ULTRASONIC RADAR SYSTEM

PREPARED BY

YASMINE IBRAHIM 221007524

IBRAHIM AHMED TOUNY 221004678

PREPARED FOR

Eng. Ahmed Morsy

Project Objective

This project aims to develop a fully functional ultrasonic radar system enhanced with additional features to improve usability and performance. The system incorporates joystick-based angle control to allow precise directional scanning, as well as a buzzer alarm that is triggered when an object is detected within a predefined distance threshold. These added functionalities enhance both the interactivity and reliability of the system

Project Description

The ultrasonic radar system incorporates many features that include the main factors of data acquisition as well as the visual feedback, data collection and storage. The main function of the system highlights the combination of a servo motor (SG90) that rotates through 180 degrees and an ultrasonic sensor on top of it. System was built using two interconnected Arduinos, MEGA and UNO, MEGA is responsible for collecting data from sensors and control
(Ultrasonic and joystick module), UNO is responsible for controlling the actuators (servo motor and buzzer).
On the side of data collection and processing, Arduino MEGA sends collected data to a specific software (Processing IDE) using serial communication (UART) in order to be processed and displayed on a 180 degrees radar visualization, with clear distance and object detection display, the programming language used in Processing IDE is python, MEGA sends the data to UNO using UART protocol, for Arduino uno to start acting on the actuators based on the defined code in ARDUINO IDE.
For data collection and storage, Arduino Uno sends the data received from sensors and actuators to a python file in order to save it and log it in a n SQLite database. Data processing was done through both Arduinos to prevent COM PORT clashes.
The system has 2 modes , Manual and Auto , the default is manual , to switch between modes the joystick module must be pressed down to switch to the other mode , the Manual mode allows the joystick module to take control while the auto mode lets the servo motor rotate continuously over the 180 degrees.

