Arduino Radar with Led diode and encoding data onto an image

1. Detailed Project Description

This project presents the practical development of a short-range radar detection system based on the Arduino Mega 2560 platform, enhanced with visual indication and data encoding techniques. The system integrates an ultrasonic sensor (HC-SR04), a servo motor for angular scanning, a light-emitting diode (LED) for visual signaling, and serial communication with a personal computer for real-time data visualization and processing.

The primary objective of the project is to demonstrate the implementation of an embedded radar system capable of detecting objects within a limited distance range and representing their position in terms of angular displacement and measured distance. The ultrasonic sensor is mounted on a servo motor, which enables controlled rotation between predefined angular limits, typically from 0° to 180°. During operation, the sensor emits ultrasonic pulses and measures the time required for the reflected signal to return, allowing calculation of the distance to a detected object.

The Arduino Mega 2560 is programmed using the Arduino IDE to control the servo motor, acquire distance measurements, activate the LED indicator when an object is detected within a specified threshold, and transmit structured data via serial communication. The transmitted data consists of angle and distance values formatted for interpretation by an external visualization program.

On the personal computer, a program developed in the Processing environment establishes serial communication with the Arduino board, reads the transmitted data, and generates a dynamic radar-style graphical interface. The radar visualization displays angular reference lines, distance arcs, and highlighted object positions, simulating the operational behavior of a conventional radar system.

In addition to radar implementation, the project incorporates a steganographic component using the QuickStego tool. The detected angle and distance values obtained from the radar system are encoded into a digital image as hidden text data. This process demonstrates how sensor-derived information can be embedded within a multimedia file without altering its visible appearance. The encoded image can later be decoded to retrieve the hidden radar data, illustrating the basic principles of information hiding and data security.

Overall, the project integrates embedded system design, sensor-based measurement, real-time data visualization, and digital steganography into a unified educational framework. It

demonstrates interdisciplinary application across digital electronics, programming, signal acquisition, data communication, and information security.

2. How the System Works

The radar system operates through coordinated interaction between the Arduino Mega 2560, the ultrasonic sensor (HC-SR04), the servo motor, the LED indicator, and the Processing-based visualization program.

The ultrasonic sensor is mounted on a servo motor, which enables controlled angular scanning between 0° and 180°. The servo motor is driven by the Arduino through pulse-width modulation (PWM), allowing precise positioning at defined angular increments. As the servo rotates, the ultrasonic sensor emits high-frequency sound pulses and measures the time interval required for the reflected signal to return after striking a nearby object. Based on this time-of-flight principle, the Arduino calculates the distance to the detected object.

For each angular position, the measured distance value is associated with the corresponding angle of the servo motor. These two parameters, angle and distance, form the fundamental data pair representing the spatial location of an object within the scanning range.

When an object is detected within a predefined distance threshold, the LED diode is activated to provide immediate visual indication of detection. This LED serves as a local alert mechanism, enhancing the physical feedback of the system.

Simultaneously, the Arduino transmits the angle and distance data through serial communication to a personal computer via a USB connection. The transmitted data are formatted in a structured manner, typically as comma-separated values, to allow efficient parsing by the Processing program.

On the computer side, the Processing application establishes a serial connection with the Arduino and continuously reads incoming data. The received angle and distance values are processed and converted into graphical coordinates. The radar interface dynamically draws angular reference lines, concentric distance arcs, and highlights detected objects at their respective polar coordinates, simulating the operation of a conventional radar display.

In the extended phase of the project, the measured data can be encoded into a digital image using the QuickStego tool. The radar output values are embedded as hidden text within a selected cover image, demonstrating the practical integration of sensor-based data acquisition and basic steganographic techniques.

Through this coordinated interaction of hardware control, sensor measurement, serial communication, graphical visualization, and data encoding, the system demonstrates a complete embedded radar detection workflow.

