EECS 3215: Embedded Systems Project:

Servo-Controlled Obstacle Detection Radar

Team Members:
Vashavi Shah (217752478)
Harsh Parmar (218220038)

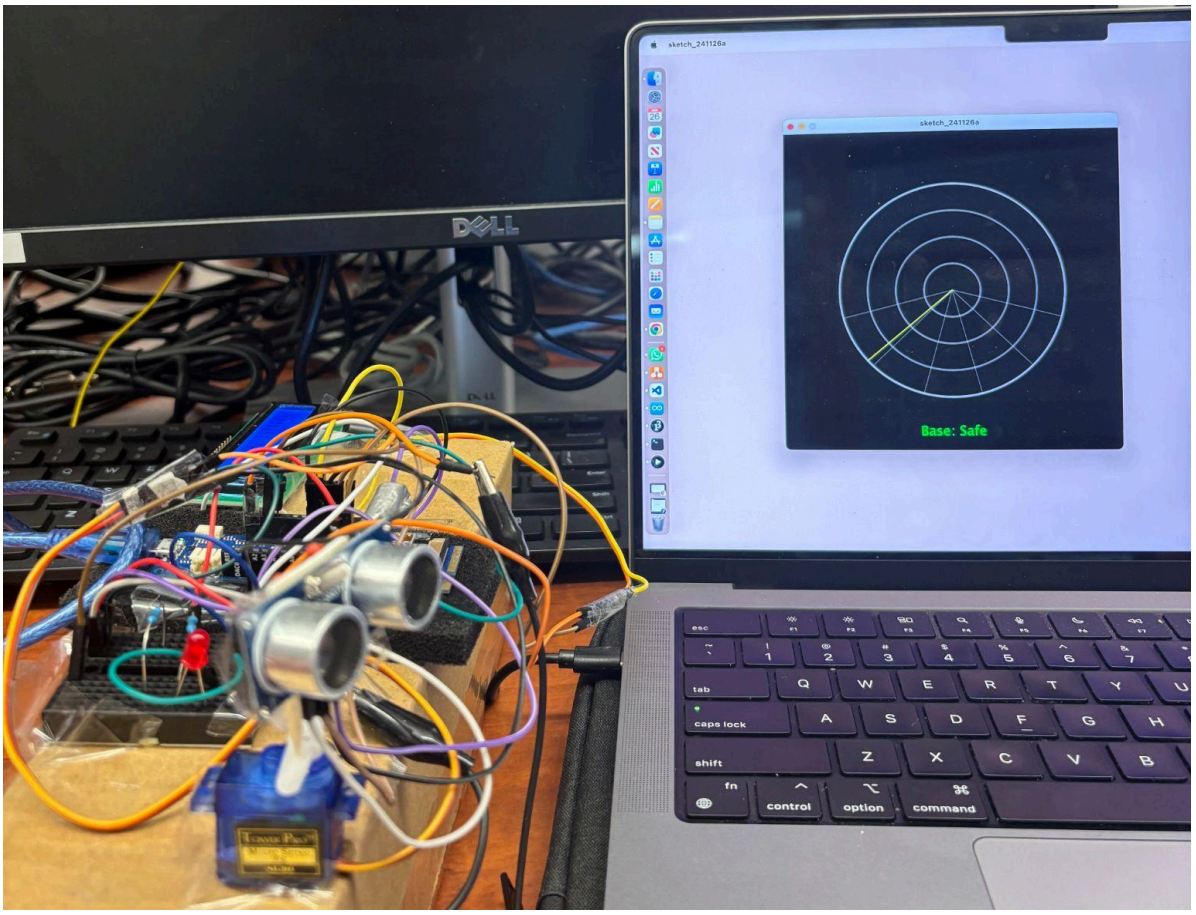


Table of Contents

1. Overview

The **Servo-Controlled Obstacle Detection Radar** is an innovative security system that combines hardware and software to create a dynamic and efficient radar for real-time object detection and visualization. At its core, the system employs an **ultrasonic sensor** mounted on a **servo motor** to scan the surroundings and measure the distance to obstacles. It provides immediate feedback via an external **LCD display**, auditory alerts through a **buzzer**, and visual indications using an **LED**. Additionally, a **Processing-based graphical interface** enhances the user experience by visualizing the radar's status, marking detected objects as **safe or dangerous.** This project highlights the potential of integrating **cost-effective hardware** with simple programming to develop practical and scalable obstacle detection systems.