Sensors used:
Ultrasonic sensor

User control by:
Joystick module

Actuators used:
Servo motor(SG90)
Buzzer

Used IDE:
Processing
Visual studio code
Arduino
SQLite
Proteus

Programming languages used:
Python
C language

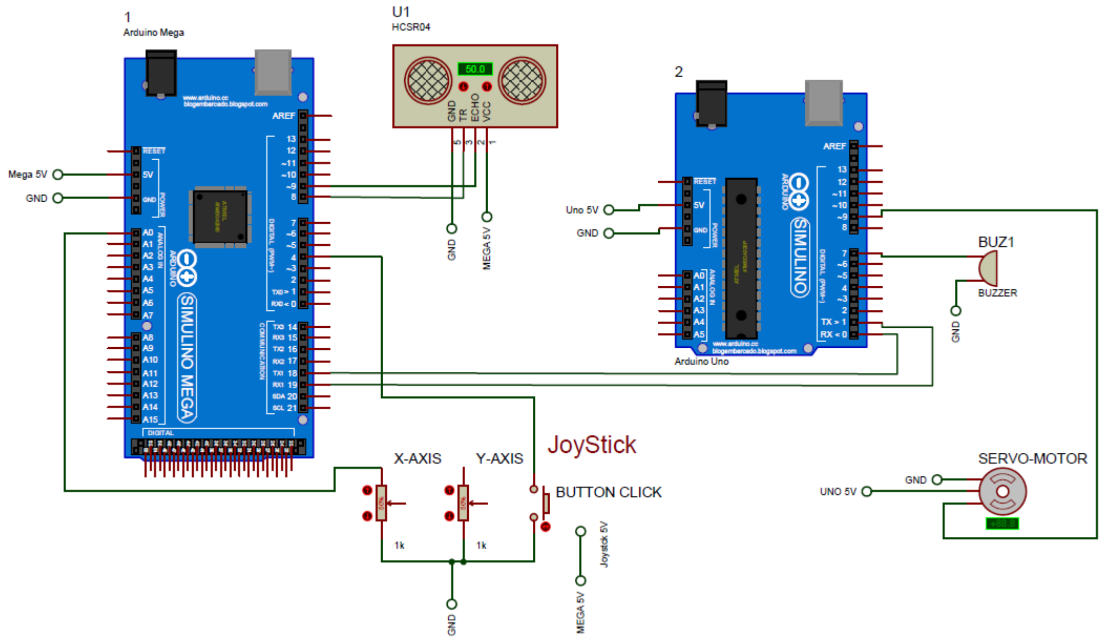
Hardware Components List

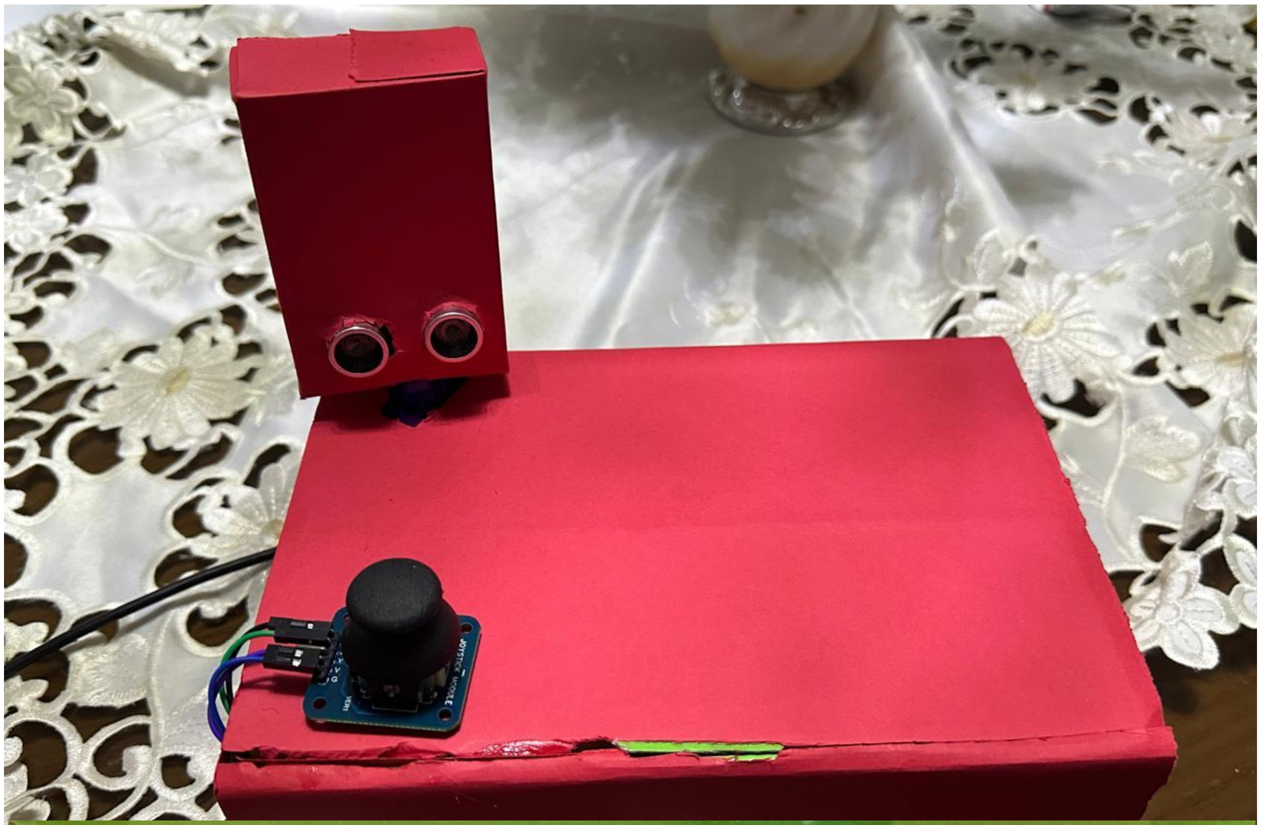
Servo Motor (SG90)
Sound Buzzer
Ultrasonic Sensor
Joystick Module
Arduino Uno
Arduino Mega
Jumpers
Breadboard

