VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Electronics Engineering (SENSE)

BECE204L – MICROPROCESSORS AND MICROCONTROLLERS PROJECT

TOPIC- ULTRASONIC RADAR FOR AIRPORT TRAFFIC CONTROL

Submitted By

22BLC1146 – SOHAM RAJENDRA BHORE

22BLC1171 - KRISH SAHU

22BLC1290 – MEGHANK KISHAN

Submitted To

Dr. Abraham Sudharson Ponraj

Abstract

Airports are complex environments that demand robust, efficient, and reliable surveillance systems to ensure the safety and security of both aircraft and personnel. With thousands of flights operating daily, maintaining situational awareness in such dynamic settings is paramount. Traditional radar systems used in airports are often sophisticated, but they come with high costs and maintenance requirements. In response to these challenges, this paper presents the development of a low-cost, scalable Ultrasonic Radar system designed to enhance airport surveillance.

The system utilizes ultrasonic sensors to detect objects within a specified range, offering real-time data visualization through Processing software. By continuously scanning the environment using a rotating ultrasonic sensor, the system can monitor areas and detect potential obstacles or unauthorized objects in real-time. This radar setup provides angular and distance measurements of detected objects, which are then graphically represented on the screen for easy interpretation by airport personnel.

The primary objective of this project is to create a simple, cost-effective, and energy-efficient radar system that can be easily implemented and maintained in critical areas of an airport. This paper also explores the scalability and future enhancements that can further improve its performance, such as extending its detection range or incorporating additional sensors. This system, although basic in its current form, has the potential to act as a supplementary surveillance tool for airport operations, contributing to the early detection and prevention of security risks in sensitive zones, such as taxiways, hangars, or near aircraft gates. The proposed system could help reduce incidents by providing timely alerts about any obstacles, ensuring smoother airport operations and enhanced safety.

Introduction

Airports are high-traffic environments where safety and efficient operations are crucial. While traditional radar systems are used for tracking aircraft in airspaces, they can be costly and complex for ground-level monitoring in areas like taxiways, hangars, or restricted zones. To address these challenges, this project introduces an ‘Ultrasonic Radar’ system designed for localized surveillance within airports.

This system uses an ultrasonic sensor to detect objects by emitting sound waves and calculating the time delay of the returning echo. A servo motor rotates the sensor to continuously scan the surrounding area, and an Arduino microcontroller processes the data. The detected objects are displayed in real-time using Processing software, providing a visual representation of both the angle and distance of obstacles within a 40 cm range.

The Ultrasonic Radar offers a cost-effective, easy-to-implement solution for monitoring restricted airport areas, enhancing safety by providing early warning of potential obstacles. This project details the system's hardware and software components, demonstrating its potential as a supplemental tool for improving airport ground-level surveillance.

Literature Review

The development of effective surveillance systems in airports has long been a subject of interest in both academic and practical contexts. This literature review explores existing research and technologies relevant to the Ultrasonic Radar system, focusing on advancements in radar technology, ultrasonic sensing, and real-time data visualization.

1. Radar Systems in Aviation

Traditional radar systems, such as Primary Surveillance Radar (PSR) and Secondary Surveillance Radar (SSR), have been extensively used for air traffic control and airport security. PSR operates by emitting electromagnetic waves and detecting the reflected signals from aircraft, while SSR relies on transponders on aircraft to provide additional information. These systems are crucial for monitoring aircraft positions and ensuring safe air traffic management [1]. However, their high cost and complexity often limit their application to larger-scale operations rather than localized monitoring within airport grounds.

2. Ultrasonic Sensing Technology

Ultrasonic sensors, which use high-frequency sound waves to measure distance, have been explored for various applications, including object detection and obstacle avoidance. According to the study by Lee et al. (2016), ultrasonic sensors offer a cost-effective solution for short-range distance measurement and are widely used in robotics and automotive applications [2]. The principles of ultrasonic sensing are well-documented, with key advantages including simplicity, affordability, and effectiveness in measuring distances within a limited range. These characteristics make ultrasonic sensors suitable for developing a radar system for airport environments.

3. Integration of Arduino and Ultrasonic Sensors

The integration of Arduino microcontrollers with ultrasonic sensors has been extensively researched for various applications. Arduino platforms provide a flexible and programmable environment for controlling sensors and processing data. In a study by Pishvaei et al. (2018), Arduino-based systems were used for implementing real-time distance measurement and obstacle detection [3]. The study highlights the benefits of using Arduino for its ease of use, scalability, and ability to interface with a wide range of sensors, making it an ideal choice for the proposed Ultrasonic Radar system.

4. Real-Time Data Visualization

Visualizing sensor data in real-time is crucial for effective monitoring and decision-making. Processing software, known for its simplicity and powerful visualization capabilities, is often employed in conjunction with Arduino and other microcontrollers. The work by McCormick et al. (2017) demonstrates the use of Processing for creating interactive data visualizations and real-time graphical displays [4]. The study emphasizes how Processing can be used to create intuitive and informative visualizations of sensor data, which is essential for the radar system's GUI.

5. Applications of Ultrasonic Radar Systems

Several studies have explored the applications of ultrasonic radar systems beyond traditional use cases. For instance, Wang et al. (2019) investigated the use of ultrasonic radar for indoor navigation and object detection [5]. The research highlights the effectiveness of ultrasonic systems in environments where traditional radar may not be practical, such as within enclosed spaces or for specific monitoring tasks. These findings support the application of ultrasonic radar technology in airport settings, where localized and precise monitoring is required.