2. Motivation

The motivation for this project stems from the need to develop an **affordable, scalable,** and **efficient obstacle detection system** that can be applied to real-world scenarios such as robotics, automation, and security. Accurate and reliable object detection is critical in these fields, yet current solutions often depend on **costly hardware or complex programming.** This project aims to bridge that gap by simulating a functional radar system that leverages **basic components** like **ultrasonic sensors** and **servo motors**, combined with accessible tools such as **Arduino and Processing**. By doing so, it demonstrates the practical application of **sensors**, **actuators**, and **microcontrollers** in creating intelligent and cost-effective monitoring solutions for a wide range of applications.

3. Objectives

The **primary objectives** of this projects are:

- To design and implement a radar system using an **Arduino microcontroller.**
- To integrate an **ultrasonic sensor** with a **servo motor** for obstacle detection.
- To provide **real-time feedback** through multiple output devices (LCD, LED, buzzer).
- To create a visual representation of the radar data using **Processing software.**
- To demonstrate the system's ability to detect and alert users of **potential threats.**
- To integrate hardware and software effectively for seamless operation.

4. Hardware Description

The Servo-Controlled Obstacle Detection Radar relies on a carefully selected set of hardware components, each contributing to the system's functionality. These components were assembled creatively using accessible materials to ensure the structure was both stable and lightweight.

4.1 Components Used

1. **Audrino MKR Vidor 4000:** It is a powerful microcontroller that serves as the brain of the system. It processes data from the ultrasonic sensor, controls the servo motor, and

communicates with output devices such as the LCD, buzzer, and LED. Its compact size and high-performance capabilities make it ideal for projects involving real-time data processing and hardware control.

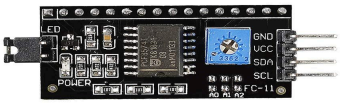
2. **Ultrasonic Sensor:** This sensor measures the distance to nearby objects by emitting sound waves and recording the time taken for the echo to return. The sensor calculates the distance using the formula:

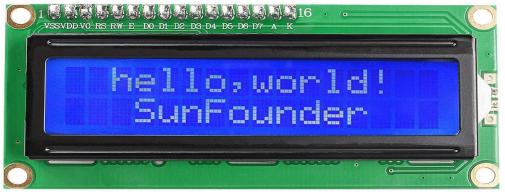
$$Distance\ (cm) = \frac{(Sound\ Duration\ (\mu s) * 0.034}{2}$$

The sensor is mounted on the servo motor to allow it to sweep across a 180° field, enabling comprehensive coverage of the surrounding area.

3. **Servo Motor:** The servo motor provides the rotational motion needed to scan the environment. It moves the ultrasonic sensor in precise increments, enabling the system to measure distances at various angles. This sweeping motion ensures the radar covers a wide detection range.

4. **External LCD Display (I2C-based):** The I2C-based LCD display provides real-time feedback on the system's status. It shows messages such as "Base Safe" or "Base Attacked", offering an intuitive way for users to understand the system's state.



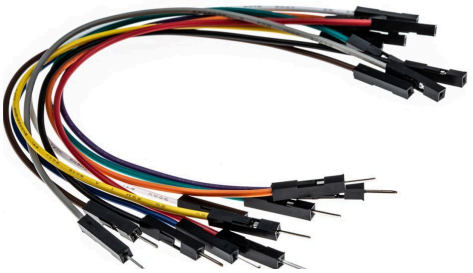


5. **Piezo Buzzer:** The piezo buzzer produces auditory alerts when a threat is detected. It provides a sharp, attention-grabbing sound that enhances the system's effectiveness in scenarios where visual feedback might go unnoticed.

6. **LED Light:** The light visually indicates the system's status. A glowing LED signifies a detected threat, adding an additional layer of real-time feedback for the user.

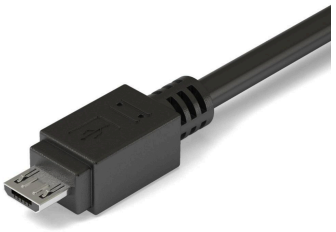
7. **Breadboard:** A breadboard was used to connect components without soldering, allowing for a flexible and modular configuration. It simplifies troubleshooting and modifications during the assembly process.

8. **Jumper Wires:** Jumper wires facilitated electrical connections between the components. Their flexibility ensured neat and organized wiring within the compact setup.