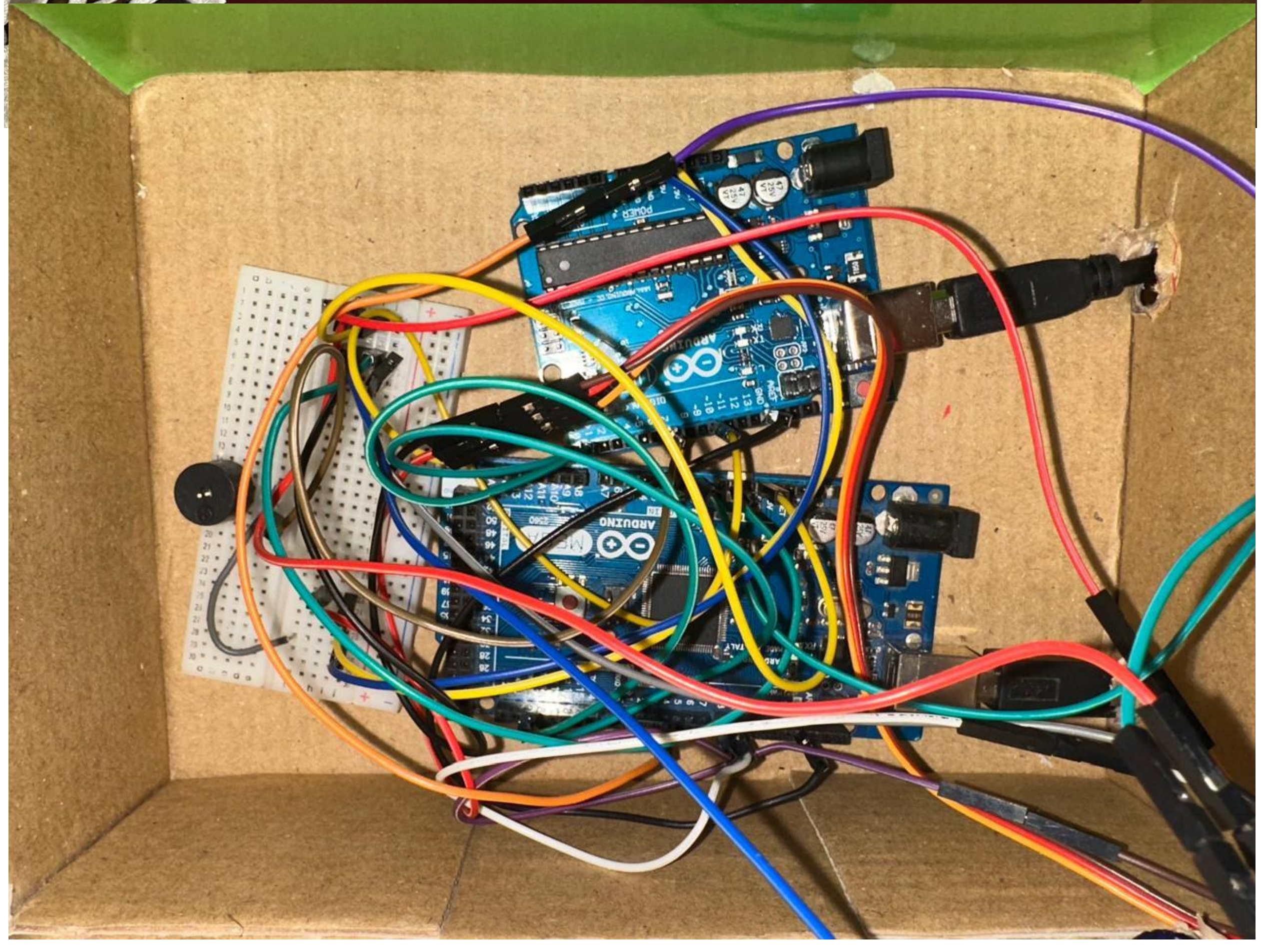
Design

The system is designed to detect distances up to 100m , buzzer is thrown off when any object is detected within 30 meters of the diameter of the ultrasonic sensor , angle is increased 10 degrees at a time on both manual and automatic mode