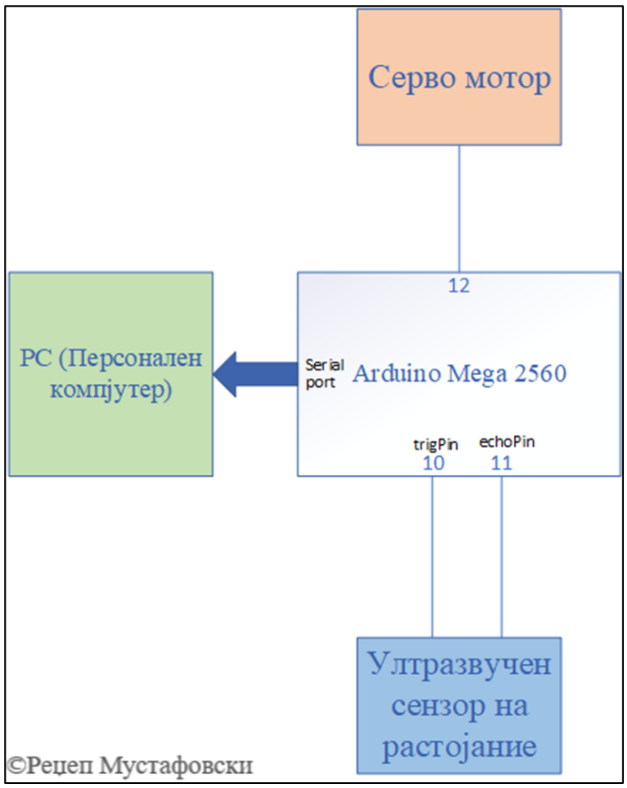

Figure 1. Block diagram of connection

3. Circuit Description

The hardware implementation of the radar system is based on the Arduino Mega 2560 microcontroller, which serves as the central control unit for sensor acquisition, actuator control, and signal indication.

The ultrasonic sensor HC-SR04 is connected to the Arduino to enable distance measurement. The sensor consists of four main pins: VCC, Trig, Echo, and GND. The VCC pin is connected to the 5V power supply of the Arduino, while the GND pin is connected to the common ground. The Trig pin is connected to a designated digital output pin of the Arduino, which generates the trigger pulse required to initiate ultrasonic transmission. The Echo pin is connected to a digital input pin, which receives the reflected signal and allows the Arduino to calculate the time-of-flight of the ultrasonic wave.

The ultrasonic sensor is mounted on an SG90 micro-servo motor to enable angular scanning. The servo motor has three connections: power (5V), ground (GND), and control

signal. The control signal wire is connected to a PWM-capable digital pin on the Arduino Mega 2560, allowing precise control of the servo shaft position between 0° and 180°. The servo rotation enables the ultrasonic sensor to scan the surrounding area in a horizontal plane.

An LED diode is incorporated into the circuit as a visual detection indicator. The anode of the LED is connected to a digital output pin of the Arduino through an appropriate current-limiting resistor, while the cathode is connected to ground. When an object is detected within a predefined distance threshold, the Arduino sets the corresponding digital pin to HIGH, activating the LED.

All components are interconnected using a breadboard, which facilitates organized wiring and stable electrical connections. The power distribution rails of the breadboard are connected to the 5V and GND pins of the Arduino, ensuring consistent voltage supply to the ultrasonic sensor and servo motor.

The Arduino Mega 2560 is connected to a personal computer via a USB cable. This connection provides both power supply and serial communication for transmitting angle and distance data to the Processing-based visualization software.

Overall, the circuit represents an integrated embedded system in which sensing, actuation, signaling, and communication are coordinated through the microcontroller platform.