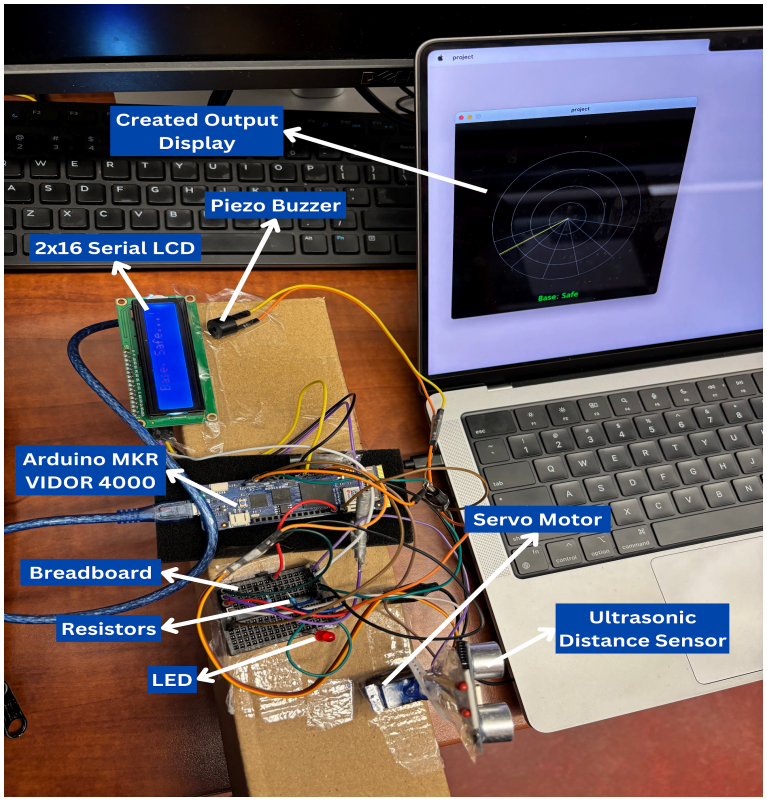
9. **Resistors:** Resistors were used to regulate current and protect sensitive components like LEDs and sensors from potential damage caused by excessive voltage.

10. **USB Cable:** The USB cable powered the Arduino and enabled programming from a computer. It also facilitated testing and debugging during development.



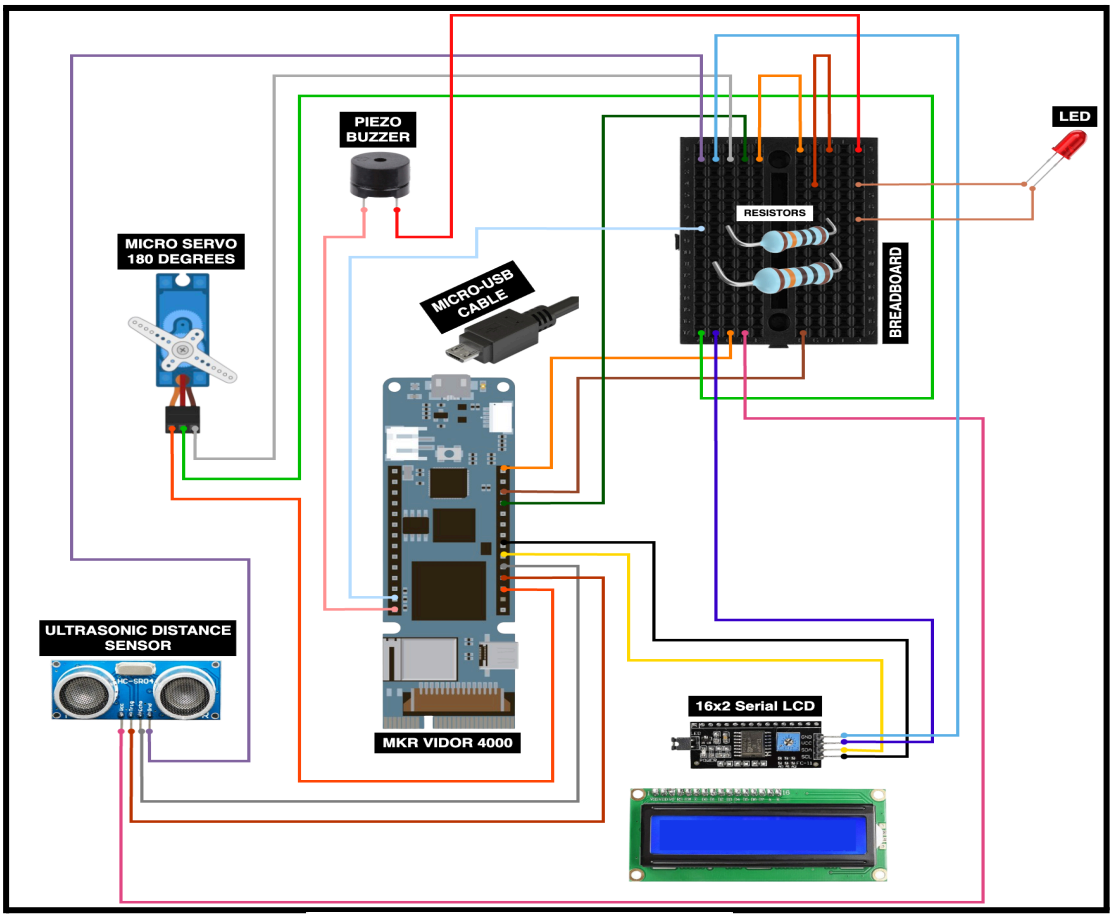
11. **Crocodile Clip:** Crocodile clips provided secure and temporary connections during the testing phase, making it easy to attach and detach components without damaging the wiring.