In summary, the literature indicates that ultrasonic sensors combined with Arduino and real-time data visualization software offer a viable solution for developing cost-effective and efficient radar systems. This background provides a solid foundation for the proposed Ultrasonic Radar system, addressing the need for localized surveillance in airport environments while leveraging established technologies for distance measurement and data visualization.

System Design and Methodology

1. Hardware Design

The system comprises the following key components:

- **Ultrasonic Sensor (HC-SR04)**: This sensor emits ultrasonic waves and detects their echoes to measure distance.
- **Arduino Uno**: The microcontroller processes data from the ultrasonic sensor and controls the servo motor.
- **Servo Motor**: Responsible for rotating the sensor, allowing the system to sweep across a range of angles.
- **Processing Software**: Visualizes the radar data by drawing objects based on their angular position and distance.

The radar system scans across a 150° range, capturing objects within its field of view and providing real-time feedback on object position and distance.

a. Arduino

The architecture of Arduino boards typically follows the Harvard architecture, which separates program code and data into distinct memory spaces. This design allows efficient processing as the microcontroller can access instructions and data simultaneously. It operates at speed of 16 MHz.

Components of Arduino Uno

A standard example of an Arduino board is the Arduino Uno, which includes:

- Microcontroller: The ATmega328P, featuring:
- 32 KB of flash memory for storing code.
- 2 KB of SRAM for temporary data storage.
- 1 KB of EEPROM for long-term data retention.

Pin Configuration:

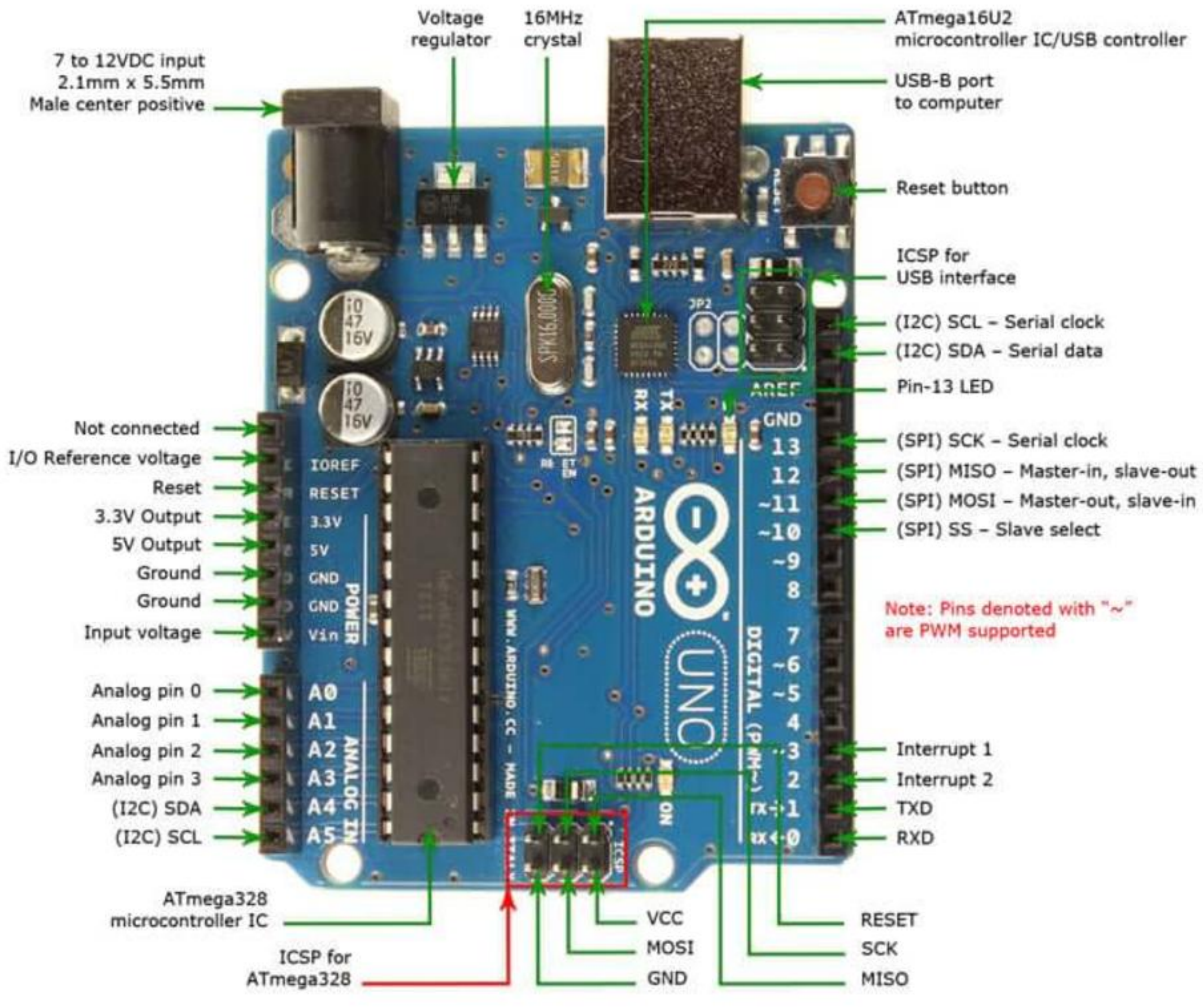
- 14 Digital I/O pins (6 capable of PWM).
- 6 Analog input pins with a resolution of 10 bits.
- USB connection for programming and power.
- Power jack for external supply (7-12V).