Circuit Diagram







Connection procedure

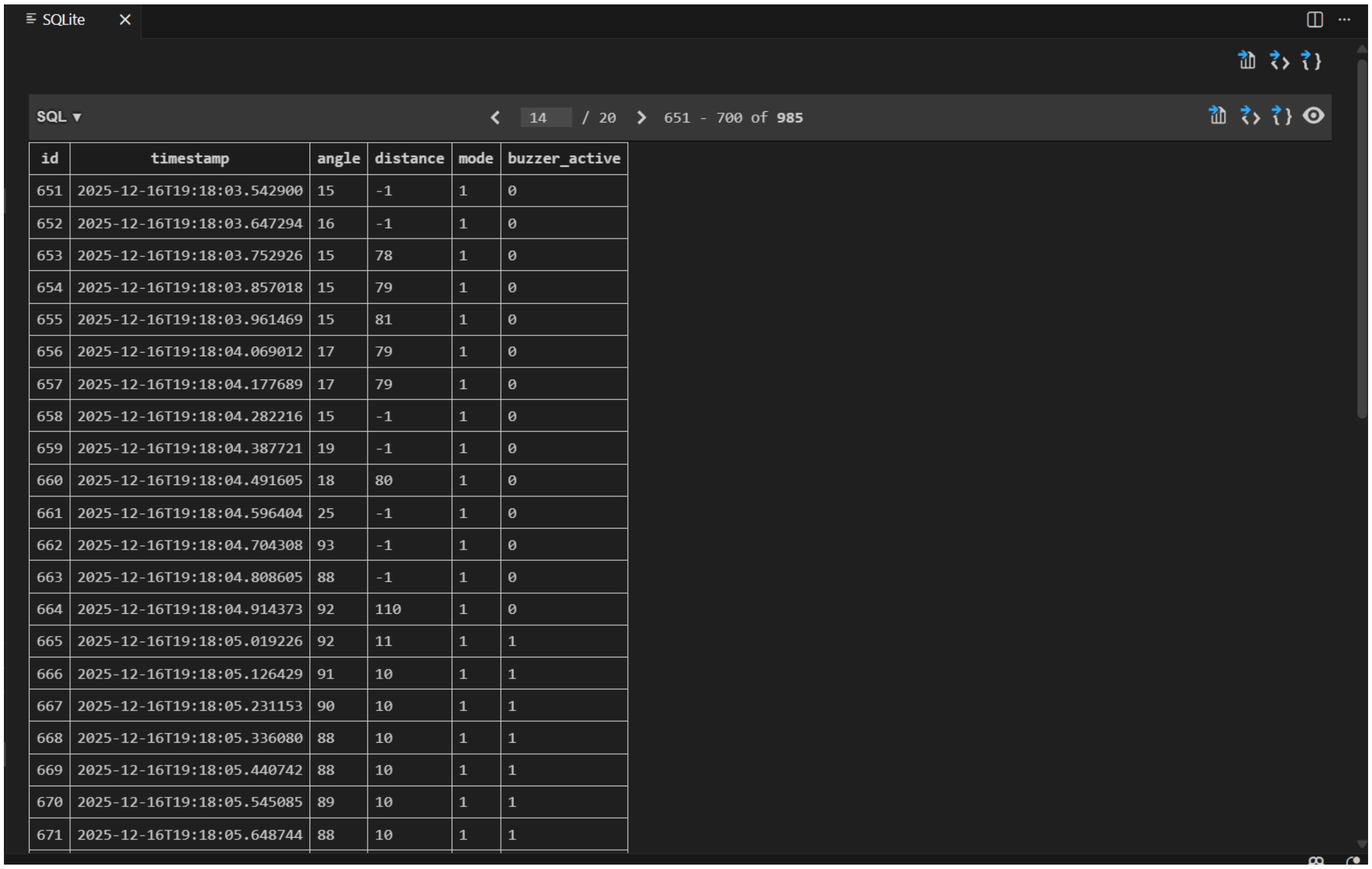
- Common ground between 2 Arduinos
- RX1 in Arduino Mega connected to TX in Arduino Uno
- TX1 in Arduino Mega connected to RX in Arduino Uno
- Ultrasonic Sensor (HC-SR04):
    - +VE pin to the Arduino Mega 5V
    - -VE pin to the GND
    - Echo pin to Pin 8 in the Arduino Mega
    - Trig pin to Pin 9 in the Arduino Mega
- Joystick Module:
    - +VE pin to the Arduino Mega 5V
    - -VE pin to GND
    - X-axis pin to pin A0 in the Arduino Mega
    - SW pin to Pin 4 in the Arduino Mega
    - Y-axis not connected
- Servo Motor (SG-90):
    - +VE pin to the Arduino Uno 5V
    - -VE pin to the GND
    - Control pin to Pin 9 in the Arduino Uno
- Buzzer:
    - +VE pin connected to Pin 7 in the Arduino Uno
    - -VE pin connected to GND