12. **Miscellaneous Components:**
    a. **Matchsticks:** To mount the ultrasonic sensor on the servo motor, matchsticks were used to provide a lightweight yet stable support structure.
    b. **Cellotape:** Cellotape was used to attach components firmly in place, including securing the matchstick-mounted sensor and other hardware elements. It provides flexibility for adjustments during assembly.
    c. **Cardboard Box:** To house the entire setup, all components were neatly affixed to a cardboard box using cellotape. This approach kept the assembly compact and portable while maintaining the necessary stability for accurate measurements.



The model diagram

4.2 Assembly Details



The circuit diagram

The Servo-Controlled Obstacle Detection Radar connects its components systematically to ensure seamless operation. The **ultrasonic sensor** is powered by connecting its **VCC** and **GND pins** to the Arduino's 5V and GND pins, respectively. Its **Trig** and **Echo pins** are linked to digital pins 9 and 10 to facilitate distance measurement. The **servo motor,** responsible for rotating the sensor, is connected via its signal pin to digital pin 8, with power and ground lines routed through the breadboard for stability. The **LCD display (I2C-based)** is integrated using its SDA and SCL pins, which are connected to the Arduino's corresponding A4 and A5 pins, while its power is sourced from the 5V and GND pins.

For auditory alerts, the **piezo buzzer** is connected to digital pin 5 and grounded through the **breadboard**. The **LED light**, used for visual indications, is linked to digital pin 4 via a **resistor** to regulate current and protect it from damage. The breadboard plays a crucial role in organizing the connections, providing a shared power supply and grounding for components like the servo motor, buzzer, and LED. A **USB cable** powers the Arduino and facilitates programming from a computer, while crocodile clips ensure temporary, secure connections during testing and debugging phases.

This setup is designed for simplicity and efficiency, ensuring all components interact harmoniously. It follows the logic to create a functional radar system that detects and visualizes objects effectively.

5. Software Description

The software component of the Servo-Controlled Obstacle Detection Radar involves two key pieces of code: the Arduino code for controlling hardware and the Processing code for visualizing data on a graphical user interface (GUI). Below is a detailed breakdown of the functionality of both codes, including explanations of significant code segments.

5.1 Arduino Code

The Arduino code controls the hardware, including the ultrasonic sensor, servo motor, buzzer, LED, and LCD. It performs distance measurement, threat detection, and hardware feedback.

a. Variables

```
// Includes the Servo library for controlling servo motors
#include <Servo.h>

// Includes the Wire and LCD libraries for I2C communication and LCD control
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Initialize an LCD object with I2C address 0x27 for a 20x4 display
LiquidCrystal_I2C lcdDisplay(0x27, 20, 4);

// Define pin numbers for the LED and Buzzer
int statusLed = 4;
int alarmBuzzer = 5;

// Define pin numbers for the Ultrasonic Sensor
const int ultrasonicTrigPin = 9;  // Trigger pin for the ultrasonic sensor
const int ultrasonicEchoPin = 10; // Echo pin for the ultrasonic sensor

// Variables to hold the duration of the sound pulse and the calculated distance
long soundDuration;
int measuredDistance;

// Create a Servo object to control the servo motor
Servo baseServo;
```

- **lcdDisplay:** Manages the I2C-based LCD to display system status.
- **statusLed and alarmBuzzer:** Control the LED and buzzer outputs.
- **ultrasonicTrigPin and ultrasonicEchoPin:** Define the pins for the ultrasonic sensor's trigger and echo functionalities.
- **soundDuration and measuredDistance:** Store the time and calculated distance for obstacle detection.
- **baseServo:** Represents the servo motor used for scanning the environment.

