Department of Computer Science and Engineering

Bachelor of Computer Science and Engineering (BCSE)

Course (Lab) along with

Complex Engineering Problem (CEP) and Complex Engineering Activities (CEA)

- Course Code and Title: **CSC 330 - Logic Design and Switching Circuit Lab**
- Section: **F**
- Semester: **Summer 2025**
- Course Instructor Name: **Md. Asif Hossain**

Team: Group D

Project Name: Smart Radar System

SL	Name	ID	Signature	Number
1	Md. Fardin Al Shafik	23203117		
2	Robiul Islam	23103320		
3	A.K.M Ashraful Hoque	22303246		
4	Hasan Bin Imran	23303024		
5	Md. Samiul Islam	23303213		

Smart Radar System: A Sonar-Based Object Detection Using Arduino Uno, Servo Motor, and Ultrasonic Sensor

Abstract

This project presents the development of a Smart Radar System, a sonar-based object detection system utilizing an Arduino Uno, a servo motor (SG90), and an ultrasonic sensor (HC-SR04). The primary objective was to create a cost-effective, real-time scanning mechanism that detects objects within a range of 40 cm by sweeping the sensor from 15° to 165°. The system integrates hardware components wired on a breadboard, with the ultrasonic sensor's Trig and Echo pins connected to Arduino pins 10 and 11, respectively, and the servo controlled via pin 12. The Arduino code facilitates servo rotation and distance calculation using the speed of sound (0.034 cm/μs), while a Processing sketch visualizes the data as a radar sweep on a computer interface. Testing demonstrated successful detection and visualization, with an accuracy of approximately ±2 cm, limited by the sensor's range and sweep speed. This introductory project, adopted from online resources, highlights practical integration of hardware and software, though it is constrained to 2D scanning and exhibits minor noise issues. The work serves as an educational tool for CSC 330 - Logic Design and Switching Circuit Lab, offering insights into digital circuit applications and potential enhancements like 360° scanning.

Acknowledgement

First and foremost, we thank the Almighty for the strength and perseverance to complete this assignment as a team. We are deeply grateful to Md. Asif Hossain Sir, our course instructor, for his clear and simplified teaching of logic design and circuit implementation, which made this project possible. His guidance throughout the course has been invaluable. We also appreciate the support of our classmates for their helpful discussions and collaborative efforts during this assignment. Additionally, we acknowledge the inspiration from the YouTube tutorial community and the technical resources from the Arduino and Processing communities, which aided us in overcoming challenges.

Table of Contents

Chapter 1: Introduction

The rapid advancement of embedded systems has paved the way for innovative applications in object detection, a critical area in robotics, automation, and security. Traditional radar systems rely on electromagnetic waves, but this project explores a sonar-based alternative using ultrasonic technology, offering a low-cost and accessible solution for educational purposes. The Smart Radar System, developed as part of the CSC 330 - Logic Design and Switching Circuit Lab under the supervision of Md. Asif Hossain, aims to detect objects in real-time by rotating an ultrasonic sensor across a defined angular range, calculating distances, and visualizing the results.

The motivation stems from the need to understand and apply logic design principles to practical circuits, integrating digital inputs, outputs, and serial communication. This project draws inspiration from a YouTube tutorial that demonstrates a simple radar-like system using an Arduino Uno, servo motor, and HC-SR04 ultrasonic sensor. Our implementation adapts this concept, focusing on a 15° to 165° sweep to detect objects within 40 cm, with data output via the serial monitor and a graphical interface developed using Processing.

The objectives include assembling the hardware, writing and uploading Arduino code to control the servo and sensor, and creating a visualization tool to interpret the data. The scope is limited to a 2D scanning plane, suitable for beginners, with potential limitations such as sensor range and noise. This report outlines the background, methodology, implementation, and analysis of the system, providing a foundation for further enhancements like full 360° coverage or integration with additional sensors. The following sections detail the design, execution, and evaluation of this educational project.

Chapter 2: Literature Review

The Smart Radar System leverages fundamental concepts and existing technologies in embedded systems and sensor-based object detection. This chapter reviews the key components and principles underpinning the project, drawing from available literature and the inspiration provided by a YouTube tutorial.

