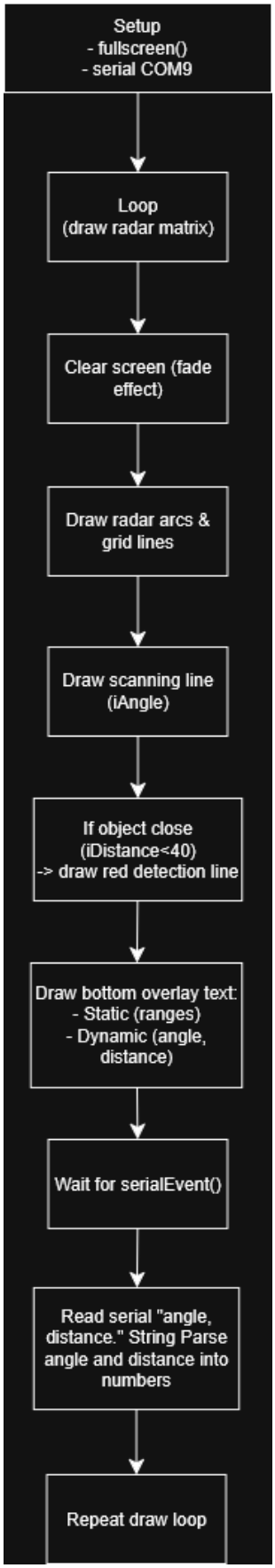

Figure 20: Processing IDE Flowchart.

V. RESULTS

After the circuit and the codes were done, we deployed the Arduino sketch and Processing visualization together to verify full system functionality. Upon powering the system, the servo began its continuous 0˚ to 180˚ sweep. As the servo rotated, the ultrasonic sensor emitted pulses and accurately measured distances, with each reading transmitted via Serial and parsed by the Processing sketch in real time.

On the Processing IDE side, the radar display initialized in full-screen mode and immediately rendered the concentric arcs and spokes at the bottom-center of the window. The bright green sweep line tracked the servo's motion smoothly, and any objects placed withing the sensor's 40cm range produced distinct red blips at the correct polar coordinates.

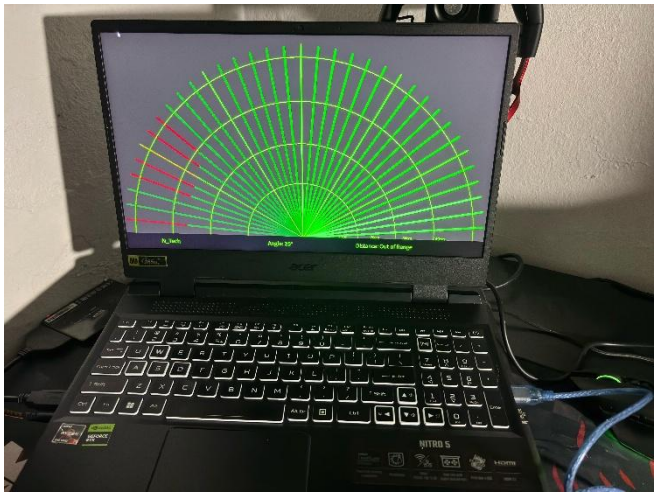

Figure 21: Processing IDE Displaying Radar.

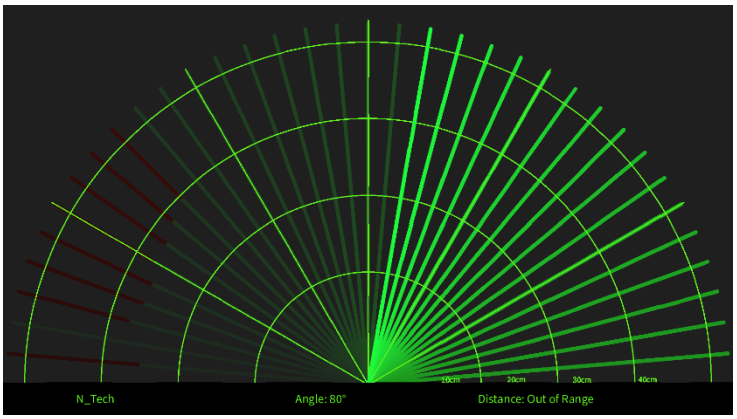

Figure 22: Radar Display.