Programming the Arduino

Programming an Arduino involves writing sketches in the IDE. Each sketch is structured into three main parts:

- Variable Declaration
- Initialization (setup function)
- Control Code (loop function)

The IDE provides tools for writing, verifying, and uploading code directly to the board without needing additional hardware programmers due to a built-in bootloader.



b. Servo Motor

A servo motor is a rotary actuator that allows precise control of angular position. It is widely used in robotics and mechatronics applications.

How Servo Motors Work:

Internal Mechanism: Servo motors have a control circuit, a DC motor, and a potentiometer for feedback.

Control Signal: The position of the servo is determined by a PWM signal (Pulse Width Modulation). The width of the pulse corresponds to a specific angle.

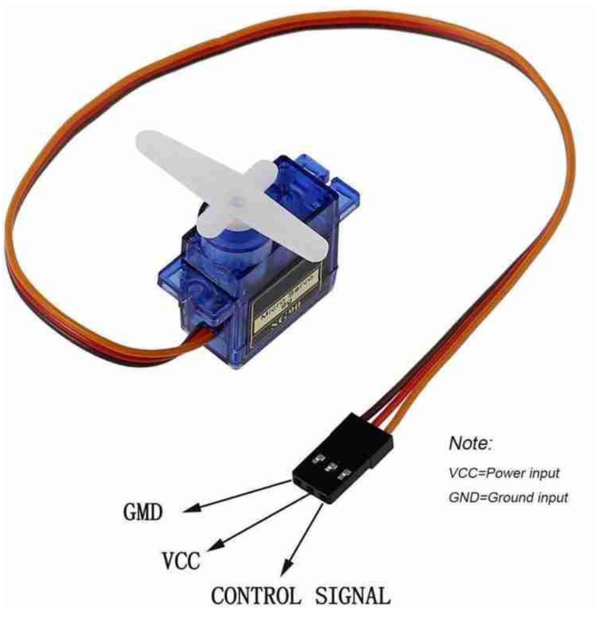
- A typical servo operates between 0° to 180°.
- For example, a pulse width of 1 ms sets the servo to 0°, while 2 ms sets it to 180°.

Changing Direction Using Voltage:

- The servo's internal controller compares the incoming PWM signal with the potentiometer's reading and adjusts the motor's voltage to align the positions.
- Clock Speed and Servo Function in Arduino:
- Arduino generates PWM signals at a certain frequency (usually 50 Hz for servos) to control the motor.

Servo Library: The Servo.h library in Arduino simplifies working with servo motors.

- servo.attach(pin); specifies the pin controlling the servo.
- servo.write(angle); moves the servo to a specific angle.



c. Ultrasonic sensor

The ultrasonic sensor, such as HC-SR04, measures distance by sending sound waves and calculating the time taken for the echo to return.

Working Principle (SONAR Technology):

Trigger Signal: Arduino sends a high signal to the sensor’s trigger pin for a brief period (usually 10 microseconds).

Sound Wave Emission: The sensor emits ultrasonic sound waves (40 kHz).

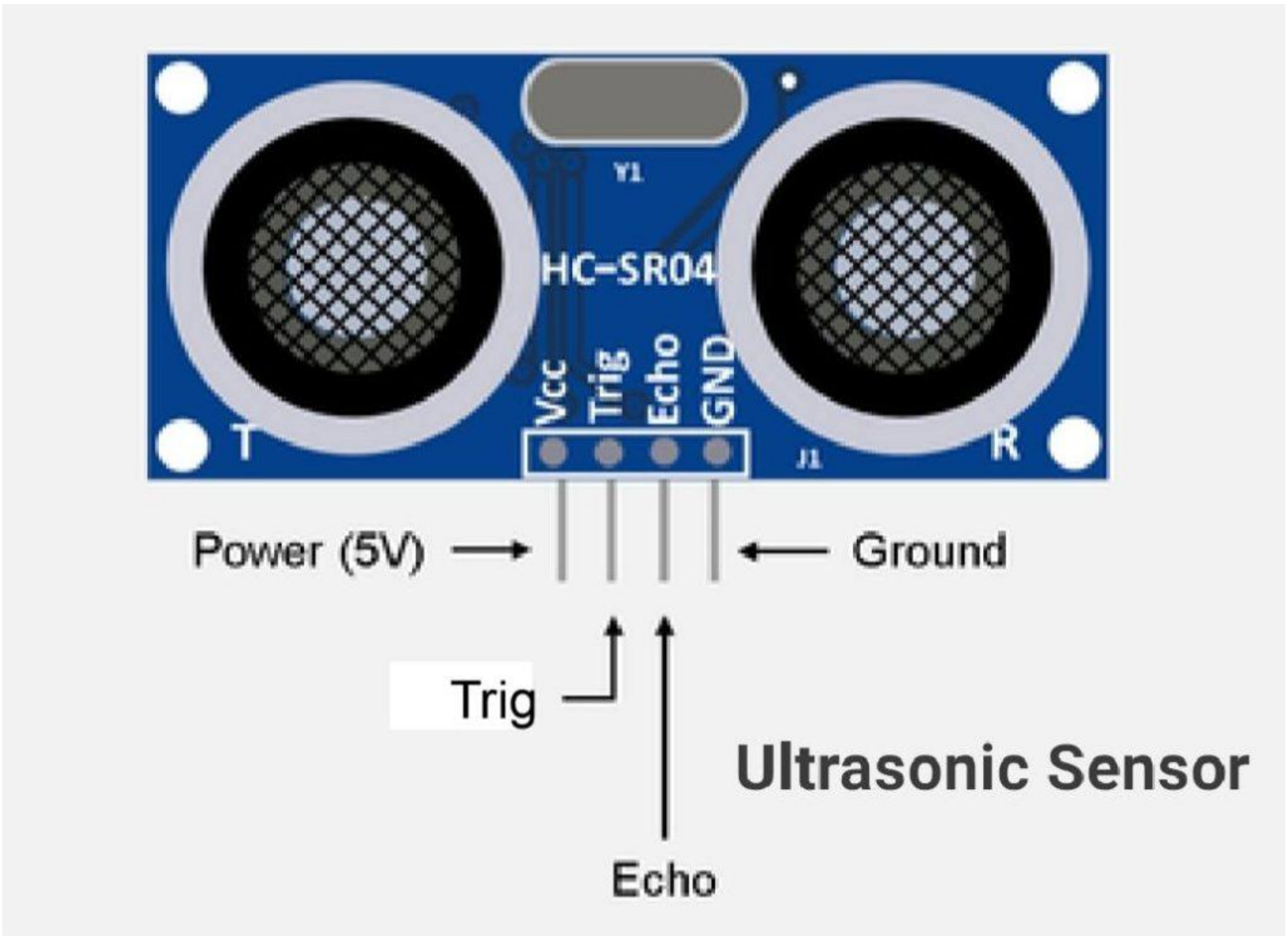
Echo Reception: When these waves hit an object, they bounce back to the sensor.