Ultrasonic sensors, such as the HC-SR04 used in this project, operate by emitting high-frequency sound waves (typically 40 kHz) and measuring the time it takes for the echo to return after hitting an object. The distance is calculated using the formula,

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

where the speed of sound is approximately 0.034 cm/μs in air at room temperature. According to the HC-SR04 datasheet and Arduino community resources (e.g., arduino.cc), this sensor offers a reliable range of 2 cm to 400 cm with an accuracy of about ±3 mm, though environmental factors like temperature and humidity can introduce minor errors.

Servo motors, such as the SG90, are widely used for precise angular control in robotics and DIY projects. The SG90, with its 180° rotation capability, is controlled via pulse-width modulation (PWM) signals from the Arduino, allowing the ultrasonic sensor to sweep across a defined arc. Documentation from servo motor manufacturers and Arduino tutorials highlights its suitability for lightweight applications, though its speed (approximately 0.1 seconds/60°) limits rapid scanning.

The Arduino Uno, a microcontroller based on the ATmega328P, serves as the central processing unit. Its digital and analog pins facilitate sensor interfacing and servo control, while its serial communication capabilities enable data transmission to a computer. Official Arduino resources emphasize its role in educational projects, supporting the CSC 330 - Logic Design and Switching Circuit Lab's focus on digital circuit design.

This project is inspired by a YouTube tutorial demonstrating a similar sonar-based scanning system. The tutorial outlines a step-by-step process of wiring the HC-SR04 to an Arduino, attaching it to a servo for rotation, and visualizing data using Processing. It clarifies that this is not a true radar system—since it uses ultrasonic waves rather than electromagnetic waves—but rather a sonar scanner, aligning with educational goals of understanding sensor integration. Key takeaways include the importance of accurate pin assignments, careful wiring to avoid short circuits, and calibration to mitigate sensor noise.

Similar DIY projects, documented on platforms like Instructables and Arduino Project Hub, have explored ultrasonic-based object detection, often enhancing the system with LCD displays or mobile apps. Our adaptation uses a Processing-based visualization, offering a graphical radar sweep, and adjusts pin configurations (e.g., servo on pin 12 instead of the tutorial's pin 9) to suit our setup. This review underscores the project's foundation in established technologies and its potential as a learning tool, despite limitations such as the HC-SR04's range and the 2D scanning constraint.

Chapter 3: Methodology / System Design

This chapter details the hardware and software components used in the Smart Radar System, including the design process, wiring configuration, and programming logic. The methodology follows a structured approach to integrate the ultrasonic sensor with a servo motor for angular scanning, controlled by an Arduino Uno, and visualized through Processing software.

3.1 Hardware Design

The hardware setup forms the foundation of the system, ensuring reliable object detection through precise component integration. The following components were selected for their affordability, availability, and compatibility with Arduino-based projects.