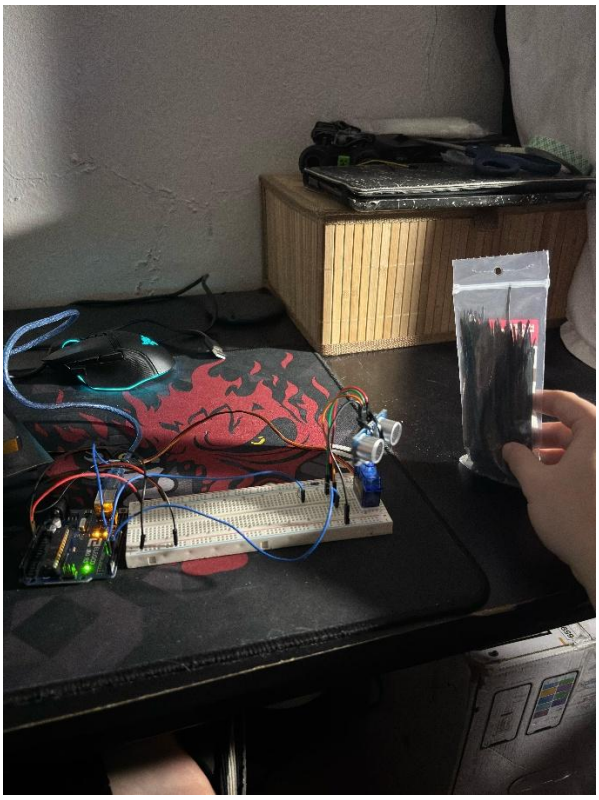

Figure 23: Object Placed in Front of Ultrasonic Sensor.

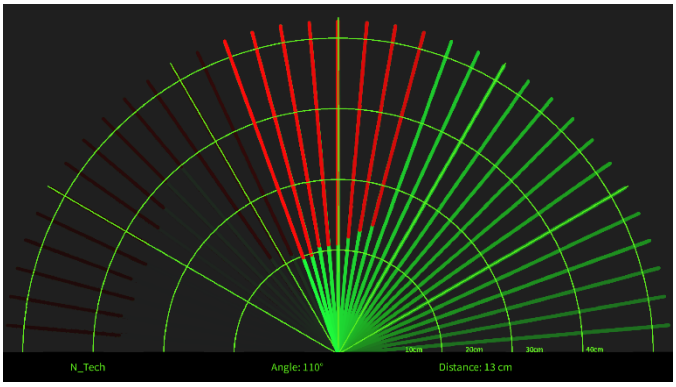

Figure 24: Radar Displaying Object Detected.

After testing various objects, we were able to conclude that our system worked properly without any issues. In fact, it worked almost exactly how one would think a military radar would function and be displayed.

VI. Conclusion

In conclusion, the ultrasonic radar system developed in this project successfully demonstrated how low-cost components like the HC-SR04 ultrasonic sensor and SG90 servo motor can be integrated with an Arduino Uno to emulate fundamental radar functionality. From hardware assembly and circuit design to software control and graphical visualization, each stage of the project operated reliably and in harmony.

The Arduino handled servo control and distance measurement with minimal delay, while Processing IDE provided a responsive and visually intuitive display. The system was capable of continuously scanning its surroundings and accurately identifying objects withing a 40cm radius. The radar interface rendered object detections with smooth animations in real-time updates, reflecting the current angle and measured distance dynamically.

Our results showed minimal error, a consistent sweep pattern from the servo, high graphical fidelity in the Processing IDE output. Moreover, by designing and simulating the circuit in Tinkercad, we ensured the robustness of our design before physical implementation, reducing trial-and-error during prototyping.

Overall, this project not only fulfilled our expectations but also deepened our understanding of embedded systems, serial communication, and real-time visualization. Future improvements could include extended range capability, adjustable sweep patterns, or wireless data transmission. This project served as a strong practical demonstration of how hardware and software can be integrated to simulate complex sensor systems like a radar.

VII. References

[1] Bay Alarm Company, “How Do Motion Sensors Work? A Guide,” Bay Alarm Blog. [Online]. Available: https://www-bayalarm-com.translate.goog/blog/how-do-motion-sensors-work-a-guide/?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=sge#:~:text=Active%20ultrasonic%20sensors%20emit%20ultrasonic,pulse%20and%20receives%20the%20echo. [Accessed: April. 2025].

[2] Servicios a la Navegación en el Espacio Aéreo Mexicano, “Sistema de Radar,” Gobierno de México. [Online]. Available: https://www.gob.mx/seneam/articulos/sistema-de-radar. [Accessed: April. 2025].

[3] N. Mishra, “Ultrasonic Radar with Arduino,” Arduino Project Hub. [Online]. Available: https://projecthub.arduino.cc/nimishac/ultrasonic-radar-with-arduino-19baa3. [Accessed: April. 2025].