b. Setup Function

```
/**
 * setup()
 * Initializes all peripherals, including LED, Buzzer, LCD, and Servo.
 * Configures the ultrasonic sensor pins and sets up serial communication.
 */
void setup() {
    // Set up LED and Buzzer as outputs
    pinMode(statusLed, OUTPUT);
    pinMode(alarmBuzzer, OUTPUT);

    // Activate the buzzer briefly at startup
    tone(alarmBuzzer, 1000, 2000);

    // Initialize the LCD display
    lcdDisplay.init();
    lcdDisplay.backlight(); // Turn on the LCD backlight

    // Set up ultrasonic sensor pins
    pinMode(ultrasonicTrigPin, OUTPUT);
    pinMode(ultrasonicEchoPin, INPUT);

    // Start serial communication for debugging
    Serial.begin(9600);

    // Attach the servo motor to pin 8
    baseServo.attach(8);
}
```

- Configures pins for the ultrasonic sensor, LED, and buzzer.

- Initializes the LCD display and enables its backlight.
- Attaches the servo motor to pin 8 and sets up serial communication for debugging.
- Activates the buzzer briefly during initialization to indicate the system is powered on.

c. Distance Measurement Function

```
/**
 * computeDistance()
 * Calculates the distance using the ultrasonic sensor by sending a sound pulse
 * and measuring the time taken for the echo to return.
 *
 * @return Distance in centimeters
 */
int computeDistance() {
    digitalWrite(ultrasonicTrigPin, LOW);
    delayMicroseconds(2);

    // Send a 10-microsecond pulse to the trigger pin
    digitalWrite(ultrasonicTrigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(ultrasonicTrigPin, LOW);

    // Measure the duration of the echo pulse
    soundDuration = pulseIn(ultrasonicEchoPin, HIGH);

    // Calculate distance based on the duration of the pulse
    measuredDistance = soundDuration * 0.034 / 2;

    return measuredDistance;
}
```

- Sends a 10-microsecond pulse from the ultrasonicTrigPin.
- Uses pulseIn to measure the time (in microseconds) taken for the echo to return to the ultrasonicEchoPin.
- Converts the duration into distance using the formula:

$$Distance\ (cm)\ = \frac{(Sound\ Duration\ (\mu s)\ *\ 0.034}{2}$$

d. Main Loop Function

```
/**
 * loop()
 * Continuously rotates the servo motor, calculates the distance from the ultrasonic sensor,
 * and updates the system state (LCD, LED, and Buzzer) based on the measured distance.
 */
void loop() {
    // Rotate the servo motor from 15 to 165 degrees
    for (int angle = 15; angle <= 165; angle++) {
        delay(30);
        measuredDistance = computeDistance(); // Calculate the distance

        if (measuredDistance <= 19) {
            // If an object is detected within 19 cm, display a warning
            lcdDisplay.setCursor(2, 0);
            lcdDisplay.print("Base: Attacked!");
            digitalWrite(statusLed, HIGH); // Turn on the warning LED
            tone(alarmBuzzer, 440); // Activate the buzzer with a warning tone
            delay(2000); // Keep the warning active for 2 seconds
        } else {
            // If no object is detected, indicate the base is safe
            baseServo.write(angle); // Move the servo to the current angle
            lcdDisplay.setCursor(2, 0);
            lcdDisplay.print("Base: Safe...!");
            noTone(alarmBuzzer); // Turn off the buzzer
            digitalWrite(statusLed, LOW); // Turn off the warning LED
        }

        // Print debug information to the Serial Monitor
        Serial.print(angle); // Current servo angle
        Serial.print(",");
        Serial.print(measuredDistance); // Measured distance
        Serial.print(".");
    }

    // Rotate the servo motor back from 165 to 15 degrees
    for (int angle = 165; angle > 15; angle--) {
        delay(30);
        measuredDistance = computeDistance(); // Calculate the distance

        if (measuredDistance <= 19) {
            // Display a warning if the base is under attack
            lcdDisplay.setCursor(2, 0);
            lcdDisplay.print("Base, Attacked!");
            digitalWrite(statusLed, HIGH);
            tone(alarmBuzzer, 440);
            delay(2000);
        } else {
            // Indicate that the base is safe
            baseServo.write(angle);
            lcdDisplay.setCursor(2, 0);
            lcdDisplay.print("Base, Safe...!");
            noTone(alarmBuzzer);
            digitalWrite(statusLed, LOW);
        }

        // Print debug information to the Serial Monitor
        Serial.print(angle);
        Serial.print(",");
        Serial.print(measuredDistance);
        Serial.print(".");
    }

    delay(300); // Small delay before restarting the loop
}
```

- Rotates the servo motor from 15° to 165°, measuring the distance at each angle.
- If the distance is ≤19 cm, it triggers the buzzer and LED, and updates the LCD with "Base: Attacked!".
- For distances >19 cm, it turns off the alerts and displays "Base: Safe!".
- Sends angle and distance data to the Serial Monitor for debugging or GUI visualization.