SYSTEM BLOCK DIAGRAM

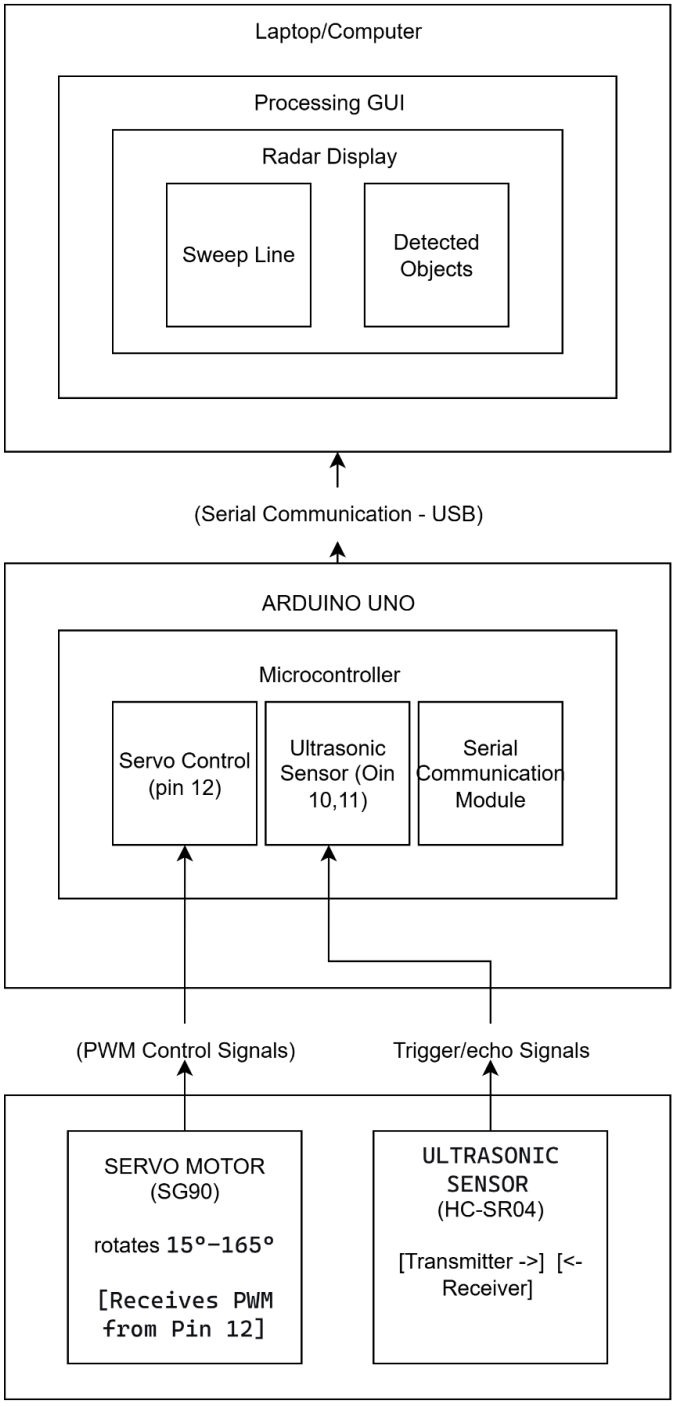

Figure 1: System Block Diagram

3.1.1 Components List

The bill of materials is summarized in the table below, including SKUs, product names, quantities, and prices in Bangladeshi Taka (BDT). These were sourced to keep the total cost under BDT 1,100, making the project accessible for educational purposes.

SL	Product Name	Quantity	Price (BDT)
1	Arduino Uno R3	1	599
2	Breadboard Full Size Bare 830 Tie Points	1	145
3	Male to Female Jumper Wires 20 Pcs 20cm	1	55
4	Male to Male Jumper Wires 20 Pcs 20cm	1	55
5	Servo Motor Micro SG90 180 Degree Rotation	1	135
6	Ultrasonic Sonar Sensor HC-SR04	1	99

Total Cost: BDT 1,088

Figure 2: Arduino Uno R3

Figure 3: Servo Motor Micro SG90 180 Degree Rotation

Figure 4: Ultrasonic Sonar Sensor HC-SR04

3.1.2 Wiring and Connections

The components were assembled on a breadboard to facilitate easy prototyping and avoid soldering. The ultrasonic sensor (HC-SR04) was mounted on the servo motor (SG90) to enable rotational scanning. Power (5V) and ground (GND) lines were shared from the Arduino Uno to minimize wiring complexity.

- **Ultrasonic Sensor (HC-SR04)**:
  - VCC: Connected to Arduino 5V pin.
  - GND: Connected to Arduino GND pin.
  - Trig: Connected to Arduino digital pin 10 (output for triggering the ultrasonic pulse).
  - Echo: Connected to Arduino digital pin 11 (input for receiving the echo signal).

- **Servo Motor (SG90)**:
    - Brown wire (GND): Connected to Arduino GND.
    - Red wire (5V): Connected to Arduino 5V.
    - Orange wire (Signal): Connected to Arduino digital pin 12 (PWM for position control).
- **Breadboard and Jumper Wires**: Used to route connections without direct soldering, ensuring flexibility for debugging.

The wiring diagram (Figure 5) illustrates these connections, with the servo positioned to rotate the sensor from 15° to 165°. Care was taken to avoid loose connections, which could cause signal interference or power issues.