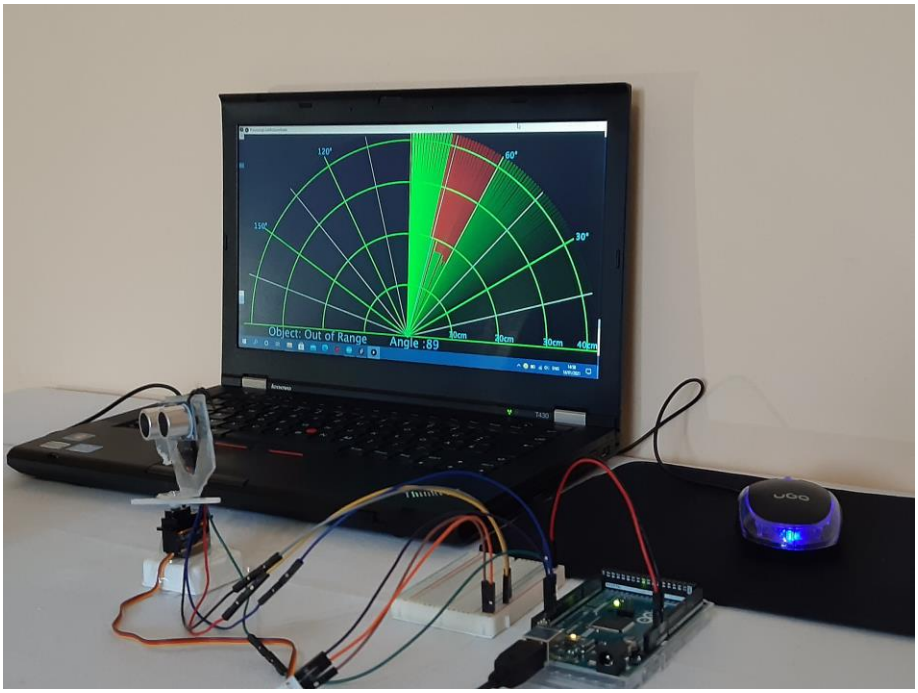

Figure 2. Experimental Setup and Real-Time Radar Visualization of the Arduino-Based Detection System

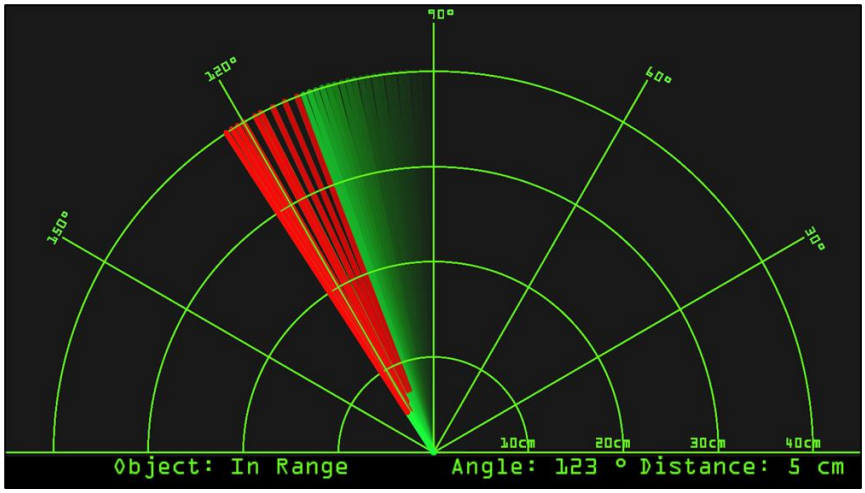

Figure 3. Real-Time Radar Interface Display Showing Object Detection at 123° and 5 cm

