Names	NetIDs	Emails
Om Goyal	ogoyal3	ogoyal3@uic.edu
Sai Rohan Kakarlapudi	skaka4	skaka4@uic.edu
Abdullah Irfan	airfan6	airfan6@uic.edu
Sufiyan Ahmed Syed	ssyed65	ssyed65@uic.edu

GROUP 19

Intelligent Parking Guidance System

Abstract :

This project develops an Intelligent Parking Guidance System designed to streamline the parking process in small lots using Arduino microcontrollers and ultrasonic sensors. By detecting vehicle presence and guiding drivers with real-time availability via LEDs and an LCD display, the system aims to reduce search time and enhance parking efficiency. Each parking spot features an ultrasonic sensor for occupancy detection, with LEDs indicating space status. A central Arduino aggregates data, guiding drivers to vacant spots and assisting with precise parking alignment, thus improving safety and optimizing parking space utilization. This innovative approach leverages basic technology to address parking challenges effectively.

1. Project Overview

The Intelligent Parking Guidance System is engineered to alleviate the challenges associated with parking in small and congested lots. Utilizing Arduino microcontrollers and ultrasonic sensors, this system detects the presence of vehicles and guides drivers to available spots using real-time feedback mechanisms. The main objective is to enhance

parking efficiency by reducing the time drivers spend searching for available spaces, thereby minimizing traffic congestion within the lot.

This system not only simplifies the parking process but also aims to improve overall vehicle safety by reducing potential collisions during parking maneuvers. By providing clear and instant visual guidance through LEDs and LCD displays, the system ensures that drivers can easily navigate the parking lot and park their vehicles safely and efficiently. The innovative use of simple technology to solve complex problems sets this project apart in the field of automated parking solutions.

2. System Architecture

The architecture of the Intelligent Parking Guidance System is designed with scalability and robustness in mind. At the heart of the system is the Central Control Unit (CCU), which processes data collected from various sensors located at each parking spot. This unit uses Arduino microcontrollers to manage and relay information to other parts of the system, ensuring efficient communication and operation throughout the parking lot.

In addition to the CCU, individual parking spot controllers are equipped with their own Arduino microcontrollers and sensors. These controllers work independently to monitor the occupancy of each spot and communicate this information back to the central unit. This decentralized approach allows the system to maintain functionality even if one part fails, providing a reliable and resilient solution to parking management.

3. Communication Protocol

Communication between the multiple Arduino units is critical for the real-time operation of the parking guidance system. Serial communication is employed to facilitate this interaction, allowing data to be transmitted quickly and reliably between the central unit and individual parking spot controllers. This setup enables the CCU to receive updates about spot availability, reporting a collision/high traffic in the parking lot and other important metrics, which it then uses to direct incoming drivers to open spots.

The system's communication protocol also includes error-checking mechanisms to ensure data integrity during transmission. In the event of communication failure or discrepancies, the system is designed to send alerts and reroute data to maintain continuous operation. This robust communication framework is essential for the dynamic allocation of parking spaces and the overall fluidity of the parking process.

4. Inputs and Outputs

The Intelligent Parking Guidance System utilizes a variety of inputs and outputs to interact with the environment and provide user feedback. Inputs include signals from ultrasonic sensors that detect the presence and position of vehicles within the parking lot. These sensors are crucial for assessing whether spots are occupied or available, and their data is used to update the system in real-time.

Outputs from the system include visual indicators such as LEDs, which light up in different colors to signify the status of each parking spot (green for available, red for occupied). Additionally, LCD screens display numerical data and text messages to provide drivers with guidance on where to find open parking spots. Servo motors are used to control entry and exit barriers, ensuring that access is granted only when appropriate. This integration of various inputs and outputs allows for a seamless and interactive parking experience.

5. Innovation and Originality

The project introduces several innovative features that distinguish it from conventional parking systems. One of the primary innovations is the use of a distributed sensor network that provides accurate and granular data on vehicle location and spot occupancy. This allows the system to guide drivers directly to available spots without the need for manual searching, significantly improving parking efficiency.

In addition to this, our blind spot alert system is completely unique, we have developed this system to be much more efficient and robust to the conditions inside the parking spot, from traditional convex mirrors which often tend to be dusty or deformed or even broken, we have converted this feature into something that is far more practical and well suited for modern parking lots, its loud buzzer and movement to cover a wide angle allows the drivers to better maneuver their vehicles in the parking lot and also have a heads up in case there is an accident or a traffic jam inside the lot.

Another original aspect of the project is its adaptive feedback mechanism. Depending on the parking lot's occupancy and incoming vehicle data, the system dynamically adjusts the information displayed on the LEDs and LCDs. This real-time response not only enhances the user experience but also optimizes the flow of traffic within the parking lot, reducing the potential for congestion and accidents.

6. Construction and Development

The construction and development of the Intelligent Parking Guidance System involves a multi-phase process beginning with the design and simulation of the system using software tools like TinkerCAD. This initial phase allows for the testing of circuit connections and software algorithms in a virtual environment, ensuring that all components work harmoniously before physical assembly begins.