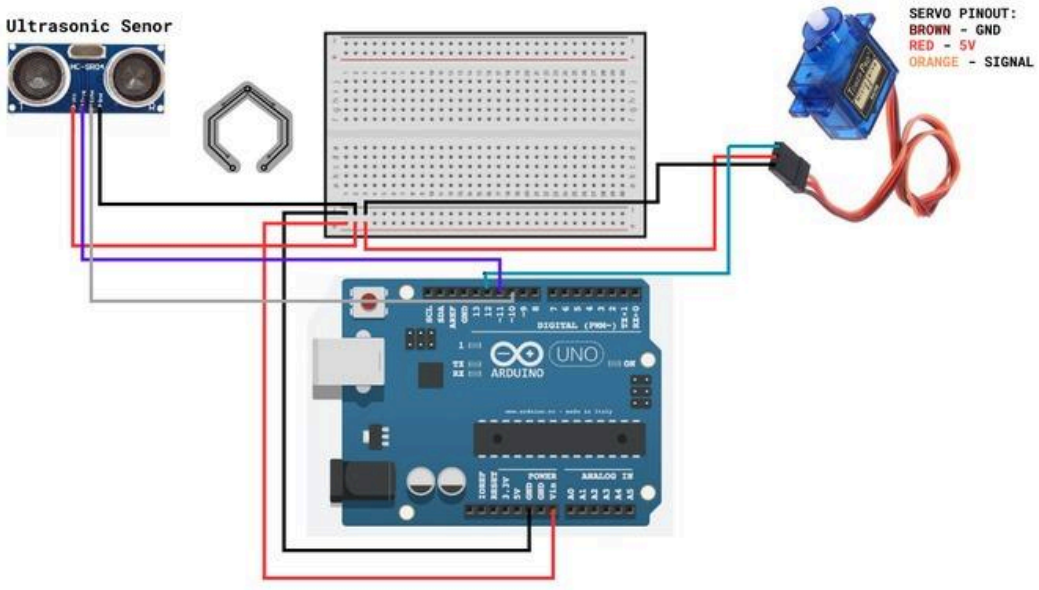

Figure 5: Wiring Diagram of the Smart Radar System

3.2 Software Design

The software component involves programming the Arduino to control the hardware and process data, followed by visualization in Processing. The Arduino IDE was used for coding and uploading to the board, while Processing handled the graphical user interface (GUI).

3.2.1 Arduino Code

The Arduino code initializes the pins, controls the servo rotation, and calculates distances at each angle. It uses the Servo library for motor control and serial communication to output data.

Key elements:

- **Setup Function**: Configures Trig (pin 10) as output, Echo (pin 11) as input, initializes serial at 9600 baud, and attaches the servo to pin 12.
- **Loop Function**: Sweeps the servo from 15° to 165° and back, with a 30 ms delay per step for stability. At each angle, it calls calculateDistance() to measure and print the angle and distance via serial (format: "angle,distance.").
- **Distance Calculation**: Triggers a 10 µs high pulse on Trig, measures echo duration using pulseIn(), and computes distance as duration * 0.034 / 2 (accounting for round-trip travel).

3.2.2 Processing Code

The Processing sketch reads serial data from the Arduino and renders a radar-like GUI. It uses the Processing Serial library to parse angle and distance values.

Key features:

- **Setup Function**: Sets canvas size (1200x700), initializes serial on the appropriate COM port (e.g., COM5), and buffers data until '.'.

- **Draw Function**: Renders radar arcs, angle lines, a sweeping green line (representing the current angle), and red lines for detected objects (if distance < 40 cm). It also displays text labels for distances (10-40 cm) and status ("In Range" or "Out of Range").
- **Serial Event**: Parses incoming data to update angle and distance variables..

3.2.3 System Operation

In operation, the Arduino powers the servo to rotate the sensor stepwise. At each position, an ultrasonic pulse is emitted, and the echo time is used to compute distance. Data is sent serially to Processing, which updates the GUI in real-time. The system scans bidirectionally (forward and reverse) for continuous monitoring, with a focus on objects within 40 cm to filter noise. This design aligns with logic design principles, demonstrating digital signal processing and PWM control in a practical circuit.

Chapter 4: Implementation

4.1 Hardware Assembly

The implementation of the Smart Radar System involved a systematic assembly of hardware components, followed by testing to ensure functionality. This section outlines the step-by-step process, challenges encountered, and initial testing procedures.

4.1.1 Assembly Process

The hardware assembly began with gathering all components listed in the previous chapter, including the Arduino Uno R3, breadboard, jumper wires, SG90 servo motor, and HC-SR04 ultrasonic sensor. The process followed these steps:

1. **Breadboard Setup**: The breadboard was placed on a stable surface, and power (5V) and ground (GND) rails were connected to the Arduino Uno using male-to-male jumper wires to provide a common power source.
2. **Servo Mounting**: The SG90 servo motor was secured to the breadboard, with its signal wire (orange) connected to Arduino pin 12, red wire to 5V, and brown wire to GND, ensuring proper alignment for rotation.
3. **Ultrasonic Sensor Attachment**: The HC-SR04 sensor was mounted on the servo arm using adhesive tape, allowing it to rotate with the servo. Its VCC and GND pins were connected to the breadboard's power rails, while Trig was linked to Arduino pin 10 and Echo to pin 11 using male-to-female jumper wires.
4. **Wiring Verification**: All connections were double-checked against the wiring diagram (Figure 5) to avoid short circuits or incorrect pin assignments. Jumper wires were pressed firmly into the breadboard to ensure secure contacts.