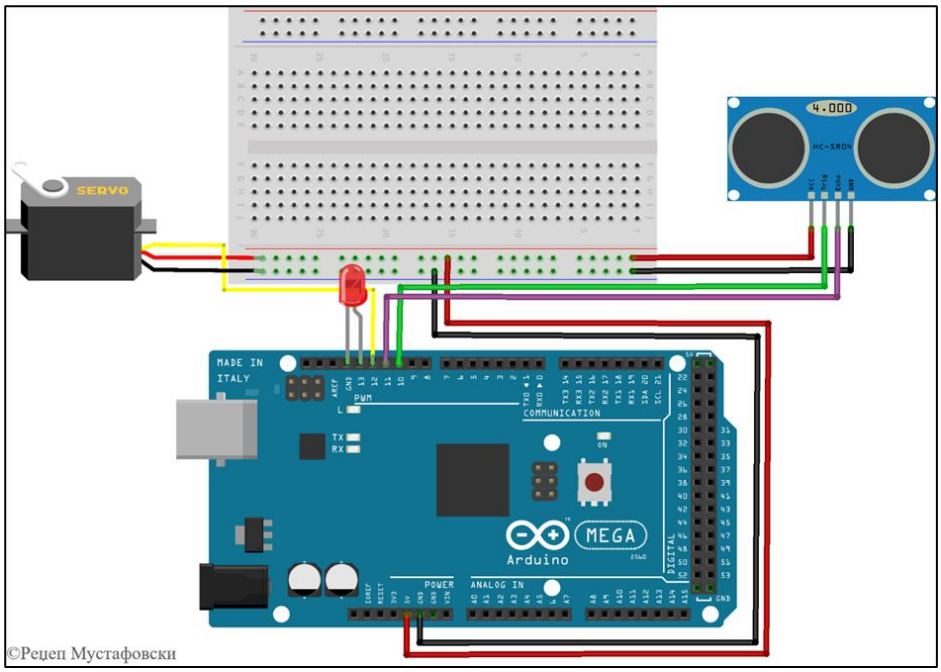

Figure 4. Circuit Diagram of the Arduino Mega 2560 Radar System with Ultrasonic Sensor, Servo Motor, and LED Indicator

4. Code

The following Processing code is used to receive structured serial data from the Arduino Mega 2560, extract the transmitted angle and distance values, and generate a real-time radar visualization interface representing servo-based scanning and object detection.