Database

Data storage is a crucial part of data acquisition systems. Our ultrasonic system stores the data collected from all sensors, it has real time data logging of all system operations , as will as histrorical tracking of objects, distance , and mode the system is operating in, which helps in debugging and performance monitoring overtime
Table: radar_logs

Database Schema:

- **id**: INTEGER PRIMARY KEY AUTOINCREMENT - Unique identifier for each log entry
- **timestamp**: TEXT - Date and time of the reading (ISO format)
- **angle**: INTEGER - Servo angle position (15° to 165°)
- **distance**: INTEGER - Object distance in centimeters (from ultrasonic sensor)
- **mode**: INTEGER - Operation mode (0 = Automatic, 1 = Manual)
- **buzzer_active**: INTEGER - Buzzer status (0 = Off, 1 = On when distance < 30cm)



id	timestamp	angle	distance	mode	buzzer_active
651	2025-12-16T19:18:03.542900	15	-1	1	0
652	2025-12-16T19:18:03.647294	16	-1	1	0
653	2025-12-16T19:18:03.752926	15	78	1	0
654	2025-12-16T19:18:03.857018	15	79	1	0
655	2025-12-16T19:18:03.961469	15	81	1	0
656	2025-12-16T19:18:04.069012	17	79	1	0
657	2025-12-16T19:18:04.177689	17	79	1	0
658	2025-12-16T19:18:04.282216	15	-1	1	0
659	2025-12-16T19:18:04.387721	19	-1	1	0
660	2025-12-16T19:18:04.491605	18	80	1	0
661	2025-12-16T19:18:04.596404	25	-1	1	0
662	2025-12-16T19:18:04.704308	93	-1	1	0
663	2025-12-16T19:18:04.808605	88	-1	1	0
664	2025-12-16T19:18:04.914373	92	110	1	0
665	2025-12-16T19:18:05.019226	92	11	1	1
666	2025-12-16T19:18:05.126429	91	10	1	1
667	2025-12-16T19:18:05.231153	90	10	1	1
668	2025-12-16T19:18:05.336080	88	10	1	1
669	2025-12-16T19:18:05.440742	88	10	1	1
670	2025-12-16T19:18:05.545085	89	10	1	1
671	2025-12-16T19:18:05.648744	88	10	1	1

GUI/Dashboard

The Processing-based graphical user interface (GUI) visualizes real-time data from the ultrasonic radar system transmitted over a serial connection. Its primary purpose is to display the **scanning angle**, **detected object distance**, and **radar sweep** in an intuitive, radar-like format.