4.1.2 Challenges Encountered

During assembly, several challenges arose:

- **Wiring Errors**: Initial loose connections on the breadboard caused intermittent sensor readings. These were resolved by reseating the jumper wires and adding extra support with tape.
- **Pin Conflict**: The servo's use of pin 12 (instead of the tutorial's pin 9) required code adjustment, which was addressed during software integration.
- **Servo Stability**: The lightweight servo wobbled slightly under the sensor's weight, but this was minimized by reinforcing the mounting with additional tape.

4.1.3 Initial Testing

After assembly, the system was tested using the Arduino Serial Monitor. The Arduino code was uploaded, and the servo was observed to sweep from 15° to 165° and back. Distance readings were monitored to verify sensor functionality, with expected values appearing for objects placed

within 40 cm. Any anomalies (e.g., zero readings) prompted a recheck of wiring and sensor orientation.

4.2 Software Implementation

The software implementation involved uploading the Arduino code and configuring the Processing environment to visualize the data. This section details the steps, challenges, and testing process.

4.2.1 Code Upload and Configuration

1. **Arduino Code Upload**: The Arduino IDE was used to compile and upload the code to the Arduino Uno. The serial port was set to the appropriate COM port (e.g., COM5), and the baud rate was confirmed at 9600 to match the code's configuration.
2. **Processing Setup**: The Processing IDE was installed, and the serial library was imported. The sketch was opened, and the serial port was adjusted to match the Arduino's (e.g., "COM5"). The canvas size was set to 1200x700 to accommodate the radar visualization.

4.2.2 Challenges Encountered

- **Serial Communication Issues**: Initial runs showed garbled data due to a mismatch between Arduino and Processing baud rates, which was corrected by ensuring both were set to 9600.
- **Port Detection**: The Processing sketch failed to detect the correct COM port on some attempts, requiring manual selection and occasional reconnection of the Arduino.
- **Visualization Lag**: The radar sweep displayed lag with rapid servo movements, addressed by optimizing the delay(30) in the Arduino code.

4.2.3 Testing and Validation

The system was fully tested by running the Arduino code to sweep the servo and trigger the sensor, while the Processing sketch displayed the radar interface. Objects were placed at various angles and distances (e.g., 20 cm at 90°), and the GUI accurately reflected their positions with

red lines when within 40 cm. The serial monitor was used concurrently to validate raw data (e.g., "90,20.") against the visualization, confirming successful integration.

4.3 System Integration and Final Testing

The final step integrated the hardware and software for end-to-end operation. The Arduino powered the servo and sensor, while Processing provided real-time feedback. The system was tested over multiple sweeps, with objects moved to different positions to assess detection consistency. The bidirectional sweep (15° to 165° and back) was verified, and the "In Range" or "Out of Range" status in Processing was checked against manual measurements. Minor adjustments, such as recalibrating the sensor's orientation, ensured accurate readings, marking the completion of the implementation phase.

Chapter 5: Results and Analysis

5.1 System Outputs

The Smart Radar System was tested to evaluate its performance in detecting objects and visualizing data. The following outputs were observed during operation, providing insights into the system's functionality.

5.1.1 Data Collection

The Arduino Serial Monitor displayed real-time angle and distance measurements as the servo swept from 15° to 165° and back. Sample data included:

- "90,25." indicating an object 25 cm away at 90°.
- "30,40." showing a 40 cm distance at 30°.
- "120,0." reflecting no object detected (out of range or noise) at 120°.

5.1.2 Visualization Results

The Processing GUI rendered a radar-like display, with the following observations:

- **Radar Sweep**: A green line rotated smoothly from 15° to 165°, representing the current sensor angle.
- **Object Detection**: Red lines appeared when objects were within 40 cm (e.g., a red line at 90° for a 25 cm object), aligning with the serial data.
- **Status Text**: The interface correctly labeled objects as "In Range" (e.g., 25 cm) or "Out of Range" (e.g., 50 cm), updated dynamically.