This is an updated and enhanced version of the original Arduino Radar visualization concept developed by Dejan Nedelkovski. The current implementation has been updated, optimized, and extended by Rexhep Mustafovski, including full-screen adaptive visualization, improved distance-to-pixel scaling, refined radar arcs and end-point geometry, and more robust parsing of serial input data.

```
/*  Arduino Radar Project
 *
 *  Updated version. Fits any screen  resolution!
 *  Just change the values in the size() function,
 *  with your screen resolution.
 *
 *  by Dejan Nedelkovski, updated by Rexhep  Mustafovski
 *
 */
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
// defubes variables
String angle="";
String distance="";
String  data="";
String noObject;
float pixsDistance;
float Razmer;  // dodadeno
int  iAngle, iDistance;
int index1=0;
int index2=0;
```

```
int pravec=1;

PFont orcFont;

void   setup() {


 //size (1200, 950); // ***CHANGE THIS TO YOUR SCREEN RESOLUTION***


 fullScreen();   //   ********************  Size e zameneto s fullScreen    taka   da avtomatski ke se zadadat height i with


 Razmer = (height-height*0.21)*0.025;   // cm pretvarame vo pikseli so formulata:  x*Razmer = x * ((height-height*0.25)*0.025);        Dolzina[vo cm] * (RabotnaVisinaEkranVoPixeli * 1/40)  max=40cm


   smooth();

 myPort = new Serial(this,"COM3", 115200); // starts the serial communication

   myPort.bufferUntil('.'); // reads the data from the serial port up to the character   '.'. So actually it reads this: angle,distance.

 orcFont = loadFont("OCRAExtended-30.vlw");


 //************************************************************ dopolna

   //iAngle = 0;

 //iDistance = 25;

 //frameRate(100);


 //************************************************************

}

void draw() {
```

```
 fill(98,245,31);
 textFont(orcFont);
  // simulating motion blur and slow fade of the moving line
 noStroke();
  fill(0,5);
 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();


  //*********************************************************** dopolna za
Avtomatski test na iscrtuvanjeto


 //iAngle = iAngle+pravec;
   // if (iAngle==0) {
 //  pravec=pravec*(-1);
 //   }
 //  else if (iAngle==180)
   //  {
 //  pravec=pravec*(-1);
 //  }


  //***********************************************************
```

```
}

void   serialEvent (Serial myPort) { // starts reading data from the Serial Port

  //   reads the data from the Serial Port up to the character '.' and puts it into the   String variable "data".

  data = myPort.readStringUntil('.');

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


  index1 = data.indexOf(","); // find the character ',' and puts it into   the variable "index1"


  // angle= (data.substring(0, index1)); // read   the data from position "0" to position of the variable index1 or thats the value   of the angle the Arduino Board sent into the Serial Port

  //distance= data.substring(index1+1,   data.length()); // read the data from position "index1" to the end of the data   pr thats the value of the distance


  angle= trim(data.substring(0, index1));                    // dodadena funkcija trim za da gi otvrli praznite mesta

   distance= trim(data.substring(index1+1, data.length()));    // dodadena funkcija   trim za da gi otvrli praznite mesta

  // converts the String variables   into Integer

  iAngle = int(angle);

  iDistance = int(distance);

}

void   drawRadar() {

  pushMatrix();

  translate(width/2,height-height*0.074); //   moves the starting coordinats to new location

  noFill();

  strokeWeight(2);

   stroke(98,245,31);
```

```
  // draws the arc lines
  arc(0,0,2*40*Razmer,2*40*Razmer,PI,TWO_PI);      //      Moi novi formuli  so krugovi na 10, 20, 30 i 40 cm
  arc(0,0,2*30*Razmer,2*30*Razmer,PI,TWO_PI);
   arc(0,0,2*20*Razmer,2*20*Razmer,PI,TWO_PI);
  arc(0,0,2*10*Razmer,2*10*Razmer,PI,TWO_PI);

  //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);             //  Originalni formuli
  //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 coordinats to new location
  strokeWeight(9);
  stroke(255,10,10);   // red color
```

```
  pixsDistance = iDistance*Razmer; // converts the distance   from the sensor from cm to pixels

  // limiting the range to 40 cms

  if(iDistance<40){

    // draws the object according to the angle and the distance


  line(pixsDistance*cos(radians(iAngle)),-
pixsDistance*sin(radians(iAngle)),40*Razmer*cos(radians(iAngle)),-
40*Razmer*sin(radians(iAngle)));   // Popravena linija so novi krajni koordinati

//  line(pixsDistance*cos(radians(iAngle)),-
pixsDistance*sin(radians(iAngle)),(iDistance+2)*Razmer*cos(radians(iAngle)),-
(iDistance+2)*Razmer*sin(radians(iAngle)));   // Objekt so golemina od 1 cm

  }

  popMatrix();

}

void drawLine()   {

  pushMatrix();

  strokeWeight(9);

  stroke(30,250,60);

  translate(width/2,height-height*0.074);   // moves the starting coordinats 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

  line(0,0,40*Razmer*cos(radians(iAngle)),-40*Razmer*sin(radians(iAngle)));   // Nova
popravena linija za krajnite koordinati

  popMatrix();

}

void drawText()   { // draws the texts on the screen


  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*0.41,height-height*0.0833);
 //text("20cm",width-width*0.3,height-height*0.0833);
//text("30cm",width-width*0.18,height-height*0.0833);
 //text("40cm",width-width*0.06,height-height*0.0833);

text("10cm",width/2+7*Razmer,height-height*0.0833);
 text("20cm",width/2+17*Razmer,height-height*0.0833);
text("30cm",width/2+27*Razmer,height-height*0.0833);
 text("40cm",width/2+37*Razmer,height-height*0.0833);

textSize(40);
 text("Object: " + noObject, 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();
}
```

5. Software Implementation

The software implementation of the system is realized through two coordinated software components: the Arduino-side control program, which performs sensing and scanning operations, and the Processing-based visualization program, which receives the transmitted data and generates a real-time radar interface on the personal computer.

On the Arduino Mega 2560 side, the control logic manages the angular scanning procedure by positioning the servo motor within a defined range, typically from 0° to 180°. For each angular step, the ultrasonic sensor HC-SR04 is triggered to generate an ultrasonic pulse, and the echo return time is measured to compute the distance to a potential object using the time-of-flight principle. The computed distance value is then associated with the corresponding servo angle. When the measured distance is within a predefined detection threshold, the system activates the LED indicator to provide local visual feedback of object presence. The Arduino formats the measurement pair (angle and distance) into a structured serial string and transmits it to the PC via USB serial communication.