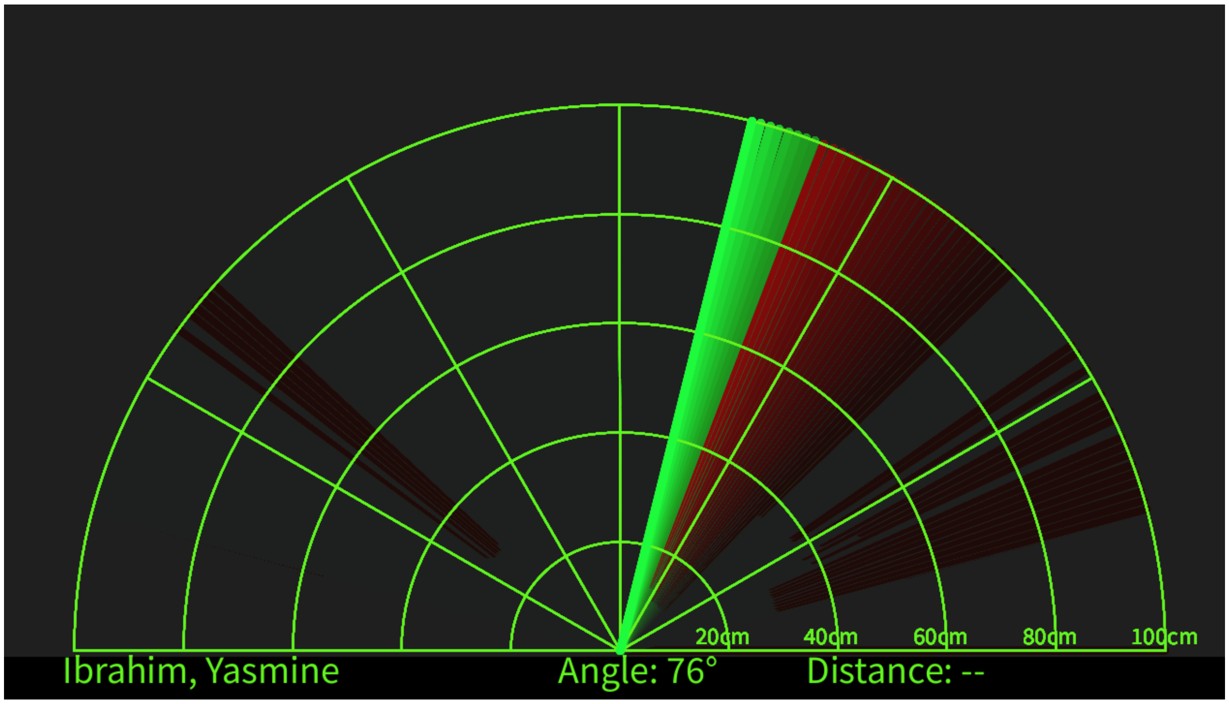
The GUI listens on COM6 at 9600 baud for incoming serial data from the Arduino uno. Each data packet is expected in the format of (angle, distance). The code parses this stream, extracts the angle and distance, and constrains the distance to a maximum of 100 cm to match the radar's display scale. A distance value of -1 is treated as no echo / no object detected.

The radar is rendered as a semi-circular (180°) scan, centered at the bottom of the window with Five concentric arcs that represent distance intervals (each corresponding to 20 cm). Radial lines every 30° provide angular reference, mimicking a real radar scope.

A bright green rotating line represents the current ultrasonic sensor direction. Its rotation angle directly corresponds to the received angle value, giving a live visualization of the servo or joystick controlled scan

When an object is detected within range, a red line is drawn from the detected point outward, indicating the object's position along the current angle. If no echo is detected, the GUI uses an orange color to indicate the absence of a valid reflection. The detected distance is mapped proportionally to the radar radius to ensure accurate representation.

A lower information bar displays some information that may be useful to the inspector such as the current angle (degrees), Measured distance (cm) or -- if no object is detected, Range status ("In Range" or "Out of Range"), Distance labels (20 cm increments) are shown along the radar arc .Names are displayed for project identification.



Code

#Arduino Mega

```
const int trigPin = 8;

const int echoPin = 9;


const int joyX = A0;

const int joySW = 4;


int angle = 90;

int dir = 1;

bool manualMode = false;

bool lastSW = HIGH;


long duration;

int distance;


// to fix manual mode

unsigned long lastManualSend = 0;

const unsigned long MANUAL_INTERVAL = 20; //
20 ms = servo-safe



int calculateDistance() {

  digitalWrite(trigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW)

  duration = pulseIn(echoPin, HIGH, 25000);

  if (duration == 0) return -1;

  return duration * 0.034 / 2;

}


void setup() {

  pinMode(trigPin, OUTPUT);

  pinMode(echoPin, INPUT);

  pinMode(joySW, INPUT_PULLUP);


  Serial.begin(9600);    // → Processing

  Serial1.begin(115200);  // → Arduino UNO

}


void loop() {


  bool sw = digitalRead(joySW);

  if (lastSW == HIGH && sw == LOW) {

    manualMode = !manualMode;

    delay(200);

  }

  lastSW = sw;


  if (manualMode) {

    int x = analogRead(joyX);
    angle = map(x, 0, 1023, 15, 165);
  } else {
    angle += dir;
    if (angle >= 165 || angle <= 15) dir = -dir;
  }

  distance = calculateDistance();

  // → Processing
  Serial.print(angle);
  Serial.print(",");
  Serial.print(distance);
  Serial.print(".");

  if (!manualMode) {
    // AUTO mode → same behavior as before
    Serial1.print(angle);
    Serial1.print(",");
    Serial1.print(distance);
    Serial1.print(",");
    Serial1.print("0");  // AUTO = 0
    Serial1.print(".");
    delay(30);  // unchanged AUTO speed
  }
  else {
    // MANUAL mode → rate limited
    unsigned long now = millis();
    if (now - lastManualSend >=
MANUAL_INTERVAL) {
      lastManualSend = now;

      Serial1.print(angle);
      Serial1.print(",");
      Serial1.print(distance);
      Serial1.print(",");
      Serial1.print("1");  // MANUAL = 1
      Serial1.print(".");
    }
  }

  if(!manualMode)
    delay(30);
}
```