$$\text{Distance} = \frac{\text{Time} \times \text{Speed of Sound}}{2}$$

(Speed of sound ≈ 343 m/s in air).

Connecting the Ultrasonic Sensor:

- VCC: Connected to the Arduino's 5V.
- GND: Connected to the ground.
- Trigger Pin: Sends a pulse signal to initiate measurement.
- Echo Pin: Receives the reflected signal.



2. Software Design

The system is controlled by two software programs:

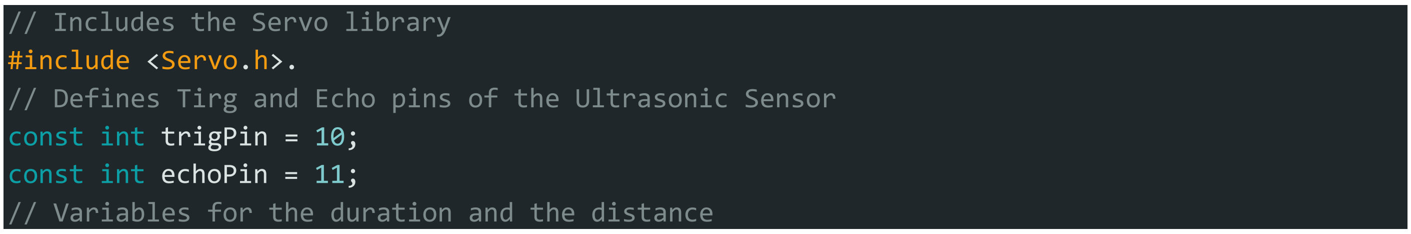
1. **Arduino Code**: Manages the sensor's distance readings and controls the servo motor.
2. **Processing Code**: Displays the radar visualization by rendering arcs representing different distance ranges and lines indicating detected objects.

The hardware and software interact through serial communication. The Arduino sends angle and distance data to the Processing program, which then visualizes it.

a. Arduino Code Overview:

- The sensor measures the time taken for the ultrasonic pulse to return, which is used to calculate the distance.
- The servo motor rotates the sensor across a defined range.
- Distance data is sent via the serial port to the Processing software.

Code screenshot:



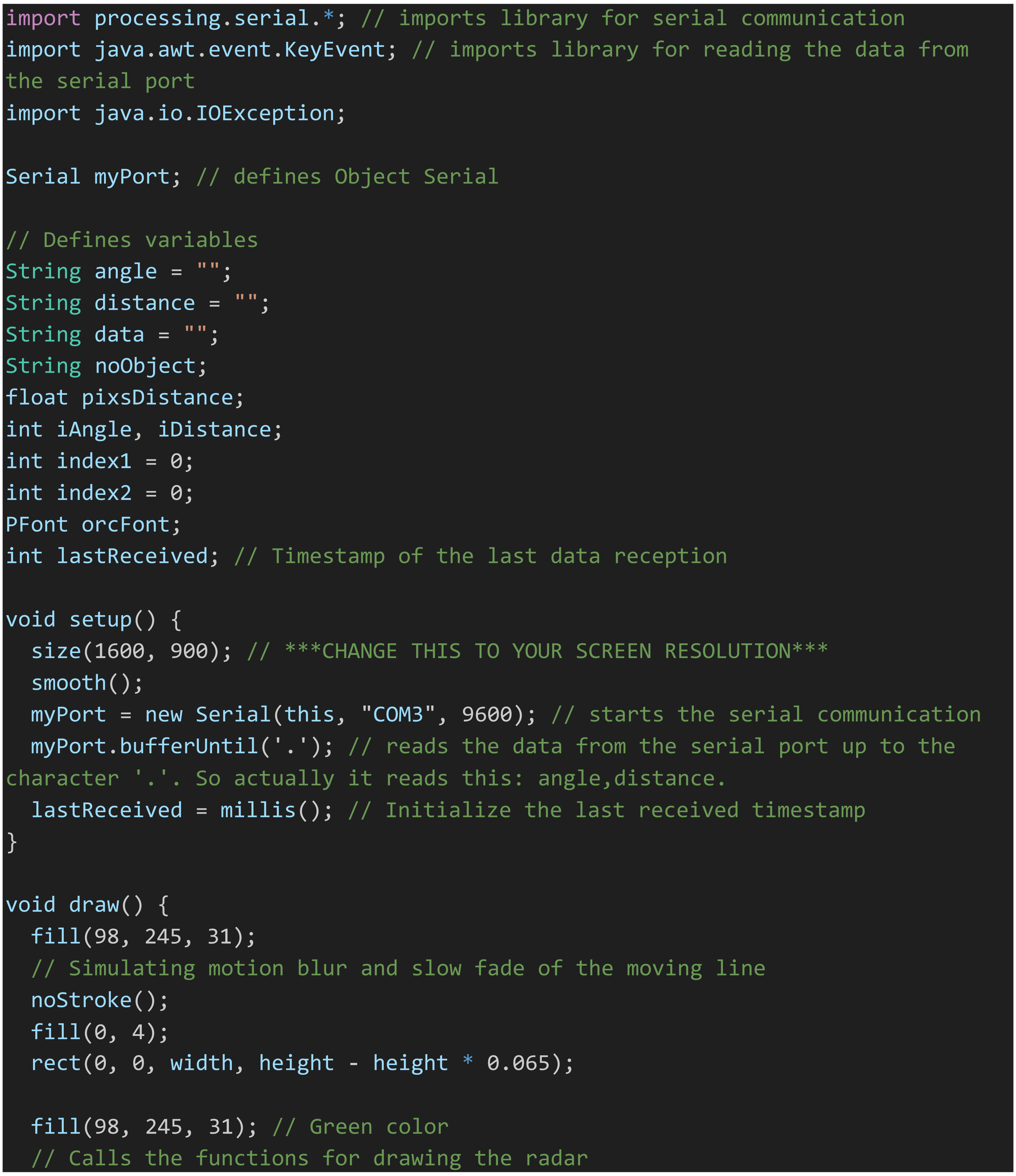
```
// Includes the Servo library
#include <Servo.h>.
// Defines Tirg and Echo pins of the Ultrasonic Sensor
const int trigPin = 10;
const int echoPin = 11;
// Variables for the duration and the distance
```

```
long duration;
int distance;
Servo myServo; // Creates a servo object for controlling the servo motor
void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600);
  myServo.attach(12); // Defines on which pin is the servo motor attached
}
void loop() {
  // rotates the servo motor from 15 to 165 degrees
  for(int i=15;i<=165;i++){
  myServo.write(i);
  delay(30);
  distance = calculateDistance();// Calls a function for calculating the distance measured
by the Ultrasonic sensor for each degree

  Serial.print(i); // Sends the current degree into the Serial Port
  Serial.print(","); // Sends addition character right next to the previous value needed
later in the Processing IDE for indexing
  Serial.print(distance); // Sends the distance value into the Serial Port
  Serial.print("."); // Sends addition character right next to the previous value needed
later in the Processing IDE for indexing
  }
  // Repeats the previous lines from 165 to 15 degrees
  for(int i=165;i>15;i--){
  myServo.write(i);
  delay(30);
  distance = calculateDistance();
  Serial.print(i);
  Serial.print(",");
  Serial.print(distance);
  Serial.print(".");
  }
}
// Function for calculating the distance measured by the Ultrasonic sensor
int calculateDistance(){

  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH); // Reads the echoPin, returns the sound wave travel
time in microseconds
  distance= duration*0.034/2;
  return distance;
}
```

b. Processing Code Overview:

- The radar system is simulated visually, displaying arcs at predefined distances (10 cm, 20 cm, etc.).
- Objects are represented as red lines at their corresponding angles and distances.

Code screenshot:



```
import processing.serial.*; // imports library for serial communication
import java.awt.event.KeyEvent; // imports library for reading the data from
the serial port
import java.io.IOException;

Serial myPort; // defines Object Serial

// Defines variables
String angle = "";
String distance = "";
String data = "";
String noObject;
float pixsDistance;
int iAngle, iDistance;
int index1 = 0;
int index2 = 0;
PFont orcFont;
int lastReceived; // Timestamp of the last data reception

void setup() {
  size(1600, 900); // ***CHANGE THIS TO YOUR SCREEN RESOLUTION***
  smooth();
  myPort = new Serial(this, "COM3", 9600); // starts the serial communication
  myPort.bufferUntil('.'); // reads the data from the serial port up to the
character '.'. So actually it reads this: angle,distance.
  lastReceived = millis(); // Initialize the last received timestamp
}

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
  // Calls the functions for drawing the radar
  drawRadar();
  drawLine();
  drawObject();
  drawText();

  // Check if no data has been received for 5 seconds
  if (millis() - lastReceived > 5000) {
    println("No data received in the last 5 seconds!");
  }
}

void serialEvent(Serial myPort) {
  // Reads the data from the Serial Port up to the character '.' and puts it
into the String variable "data".
  data = myPort.readStringUntil('.');

  if (data != null && data.length() > 0) {
    data = trim(data); // Remove any extra whitespace

    // Ensure we have valid data with a comma and sufficient length
    if (data.contains(",") && data.length() > 2) {
      index1 = data.indexOf(",");

      if (index1 > 0 && index1 < data.length() - 1) {
        angle = data.substring(0, index1); // Extract the angle
        distance = data.substring(index1 + 1); // Extract the distance

        // Try parsing the values safely
        try {
          iAngle = int(angle);
          iDistance = int(distance);
          lastReceived = millis(); // Update last received timestamp
        } catch (Exception e) {
          println("Error parsing data: " + data);
        }
      } else {
        println("Malformed data: " + data);
      }
    }
  }
}

void drawRadar() {
  pushMatrix();
  translate(width / 2, height - height * 0.074); // Moves the starting
coordinates to new location
```