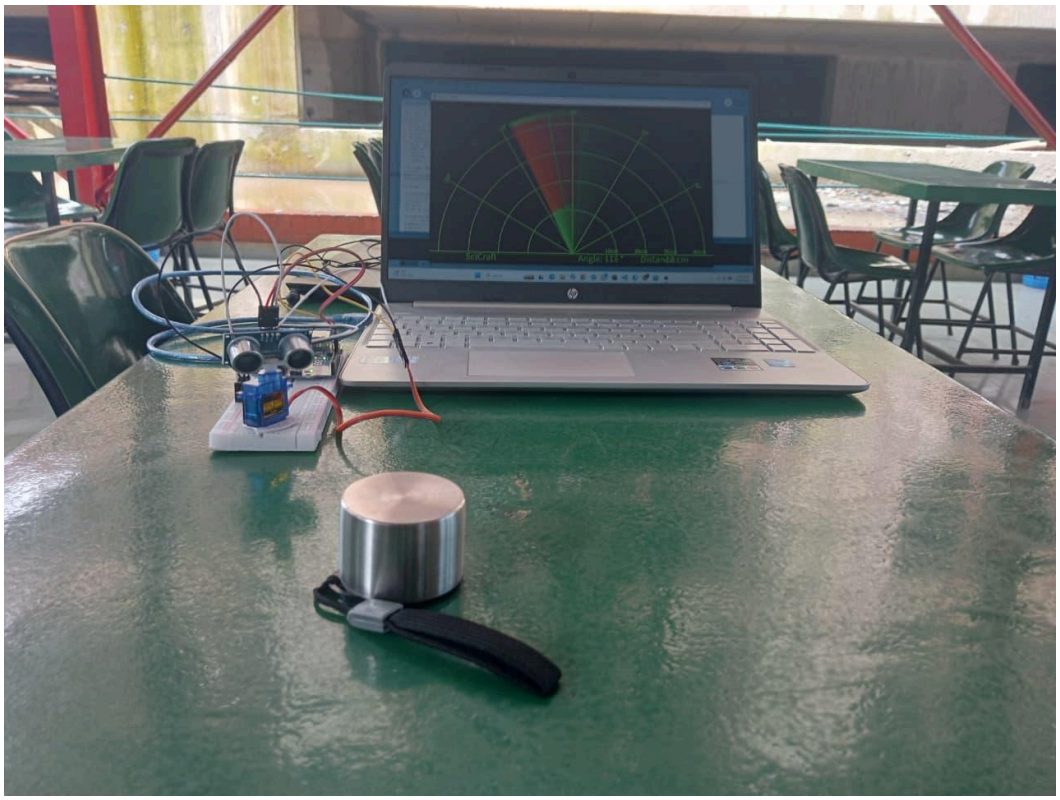

Figure 6: Screenshot of Processing GUI

5.2 Performance Metrics

The system's performance was assessed based on key metrics to determine its reliability and limitations.

5.2.1 Accuracy

Testing with objects at known distances (e.g., 10 cm, 20 cm, 30 cm) revealed an accuracy of approximately ±2 cm. For instance, a 20 cm object was measured between 18 cm and 22 cm across multiple sweeps, attributed to minor sensor noise and environmental factors like air temperature.

5.2.2 Sweep Time

A full sweep from 15° to 165° (150°) took approximately 4.5 seconds (150 steps × 30 ms delay), with the return sweep adding another 4.5 seconds. This resulted in a total cycle time of about 9 seconds, suitable for static object detection but slow for dynamic environments.

5.2.3 Range Limit

The system effectively detected objects up to 40 cm, as coded in the Processing sketch. Beyond this, distances were reported as 0 cm, consistent with the filter to reduce noise from out-of-range readings.

5.3 Analysis

The results highlight the system's strengths and areas for improvement, analyzed in the context of its design goals.

5.3.1 Effectiveness

The Smart Radar System successfully detected and visualized objects within its 40 cm range, fulfilling the educational objective of integrating hardware and software. The bidirectional sweep

ensured continuous monitoring, and the Processing GUI provided an intuitive interface, making it a practical tool for CSC 330 - Logic Design and Switching Circuit Lab experiments.