#Arduino Uno

```
#include <Servo.h>


#define SERVO_PIN  10
#define BUZZER_PIN 7


Servo myServo;
#define BUF_SIZE 32
char rxBuffer[BUF_SIZE];
byte rxIndex = 0;
int angle = 90;
int distance = -1;
int mode = 0; // 0 = AUTO, 1 = MANUAL


void setup() {
  Serial.begin(115200);   // UART from Mega
  myServo.attach(SERVO_PIN);
  pinMode(BUZZER_PIN, OUTPUT);
}


void loop() {
  readSerial();
}


void readSerial() {
  while (Serial.available()) {
    char c = Serial.read();
    if (c == '.') {
      rxBuffer[rxIndex] = '\0';
      parsePacket(rxBuffer);
      rxIndex = 0;              // reset buffer
    }
    else {
      if (rxIndex < BUF_SIZE - 1) {
        rxBuffer[rxIndex++] = c;
      } else {
        // buffer overflow → reset safely
        rxIndex = 0;
      }
    }
  }
}


void parsePacket(char *packet) {


  char *comma1 = strchr(packet, ',');
  if (!comma1) return;


  *comma1 = '\0';


  char *comma2 = strchr(comma1 + 1, ',');
  if (!comma2) return;


  *comma2 = '\0';
  int newAngle = atoi(packet);
  int newDistance = atoi(comma1 + 1);
  int newMode = atoi(comma2 + 1);
  if (newAngle < 15 || newAngle > 165) return;
  angle = newAngle;
  distance = newDistance;
  mode = newMode;

  myServo.write(angle);

  bool buzzerActive = false;
  if (distance > 0 && distance < 30) {
    tone(BUZZER_PIN, 2000);
    buzzerActive = true;
  } else {
    noTone(BUZZER_PIN);
    buzzerActive = false;
  }
  Serial.print(angle);
  Serial.print(",");
  Serial.print(distance);
  Serial.print(",");
  Serial.print(mode);
  Serial.print(",");
  Serial.print(buzzerActive ? 1 : 0);
  Serial.println();
}
```