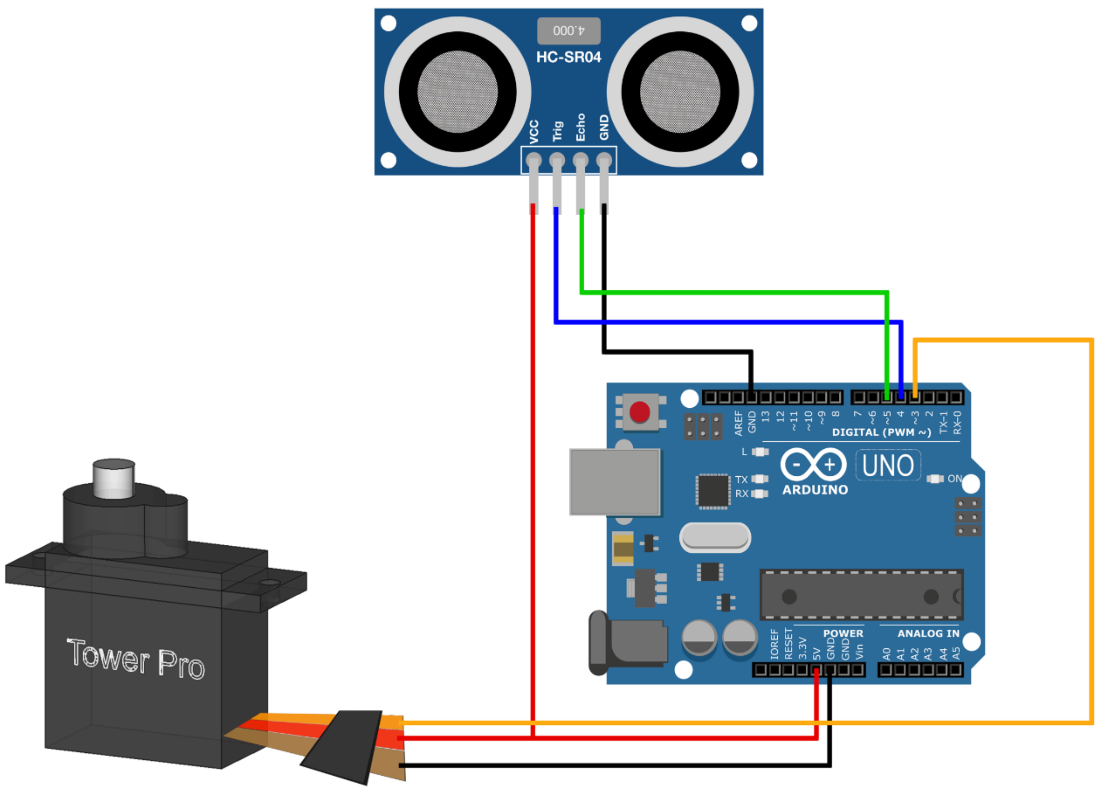
```
  noFill();
  strokeWeight(2);
  stroke(98, 245, 31);
  // Draws the arc lines
  arc(0, 0, (width - width * 0.0625), (width - width * 0.0625), PI, TWO_PI);
  arc(0, 0, (width - width * 0.27), (width - width * 0.27), PI, TWO_PI);
  arc(0, 0, (width - width * 0.479), (width - width * 0.479), PI, TWO_PI);
  arc(0, 0, (width - width * 0.687), (width - width * 0.687), PI, TWO_PI);
  // Draws the angle lines
  line(-width / 2, 0, width / 2, 0);
  line(0, 0, (-width / 2) * cos(radians(30)), (-width / 2) * sin(radians(30)));
  line(0, 0, (-width / 2) * cos(radians(60)), (-width / 2) * sin(radians(60)));
  line(0, 0, (-width / 2) * cos(radians(90)), (-width / 2) * sin(radians(90)));
  line(0, 0, (-width / 2) * cos(radians(120)), (-width / 2) *
sin(radians(120)));
  line(0, 0, (-width / 2) * cos(radians(150)), (-width / 2) *
sin(radians(150)));
  line((-width / 2) * cos(radians(30)), 0, width / 2, 0);
  popMatrix();
}

void drawObject() {
  pushMatrix();
  translate(width / 2, height - height * 0.074); // Moves the starting
coordinates to new location
  strokeWeight(9);
  stroke(255, 10, 10); // Red color
  pixsDistance = iDistance * ((height - height * 0.1666) * 0.025); // Converts
the distance from cm to pixels
  // Limiting the range to 40 cm
  if (iDistance < 40) {
    // Draws the object according to the angle and the distance
    line(pixsDistance * cos(radians(iAngle)), -pixsDistance *
sin(radians(iAngle)),
         (width - width * 0.505) * cos(radians(iAngle)), - (width - width *
0.505) * sin(radians(iAngle)));
  }
  popMatrix();
}

void drawLine() {
  pushMatrix();
  strokeWeight(9);
  stroke(30, 250, 60);
  translate(width / 2, height - height * 0.074); // Moves the starting
coordinates to new location
```

```
  line(0, 0, (height - height * 0.12) * cos(radians(iAngle)), - (height -
height * 0.12) * sin(radians(iAngle))); // Draws the line according to the
angle
  popMatrix();
}

void drawText() {
  pushMatrix();
  if (iDistance > 40) {
    noObject = "Out of Range";
  } else {
    noObject = "In Range";
  }
  fill(0, 0, 0);
  noStroke();
  rect(0, height - height * 0.0648, width, height);
  fill(98, 245, 31);
  textSize(25);

  text("10cm", width - width * 0.3854, height - height * 0.0833);
  text("20cm", width - width * 0.281, height - height * 0.0833);
  text("30cm", width - width * 0.177, height - height * 0.0833);
  text("40cm", width - width * 0.0729, height - height * 0.0833);
  textSize(40);
  text("VITC_ATC", width - width * 0.875, height - height * 0.0277);
  text("Angle: " + iAngle + " °", width - width * 0.48, height - height *
0.0277);
  text("Distance: ", width - width * 0.3, height - height * 0.0277);
  if (iDistance < 40) {
    text("        " + iDistance + " cm", width - width * 0.225, height - height
* 0.0277);
  }
  textSize(25);
  fill(98, 245, 60);
  translate((width - width * 0.4994) + width / 2 * cos(radians(30)), (height -
height * 0.0907) - width / 2 * sin(radians(30)));
  rotate(-radians(-60));
  text("30°", 0, 0);
  resetMatrix();
  translate((width - width * 0.503) + width / 2 * cos(radians(60)), (height -
height * 0.0888) - width / 2 * sin(radians(60)));
  rotate(-radians(-30));
  text("60°", 0, 0);
  resetMatrix();
  translate((width - width * 0.507) + width / 2 * cos(radians(90)), (height -
height * 0.0833) - width / 2 * sin(radians(90)));
```