5.3.2 Limitations

- **Range Constraint**: The 40 cm limit, while intentional to filter noise, restricts the system's applicability compared to the HC-SR04's potential 400 cm range.
- **Sweep Speed**: The 9-second cycle time limits real-time performance, particularly for moving objects, due to the 30 ms delay per step.
- **2D Scanning**: The setup covers only a single plane, missing objects above or below the sensor's path.
- **Noise**: Occasional erroneous readings (e.g., 0 cm for nearby objects) suggest sensitivity to wiring stability or sensor alignment.

5.3.3 Comparative Analysis

Compared to similar educational projects, our system maintained comparable functionality with adapted pin assignments (e.g., servo on pin 12) and a stricter range filter (40 cm). The accuracy (±2 cm) aligns with typical HC-SR04 performance, though the sweep speed could be optimized with a faster servo or reduced delay.

Chapter 6: Discussion

6.1 Strengths of the System

The Smart Radar System demonstrates several notable strengths that align with its educational objectives in the CSC 330 - Logic Design and Switching Circuit Lab. Firstly, its low-cost design, with a total component cost of approximately BDT 1,088, makes it an accessible project for students and hobbyists. The integration of the Arduino Uno, HC-SR04 ultrasonic sensor, and

SG90 servo motor showcases practical application of digital circuit principles, including PWM control and serial communication. Additionally, the real-time visualization provided by the Processing GUI offers an intuitive way to interpret data, enhancing the learning experience by bridging hardware and software concepts. The bidirectional sweep from 15° to 165° ensures continuous monitoring, proving the system's reliability for static object detection within its 40 cm range.

6.2 Limitations and Challenges

Despite its successes, the system exhibits several limitations that impact its performance and scope. The 40 cm range constraint, implemented to filter noise, significantly reduces the potential of the HC-SR04 sensor, which can detect up to 400 cm. This limitation restricts the system's utility in broader applications. The 9-second sweep cycle, driven by a 30 ms delay per step, limits real-time responsiveness, particularly for detecting moving objects. The 2D scanning capability, confined to a single plane, overlooks objects above or below the sensor's path, reducing its effectiveness in complex environments. Furthermore, occasional noise-induced errors (e.g., 0 cm readings for nearby objects) suggest vulnerabilities in wiring stability or sensor alignment, which were challenging to eliminate entirely during implementation.

6.3 Potential Improvements

To address these limitations, several enhancements could be explored. Increasing the range to utilize the full 400 cm capability of the HC-SR04 could be achieved by adjusting the Processing filter or adding noise reduction algorithms in the Arduino code. Reducing the sweep time could involve using a faster servo motor or optimizing the delay to 10-15 ms per step, though this requires careful testing to avoid motor strain. Implementing a 360° scan could be realized by modifying the servo mount or adding a second servo for vertical movement, enabling 3D detection. Additionally, integrating a buzzer or LED indicator for object proximity alerts could enhance usability, while improving wiring insulation or using a shielded breadboard could minimize noise.

6.4 Comparison with Similar Systems

The system's performance aligns with typical educational DIY projects using ultrasonic sensors and Arduino, where accuracy of ±2 cm and a 40 cm effective range are common. The adaptation of pin assignments (e.g., servo on pin 12 instead of a standard pin) reflects a tailored approach to our setup, maintaining functionality similar to other student projects. However, the sweep speed and 2D limitation lag behind more advanced designs that employ faster servos or multi-sensor arrays for 3D mapping. This comparison highlights the system's role as a foundational learning tool, with room for growth toward professional-grade applications through the suggested improvements.

Chapter 7: Conclusion

7.1 Summary of Objectives

The Smart Radar System project, developed as part of the CSC 330 - Logic Design and Switching Circuit Lab under the guidance of Md. Asif Hossain, successfully achieved its primary objectives. The system effectively integrated an Arduino Uno, HC-SR04 ultrasonic sensor, and SG90 servo motor to create a sonar-based object detection mechanism. It enabled real-time scanning from 15° to 165°, calculated distances within a 40 cm range using the speed of sound, and provided a visual representation through a Processing GUI. These accomplishments demonstrate a practical application of logic design and switching circuit principles in a cost-effective, educational context.

7.2 Key Learnings

This project offered valuable insights into the interplay of hardware and software in embedded systems. The process of wiring components on a breadboard, configuring pin assignments, and managing serial communication enhanced our understanding of digital circuit design. Troubleshooting challenges such as noise and wiring stability improved problem-solving skills,

while developing the Processing visualization deepened our knowledge of data interpretation and graphical programming. The experience underscored the importance of calibration and iterative testing in achieving reliable performance.