# Processing

```
import processing.serial.*;
import java.awt.event.KeyEvent;
import java.io.IOException;

Serial myPort;

String distance = "";
String data = "";
String noObject;
String angle = "";

float pixsDistance;
int iAngle, iDistance;
int index1 = 0;

float radarRadius;

void setup() {

  size(1280, 720);
  smooth();

  radarRadius = width * 0.44;

  myPort = new Serial(this, "COM6", 9600);
  myPort.bufferUntil('.');
}
```

```
void draw() {

  noStroke();
  fill(0, 4);
  rect(0, 0, width, height - height * 0.065);

  fill(98, 245, 31);
  drawRadar();
  drawLine();
  drawObject();
  drawText();
}

void serialEvent(Serial myPort) {

  data = myPort.readStringUntil('.');
  data = data.substring(0, data.length() - 1);

  index1 = data.indexOf(',');

  angle = data.substring(0, index1);
  distance = data.substring(index1 + 1);

  iAngle = int(angle);
  iDistance = min(int(distance), 100);
}

void drawRadar() {

  pushMatrix();
  translate(width / 2, height - height * 0.074);
  noFill();
  strokeWeight(2);
  stroke(98, 245, 31);

  for (int i = 1; i <= 5; i++) {
    float r = radarRadius * i / 5.0;
    arc(0, 0, r * 2, r * 2, PI, TWO_PI);
  }

  for (int a = 0; a <= 180; a += 30) {
    line(0, 0,
         radarRadius * cos(radians(a)),
        -radarRadius * sin(radians(a)));
  }

  popMatrix();
}

void drawObject() {

  pushMatrix();
  translate(width / 2, height - height * 0.074);
  strokeWeight(9);

  if (iDistance == -1) {
    stroke(255, 165, 0); // ORANGE → no echo
  } else {
    stroke(255, 10, 10); // RED → valid object
  }
  pixsDistance = map(constrain(iDistance, 0, 100),
0, 100, 0, radarRadius);

  if (iDistance < 100 && iDistance != -1) {
    line(
      pixsDistance * cos(radians(iAngle)),
     -pixsDistance * sin(radians(iAngle)),
      radarRadius * cos(radians(iAngle)),
     -radarRadius * sin(radians(iAngle))
    );
  }
  popMatrix();
}

void drawLine() {
  pushMatrix();
  translate(width / 2, height - height * 0.074);
  strokeWeight(9);
  stroke(30, 250, 60);
  line(0, 0,
       radarRadius * cos(radians(iAngle)),
      -radarRadius * sin(radians(iAngle)));
  popMatrix();
}

void drawText() {

  pushMatrix();
  noObject = (iDistance == -1 || iDistance >= 100)
? "Out of Range" : "In Range";
  fill(0);
  noStroke();
  rect(0, height - height * 0.0648, width, height);

  fill(98, 245, 31);
  textSize(25);

  for (int i = 1; i <= 5; i++) {
    text(i * 20 + "cm",
         width / 2 + radarRadius * i / 5.0 - 35,
         height - height * 0.0833);
  }

  textSize(40);
  text("Ibrahim, Yasmine", width * 0.05, height -
height * 0.0277);
  text("Angle: " + iAngle + "°", width * 0.45,
height - height * 0.0277);

  if (iDistance != -1) {
    text("Distance: " + iDistance + " cm", width *
0.65, height - height * 0.0277);
  } else {
    text("Distance: --", width * 0.65, height - height
* 0.0277);
  }

  popMatrix();
}
```

# Python

```
import serial

import sqlite3

from datetime import datetime

PORT = "COM5"     # CHANGE to your Arduino
UNO COM port

BAUD = 115200


try:

    ser = serial.Serial(PORT, BAUD, timeout=1)

    print(f"Connected to {PORT} at {BAUD}
baud")

except Exception as e:

    print(f"Error connecting to serial port: {e}")

    exit(1)

conn = sqlite3.connect("radar_logs.db")

cursor = conn.cursor()


cursor.execute("""

CREATE TABLE IF NOT EXISTS radar_logs (

    id INTEGER PRIMARY KEY
AUTOINCREMENT,

    timestamp TEXT,

    angle INTEGER,

    distance INTEGER,

    mode INTEGER,

    buzzer_active INTEGER

)

""")

conn.commit()

print("Database initialized: radar_logs.db")

print("Logging started...\n")

try:

    while True:

        line =
ser.readline().decode(errors="ignore").strip()


        if not line:

            continue


        try:

            # Expected format:
angle,distance,mode,buzzer

            parts = line.split(",")


            if len(parts) != 4:

                print(f"Invalid format: {line}")

                continue


            angle = int(parts[0])

            distance = int(parts[1])

            mode = int(parts[2])

            buzzer = int(parts[3])


            timestamp = datetime.now().isoformat()

            cursor.execute(

                "INSERT INTO radar_logs (timestamp,
angle, distance, mode, buzzer_active) VALUES
(?, ?, ?, ?, ?)",
                (timestamp, angle, distance, mode,
buzzer)
            )

            conn.commit()

            mode_str = "MANUAL" if mode == 1
else "AUTO"
            buzzer_str = "ON" if buzzer == 1 else
"OFF"
            distance_str = f"{distance}cm" if distance
!= -1 else "OUT OF RANGE"

            print(f"[{timestamp}] Angle: {angle}° |
Distance: {distance_str} | Mode: {mode_str} |
Buzzer: {buzzer_str}")

        except ValueError as e:
            print(f"Parse error: {line} → {e}")

except KeyboardInterrupt:
    print("\n\nLogging stopped by user")

finally:
    ser.close()
    conn.close()
    print("Resources cleaned up")
```