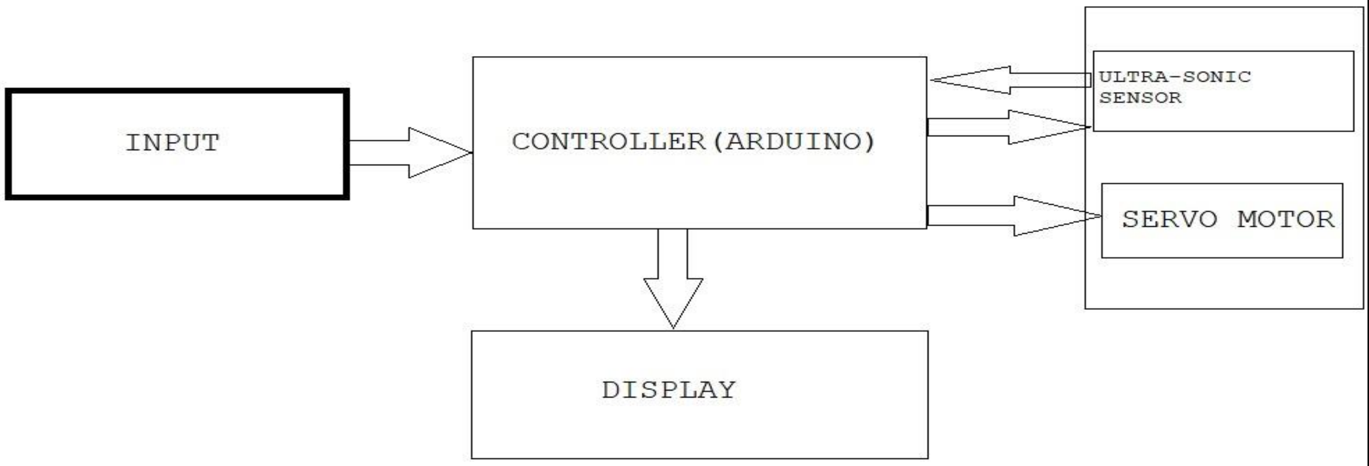
```
  rotate(radians(0));
  text("90°", 0, 0);
  resetMatrix();
  translate(width - width * 0.513 + width / 2 * cos(radians(120)), (height -
height * 0.07129) - width / 2 * sin(radians(120)));
  rotate(radians(-30));
  text("120°", 0, 0);
  resetMatrix();
  translate((width - width * 0.5104) + width / 2 * cos(radians(150)), (height -
height * 0.0574) - width / 2 * sin(radians(150)));
  rotate(radians(-60));
  text("150°", 0, 0);
  popMatrix();
}
```

Component Analysis

Circuit Diagram :



Block Diagram :



Component Price analysis :

Component	Description	Quantity	Price (INR)
Ultrasonic Sensor (HC-SR04)	Used for distance measurement (sonar principle)	1	150
Servo Motor (SG90)	Used for rotating the sensor to scan different angles	1	100
Arduino Board (Uno)	The microcontroller to process sensor data and control servo	1	400
Jumper Wires	For connecting components on the breadboard or Arduino	10	30
Breadboard	For testing the circuit without soldering	1	50
		Total Cost	Rs. 730