7.3 Future Scope

The Smart Radar System lays a foundation for further development. Future iterations could extend the detection range to the HC-SR04's full 400 cm capability, optimize the sweep speed with a faster servo or reduced delay, and incorporate 3D scanning with an additional servo. Integrating features like proximity alerts or mobile app connectivity could broaden its practical applications in robotics or security. These enhancements would align the system with more advanced projects, offering opportunities to explore complex circuit designs and real-world implementations.

References

1. Arduino. (n.d.). *Arduino Uno Rev3*. Retrieved from https://store.arduino.cc/products/arduino-uno-rev3 (Official documentation and technical specifications for the Arduino Uno microcontroller used in the project.)
2. Arduino. (n.d.). *HC-SR04 Ultrasonic Sensor*. Retrieved from https://www.arduino.cc/en/Tutorial/UltrasonicSensor (Resource providing details on the HC-SR04 sensor's operation and integration with Arduino.)
3. Processing Foundation. (n.d.). *Processing*. Retrieved from https://processing.org/ (Official site for the Processing programming environment used for visualization in the project.)
4. ServoCity. (n.d.). SG90 Micro Servo Motor. Retrieved from https://www.servocity.com/sg90-micro-servo (Technical information on the SG90 servo motor's specifications and control methods.)
5. Build Your Own DIY Radar System Using Arduino: A Step-by-Step Guide! By SciCraft. Youtube. https://youtu.be/SvLObGL-5ZY?si=bGfAhaNbuGm9a0br

Appendices

Appendix A: Arduino Code

Below is the complete Arduino code used to control the servo motor and ultrasonic sensor, calculate distances, and send data via the serial port.

```
// Includes the Servo library
#include <Servo.h>

// Defines Trig 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

    Serial.print(i); // Sends the current degree into the Serial Port
    Serial.print(","); // Sends addition character for indexing
    Serial.print(distance); // Sends the distance value into the Serial Port
    Serial.print("."); // Sends addition character 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
  distance = duration * 0.034 / 2;
  return distance;
}
```

Appendix B: Processing Code

Below is the complete Processing code used to visualize the radar sweep and object detection data received from the Arduino via serial communication.

```
import processing.serial.*; // Imports library for serial communication
import java.awt.event.KeyEvent; // Imports library for reading 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;


void setup() {

 size(1200, 700); // Sets canvas size

 smooth();

 myPort = new Serial(this, "COM5", 9600); // Starts serial communication

 myPort.bufferUntil('.'); // Reads data up to the character '.'

}


void draw() {

 fill(98, 245, 31);

 // Simulating motion blur and slow fade

 noStroke();

 fill(0, 4);

 rect(0, 0, width, height - height * 0.065);
```

```
  fill(98, 245, 31); // Green color
  // Calls functions for drawing the radar
  drawRadar();
  drawLine();
  drawObject();
  drawText();
}

void serialEvent(Serial myPort) {
  // Reads data from the Serial Port up to the character '.'
  data = myPort.readStringUntil('.');
  data = data.substring(0, data.length() - 1);

  index1 = data.indexOf(","); // Finds the character ','
  angle = data.substring(0, index1); // Reads angle value
  distance = data.substring(index1 + 1, data.length()); // Reads distance value

  // Converts String variables into Integer
  iAngle = int(angle);
  iDistance = int(distance);
}
```

```
void drawRadar() {
  pushMatrix();
  translate(width / 2, height - height * 0.074); // Moves starting coordinates
  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 starting coordinates
  strokeWeight(9);
  stroke(255, 10, 10); // Red color
  pixsDistance = iDistance * ((height - height * 0.1666) * 0.025); // Converts distance to pixels
  // Limits range to 40 cm
  if (iDistance < 40) {
    // Draws the object
    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 starting coordinates
   line(0, 0, (height - height * 0.12) * cos(radians(iAngle)), -(height - height * 0.12) * sin(radians(iAngle)));
// Draws the line
```

```
 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("SciCraft ", width - width * 0.875, height - height * 0.0277);
```

```
  text("Angle: " + iAngle + " °", width - width * 0.48, height - height * 0.0277);

  text("Distance: ", width - width * 0.26, 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();

}
```