On the personal computer side, the Processing software establishes a serial connection with the Arduino Mega 2560 and continuously reads incoming data in the format angle,distance. The serial reception is handled through an event-driven approach using the buffer delimiter character, enabling stable real-time acquisition. The received string is parsed to extract angle and distance values, which are converted into integer parameters used for visualization.

The Processing program then renders the radar interface by drawing angular reference lines and distance arcs representing the scanning area. The scanning beam is plotted dynamically based on the current angle, simulating radar rotation. If an object is detected within range, its position is displayed as a highlighted segment in the corresponding direction and distance. Additional interface elements display real-time textual information including object status, measured angle, and distance. The updated implementation supports automatic scaling to different screen resolutions through full-screen rendering and adaptive conversion of distance values into pixel coordinates.

Overall, the software implementation provides a complete workflow from sensor measurement and embedded processing to real-time communication and graphical

representation, demonstrating key principles of embedded system programming, data acquisition, and interactive visualization.

6. Educational Value

The Arduino Radar with LED and Data Encoding project provides significant educational value by integrating multiple engineering concepts into a single practical laboratory activity. The system combines embedded system programming, sensor-based measurement, actuator control, serial communication, real-time visualization, and introductory information security techniques within a structured learning framework.

From an embedded systems perspective, students gain hands-on experience in configuring and programming the Arduino Mega 2560 to control a servo motor, acquire ultrasonic distance measurements, and manage digital output through an LED indicator. This strengthens understanding of microcontroller architecture, input–output interfacing, timing control, and hardware–software integration.

The project also enhances knowledge of signal acquisition and spatial representation. Students learn how distance measurements are calculated using the time-of-flight principle and how angular positioning enables two-dimensional spatial mapping. By transmitting structured angle–distance data to a personal computer, they understand the fundamentals of serial communication protocols and real-time data exchange between hardware and software environments.

The Processing-based visualization component introduces graphical programming concepts, coordinate transformation, polar-to-Cartesian conversion, and dynamic rendering techniques. Students observe how raw sensor data can be converted into an intuitive radar-style interface, reinforcing concepts of data interpretation and human–machine interaction.

The integration of steganography through the QuickStego tool extends the educational scope toward information security. By encoding radar-generated data into digital images, students are introduced to basic principles of data hiding and secure information transmission. This interdisciplinary approach connects embedded systems with cybersecurity fundamentals.

Overall, the project supports the development of analytical thinking, system integration skills, and practical engineering competence. It is particularly suitable for undergraduate students in electrical engineering, electronic engineering, computer engineering, and related technical disciplines, especially at the second- or third-year level.

7. Conclusion

The developed Arduino-based radar system demonstrates the practical implementation of a short-range object detection platform integrating sensing, actuation, visualization, and data encoding functionalities. Through coordinated interaction between the Arduino Mega 2560, the ultrasonic sensor, the servo motor, and the Processing visualization environment, the system successfully performs angular scanning and real-time distance measurement. The detected object positions are accurately represented on a radar-style graphical interface, providing intuitive spatial feedback.

The incorporation of an LED indicator enhances the system by offering immediate physical confirmation of object detection within a defined threshold range. Furthermore, the additional implementation of steganography using the QuickStego tool extends the project beyond conventional embedded system applications, demonstrating how sensor-derived data can be securely encoded into digital images. This interdisciplinary integration illustrates the practical connection between embedded systems and information security concepts.

From an educational standpoint, the project provides a comprehensive laboratory framework that combines microcontroller programming, hardware interfacing, serial communication, graphical data visualization, and introductory cybersecurity techniques. It reinforces both theoretical understanding and practical engineering skills, making it suitable for undergraduate-level instruction in electrical, electronic, and computer engineering disciplines.

Overall, the project confirms the effectiveness of model-based and code-based embedded system development for real-time sensing applications and highlights the value of integrating multiple technological domains within a single coherent engineering solution.