Results and Analysis

Object Detection

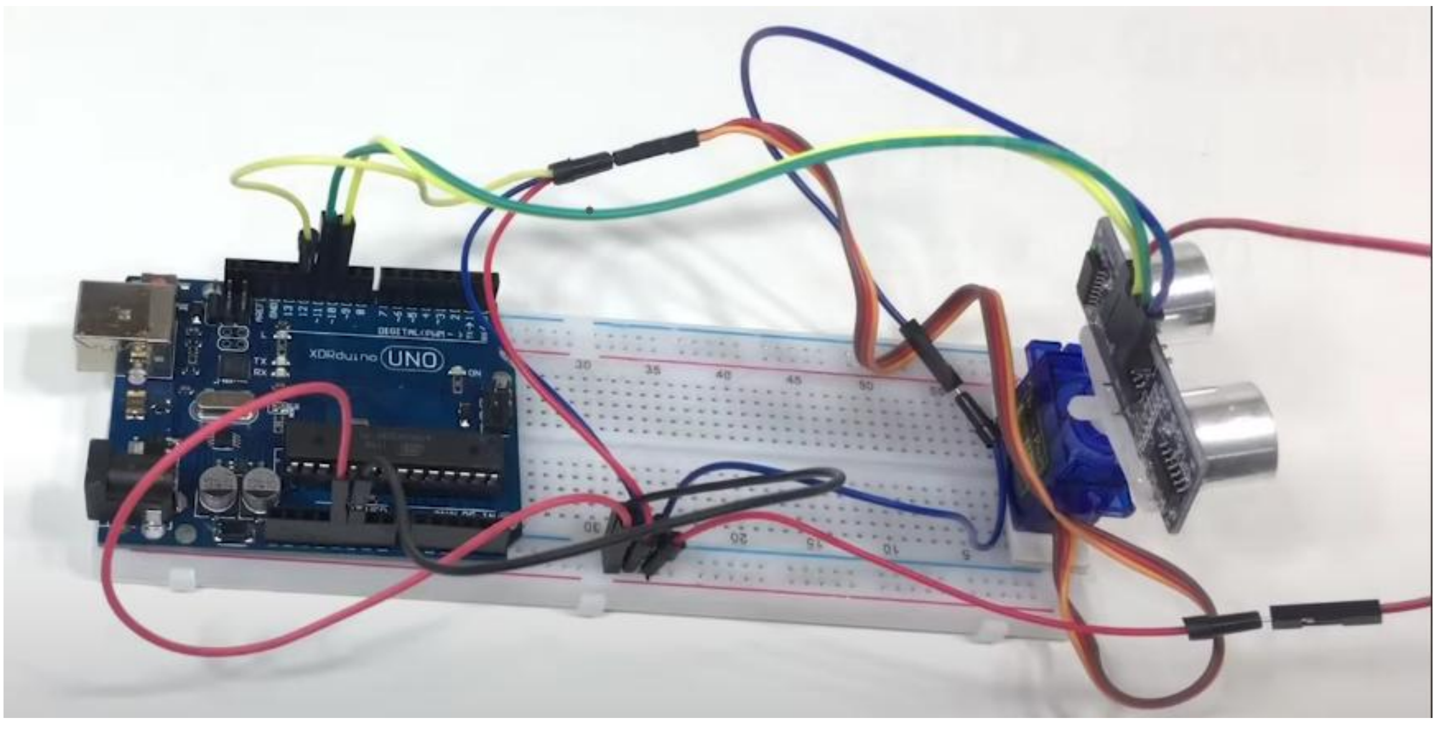
The radar successfully detected objects within a range of 0–40 cm. The real-time data displayed in the Processing interface allowed for the identification of object locations based on their angular position and distance.

Accuracy

Testing revealed that the ultrasonic radar was accurate within 1 cm for objects up to 40 cm away. The system's accuracy decreases beyond this range, limiting its practical use in larger spaces without additional sensors.

Limitations

- **Range**: The system has a limited detection range of 40 cm due to the sensor's specifications.
- **Interference**: Environmental factors such as noise and temperature variations can affect the sensor's readings.



Discussion and Future Work

- **Discussion**

The Ultrasonic Radar system developed offers a cost-effective solution for localized airport monitoring. Utilizing basic components—ultrasonic sensors, a servo motor, an Arduino microcontroller, and Processing software—this system provides real-time visualization of objects within a 40 cm range. Its affordability and modularity are significant advantages, making it accessible for airports looking to enhance ground-level surveillance.

However, the system's short detection range and limited precision restrict its application to small, confined areas. While effective for monitoring specific zones, it may not be suitable for larger or more complex environments.

- **Future Work**

To improve the system, several areas can be explored:

1. **Extended Detection Range**: Implement sensors with greater range or alternative technologies like infrared or laser to cover larger areas.
2. **Increased Accuracy**: Enhance measurement precision through better sensors and refined data processing techniques.
3. **System Integration**: Combine with existing airport security systems for a more comprehensive solution, potentially integrating with CCTV or alarm systems.
4. **Enhanced User Interface**: Improve the Processing software's graphical interface and add features like interactive controls and data logging.
5. **Field Testing**: Conduct real-world testing to validate system performance and identify areas for improvement.

By addressing these aspects, the Ultrasonic Radar system can be further developed to offer more effective and versatile monitoring capabilities for airport environments.

Conclusion

The Ultrasonic Radar system presents an effective and affordable solution for localized airport monitoring. Using simple components—a sensor, servo motor, Arduino, and Processing software—the system offers real-time object detection within a 40 cm range. Its cost-effectiveness and modularity make it suitable for enhancing surveillance in specific airport areas.

While the system is functional for its intended purpose, future improvements could focus on extending its detection range, increasing accuracy, integrating with existing security systems, and refining the user interface. Further testing will help validate its effectiveness and guide future enhancements.

Overall, this system provides a promising tool for airport security, improving safety and operational efficiency.

References

1. International Civil Aviation Organization. (2021). *Manual on Air Traffic Services (ATS) Data Link Communications*. ICAO.

2. Lee, J., Park, J., & Kim, K. (2016). Ultrasonic distance measurement system for obstacle avoidance. *Journal of Robotics and Mechatronics*, 28(3), 408-415.

3. Pishvaei, M., & Fathi, M. (2018). Real-time distance measurement using Arduino and ultrasonic sensors. *International Journal of Engineering Research & Technology*, 7(9), 421-425.

4. McCormick, L., & DeHaan, K. (2017). Interactive data visualization with Processing. *Computer Graphics Forum*, 36(1), 19-29.

5. Wang, X., Liu, Y., & Zhang, L. (2019). Ultrasonic radar system for indoor navigation and obstacle detection. *Sensors*, 19(11), 2523.