5.2 Processing Code

The Processing code creates a GUI that visualizes the radar's sweep and detected objects.

a. Variables

```
import processing.serial.*; // Serial library for communication
Serial serialPort;          // Serial object for communication

// Variables to store the angle and distance
String currentAngleString = "";
String currentDistanceString = "";
int currentAngle, currentDistance;  // Integer variables for angle and distance
int radarStatus = 0;  // 0 = Safe, 1 = Danger (Attacked)
int previousRadarStatus = 0;  // To track the previous radar status for comparison

// Constants for radar visualization
int radarMaxRange = 200;  // Maximum radar range (20 cm converted to pixels)
int radarCenterX, radarCenterY;  // Radar's center position on the screen
int servoSweepStart = 15;  // Servo sweep start angle (in degrees)
int servoSweepEnd = 165;   // Servo sweep end angle (in degrees)

// Constants for color coding
int safeZoneColor = color(0, 255, 0);     // Green color for Safe zone
int dangerZoneColor = color(255, 0, 0);  // Red color for Danger zone
int radarGridColor = color(255, 100);    // Light white for radar grid lines
```

- The **serialPort** object handles communication with the Arduino.
- Variables like **currentAngle** and **currentDistance** store real-time data received from the Arduino.
- **radarMaxRange** defines the visual range of the radar, mapped to a pixel value of 200.
- **radarCenterX** and **radarCenterY** position the radar at the center of the GUI window.
- Constants such as **safeZoneColor** and **dangerZoneColor** determine the colors used for detected objects.

b. Setup Function

```
void setup() {
  size(600, 600);  // Set the window size
  radarCenterX = width / 2;  // Calculate radar center X
  radarCenterY = height / 2; // Calculate radar center Y

  background(0);  // Set the background color to black

  // Initialize serial communication (replace with your port)
  serialPort = new Serial(this, "/dev/tty.usbmodem101", 9600);
  serialPort.bufferUntil('.');  // Read data until '.' character
}
```

- The GUI window size is set to 600x600 pixels.
- The radar's center is positioned at the middle of the screen using **radarCenterX** and **radarCenterY.**
- A black background is created to mimic the appearance of a radar interface.
- Serial communication is initialized, and the program reads data from the Arduino until a period **(.)** character is received.

c. draw Function

```
void draw() {
  // Clear the screen and redraw the radar
  background(0);
  drawRadarGrid();

  // Draw servo sweep line and detected object blips
  drawRadarSweep();
  drawDetectedObjects();

  // Display radar status and angle/distance information
  displayRadarStatus();
  if (currentDistance <= 20) {  // Display only when an object is detected within 20 cm
    displayAngleAndDistance();
  }
}
```

- **background(0):** Clears the screen by setting the background color to black. This ensures that each frame is drawn fresh, avoiding overlapping visuals.
- **drawRadarGrid():** Draws the radar grid, including concentric circles and angle lines, to provide a reference for object visualization.
- **drawRadarSweep()**: Renders the sweeping line of the radar, mimicking the servo motor's rotation.

- **drawDetectedObjects():** Displays the detected objects as circles (safe in green or dangerous in red) based on their distance.
- **displayRadarStatus():** Shows whether the radar is in "Safe" or "Attacked" mode.
- **displayAngleAndDistance():** Displays the angle and distance of the detected object when it is within the 20 cm threshold.

d. Serial Data Processing Function

```
// Function to read the serial data from Arduino
void serialEvent(Serial serialPort) {
  String serialData = serialPort.readStringUntil('.');  // Read data until '.' character

  if (serialData != null) {
    // Remove any extra spaces or newline characters
    serialData = trim(serialData);

    // Split the data into angle and distance parts
    String[] dataParts = split(serialData, ',');
    if (dataParts.length == 2) {
      currentAngleString = dataParts[0];  // Angle part
      currentDistanceString = dataParts[1];  // Distance part

      // Convert string data to integers
      currentAngle = int(currentAngleString);
      currentDistance = int(currentDistanceString);

      // Update radar status based on the distance
      radarStatus = (currentDistance <= 19) ? 1 : 0;  // Danger if distance <= 19 cm
    }
  }
}
```

- **readStringUntil('.'):** Reads incoming serial data from the Arduino until a period (.) character is encountered, ensuring complete data is captured.
- **trim():** Removes extra whitespace or newline characters from the incoming data.
- **split(serialData, ','):** Splits the data into two parts: angle and distance, based on the comma delimiter.
- **currentAngleString and currentDistanceString:** Temporarily store the angle and distance as strings for conversion.
- **int(currentAngleString) and int(currentDistanceString):** Convert the string values into integers for calculations and visualization.
- **radarStatus:** Updates the radar's state to "Danger" (1) if the detected distance is ≤19 cm; otherwise, it remains "Safe" (0).

e. Radar Grid Function

```
// Function to draw the radar grid (background circles and angle lines)
void drawRadarGrid() {
  noFill();
  stroke(radarGridColor);  // Light white color for grid

  // Draw concentric circles to represent distance intervals (5 cm each)
  for (int i = 5; i <= 20; i += 5) {
    float radius = map(i, 0, 20, 0, radarMaxRange);  // Map distance to radar's radius
    ellipse(radarCenterX, radarCenterY, radius * 2, radius * 2);
  }

  // Draw angle lines every 30°
  for (int angle = servoSweepStart; angle <= servoSweepEnd; angle += 30) {
    float xEnd = radarCenterX + radarMaxRange * cos(radians(angle));
    float yEnd = radarCenterY + radarMaxRange * sin(radians(angle));
    line(radarCenterX, radarCenterY, xEnd, yEnd);  // Draw the angle line
  }
}
```

- Draws concentric circles representing distance intervals (5 cm, 10 cm, etc.) using the map function.
- Adds radial lines at 30° intervals for better visualization of the radar's field of view.

f. Radar Sweep Line Function

```
// Function to draw the servo sweep line (rotating line)
void drawRadarSweep() {
  stroke(255, 255, 0);  // Yellow color for sweep line

  // Calculate the end point of the sweep line
  float sweepX = radarCenterX + radarMaxRange * cos(radians(currentAngle));
  float sweepY = radarCenterY + radarMaxRange * sin(radians(currentAngle));
  line(radarCenterX, radarCenterY, sweepX, sweepY);  // Draw the sweep line
}
```

- The servo angle **(currentAngle)** determines the position of the sweep line.
- The sweep line extends from the radar center to the radar's maximum range.

g. Detected Objects Function

```
// Function to draw detected objects (blips)
void drawDetectedObjects() {
  // Map the detected distance to the radar's radius
  float objectDistance = map(currentDistance, 0, 20, 0, radarMaxRange);

  // Calculate the object's position based on the angle and distance
  float objectX = radarCenterX + objectDistance * cos(radians(currentAngle));
  float objectY = radarCenterY + objectDistance * sin(radians(currentAngle));

  // Set color based on radar status
  fill(radarStatus == 1 ? dangerZoneColor : safeZoneColor);
  noStroke();
  ellipse(objectX, objectY, 10, 10);  // Draw the object as a circle
}
```

- Converts the object's distance to a pixel radius using the **map** function.
- Calculates the object's position based on its angle and radius.
- Draws the object as a colored circle (red for danger, green for safe).

h. Display Radar Status Function

```
// Function to display the radar status (Safe or Danger)
void displayRadarStatus() {
  fill(255);  // White text color
  textSize(24);
  textAlign(CENTER, CENTER);

  // Display status message
  if (radarStatus == 0) {
    fill(safeZoneColor);  // Green for Safe
    text("Base: Safe", width / 2, height - 40);
  } else {
    fill(dangerZoneColor);  // Red for Danger
    text("Base: Attacked", width / 2, height - 40);
  }
}
```

- **fill(safeZoneColor) or fill(dangerZoneColor):** Sets the status text color to green for "Safe" or red for "Danger."
- **text():** Displays the radar's current status at the bottom of the GUI.

i. Display Radar Information Function

```
// Function to display angle and distance information
void displayAngleAndDistance() {
  fill(255);  // White text color
  textSize(18);
  textAlign(CENTER, TOP);

  // Display the current angle and distance
  text("Angle: " + currentAngle + "°", width / 2, 20);
  text("Distance: " + currentDistance + " cm", width / 2, 40);
}
```

- **text():** Displays the angle and distance of the detected object at the top center of the screen.

6. Working Principle Description

Distance Measurement:

The ultrasonic sensor calculates distances using the formula:

$$Distance\ (cm)\ = \frac{(Sound\ Duration\ (\mu s)\ *\ 0.034}{2}$$

Sound pulses are emitted, and the time taken for the echo to return is used to compute the distance.

Servo Motor Sweep:

The servo motor rotates the ultrasonic sensor from 15° to 165°, enabling a comprehensive scan of the environment. Measurements are taken at each angle.

Visualization:

The GUI plots detected objects:

- Green Dots: Objects within the safe range.
- Red Dots: Objects within the critical range (≤19 cm).

7. Demonstration Walkthrough

Safe State

- **LCD Output:** Displays "Base Safe."
- **Processing GUI:** Marks detected objects as green dots.
- **LED and Buzzer:** Remain off.

Danger State

- **LCD Output:** Displays "Base Attacked."
- **Processing GUI:** Marks detected objects as red dots, logs their angle and distance.
- **LED and Buzzer:** Activate to provide alerts.

8. Applications

This Servo Control Obstacle Detection Radar system has potential application in various fields:

- **Security Systems:** For monitoring restricted areas or perimeters.
- **Robotics:** Assisting in obstacle avoidance and navigation.
- **Automated Vehicles:** Enhancing proximity detection systems.
- **Industrial Safety:** Monitoring dangerous zones in factories or warehouses.
- **Smart Home Systems:** Integrating with home automation for enhanced security.

9. Challenges and Solutions

The development of the Servo-Controlled Obstacle Detection Radar presented several challenges, ranging from sensor calibration issues to synchronization between hardware and software components. Below is a detailed analysis of these challenges and the solutions implemented to address them effectively.

Challenge 1: Sensor Calibration

The ultrasonic sensor occasionally produced inconsistent distance readings due to environmental noise and timing mismatches.

Solution: Sensor timing was optimized by adjusting delays between pulse generation and echo reception. Filtering mechanisms, such as averaging multiple readings, were implemented to eliminate outliers. Additionally, the sensor was securely mounted to minimize vibration-induced errors.

Challenge 2: Real-Time Feedback

Synchronizing sensor readings with the servo motor's movement was difficult, leading to skipped or outdated data.

Solution: Proper delays were added to ensure the sensor completed readings before the servo moved to the next position. The Arduino code was streamlined to balance sensor measurement, data processing, and feedback generation efficiently.

Challenge 3: Visualization Sync

Data transfer between the Arduino and the Processing GUI occasionally resulted in delays or mismatches, disrupting the radar visualization.

Solution: Robust serial communication was implemented, with consistent data formatting and buffering until complete messages were received. Error-handling

mechanisms ensured only valid data was visualized, improving synchronization and reliability.

10. Future Enhancements

Potential improvements for the project could include:

- Implementing multiple sensors for 3D object detection.
- Adding Wi-Fi or Bluetooth connectivity for remote monitoring.
- Incorporating machine learning for intelligent threat assessment.
- Developing a mobile app interface for easy control and monitoring.
- Enhancing the range and accuracy of the detection system.

11. Conclusion

The **Servo-Controlled Obstacle Detection Radar** exemplifies how **cost-effective hardware** can be integrated with **efficient programming** to create a reliable monitoring system. By leveraging the ultrasonic sensor for **distance measurement,** the servo motor for **dynamic scanning,** and a Processing-based graphical interface for **real-time visualization**, the system effectively detects and highlights obstacles in its surroundings. The combination of auditory and visual alerts further enhances its functionality, making it a comprehensive solution for **obstacle detection.**

This project underscores its potential for **scalability** and **adaptability** in various real-world applications, such as robotics, safety systems, and security monitoring.

The use of **accessible components** and simple yet **efficient programming** demonstrates its practicality for implementation in **resource-constrained environments**. Overall, the project serves as a valuable foundation for developing more sophisticated monitoring systems tailored to specific **industrial** or **domestic needs.**

12. References

Arduino MKR Vidor 4000 Datasheet:
Official documentation providing detailed specifications for the Arduino MKR Vidor 4000 microcontroller.

https://docs.arduino.cc/hardware/mkr-vidor-4000

Ultrasonic Sensor HC-SR04 Datasheet:
Comprehensive datasheet for the HC-SR04 ultrasonic sensor, explaining its functionality and applications.

https://components101.com/sensors/ultrasonic-sensor-working-pinout-datasheet

Servo Motor Basics:
A guide on how servo motors work, including their control and practical uses in projects.

https://docs.arduino.cc/learn/electronics/servo-motors/

Processing Programming Environment Documentation:
Official documentation for the Processing language, offering tutorials and API references.

https://processing.org/reference/

I2C-Based LCD Displays Tutorial:
A step-by-step tutorial on interfacing I2C-based LCD displays with Arduino.

https://www.geeksforgeeks.org/how-to-interface-i2c-lcd-display-with-arduino/