Following the simulation, the physical assembly of the system involves setting up the Arduino boards, connecting ultrasonic sensors, LEDs, LCD displays, and configuring the necessary wiring. This stage is critical as it requires precise calibration of sensors and meticulous programming of the Arduinos to handle real-world scenarios effectively. Systematic testing during assembly helps identify and rectify potential issues early in the development process.

System Development Phases

Building the Arduino Subsystems:

- Refer to the Fritzing diagrams provided for having an understanding of each subsystem and each of the I/O devices used within them
- Begin by assembling four distinct Arduino subsystems, each responsible for different aspects of the parking guidance system. Each subsystem will have specific sensors and output components like LEDs or LCDs, tailored to its role, for instance, vehicle detection or spot status indication.
- Use a structured approach to connect ultrasonic sensors, LEDs, LCD displays, and other necessary components to the Arduino boards. The precise connection details and configurations should align with the system's design requirements, ensuring all components interact correctly without interference .

Testing Each Subsystem Individually:

- Conduct individual tests for each subsystem to ensure they function correctly in isolation. This includes testing sensor responsiveness, LED functionality, and LCD output accuracy. Utilize test scripts written in the Arduino IDE to simulate various parking scenarios, checking for correct sensor data interpretation and response handling .

Integrating Serial Communication:

- Implement serial communication between the central control unit (CCU) and each subsystem. This involves setting up communication protocols to allow seamless data transmission across the subsystems and the CCU. This setup is crucial for the

real-time data exchange necessary for the system's functionality, such as spot availability updates and vehicle detection alerts .

Comprehensive System Testing:

- After integrating all subsystems with the CCU, conduct comprehensive testing to verify the overall system integrity and inter-component communication. Test the data flow from the sensors to the CCU and then to the output devices, ensuring the system operates as intended under various scenarios, such as high vehicle inflow rates and simultaneous multiple spot changes. This phase might involve real-life stress testing to assess the system's performance under typical parking lot conditions .

Additional notes:

- Ensure that you are able to send and receive data by using appropriate coding to ensure that the system is functional
- Ensure proper wire connections, loose connections cause power loss and faulty LCD displays

7. User Interaction and System Operation

The Intelligent Parking Guidance System is designed to be user-friendly and intuitive. As a vehicle approaches the entrance, the system detects its presence using an ultrasonic sensor. If parking spots are available, the system activates a servo motor to lift the entry gate and allows the vehicle to enter. The driver is then guided by clear, real-time visual cues on an LCD display and LED indicators that direct them to the nearest available parking spot.

Once parked, the system continues to monitor the vehicle's position to ensure it is well-placed within the spot. If the vehicle is too close to line markings or other cars, the system alerts the driver via a buzzer or text message on the LCD, suggesting adjustments. This proactive feature not only improves safety but also enhances the overall efficiency of the parking lot by ensuring that each vehicle is correctly positioned within its designated space.

Timeline of Development

Week 1 (Feb 26): Project Planning and System Architecture Design

- Conceptualized the Intelligent Parking Guidance System, laying out a comprehensive blueprint integrating sensor-driven vehicle detection, automated gate control, and dynamic user feedback mechanisms.
- Selected Arduino Uno and Arduino Mega boards for their robust I/O capabilities and community support, designating them for the Central Control Unit (CCU) and individual parking spot controllers, respectively.
- Identified essential components: HC-SR04 ultrasonic sensors for precise vehicle proximity detection, SG90 servo motors for automated gate mechanisms, RGB LEDs for real-time status indication, a 16x2 character LCD for displaying parking guidance, and a TM1637 4-digit 7-segment display for real-time parking occupancy data.
- Segregated the project into a dual-tier architecture comprising a Central Control Unit for data aggregation and management, alongside discrete parking spot modules for localized control.
- Assigned project roles, aligning team expertise with system development phases: hardware integration, software programming, and system testing & optimization.

Week 2 (Mar 4): Hardware Selection and Prototyping

- Procured high-fidelity components, securing multiple Arduino boards, ultrasonic sensors for distance measurement, LEDs and LCDs for user interfaces, a servo motor for gate simulation, and foundational circuitry elements including breadboards, jumper wires, and resistors.
- Engineered a prototype circuit for the CCU, employing UART (Universal Asynchronous Receiver-Transmitter) protocol for serial communication to synchronize data across Arduino nodes.
- Initiated physical assembly, positioning ultrasonic sensors for optimal vehicular detection while strategically placing LEDs and LCDs to ensure user visibility and engagement.

Week 3 (Mar 10-15): Software Development and Preliminary Integration

- Crafted Arduino sketches for the CCU and individual parking modules, embedding algorithms for vehicle detection, servo motor actuation for gate control, and dynamic feedback via RGB LEDs and LCD messages.
- Programmed the CCU to compile occupancy data, visually represented on the TM1637 display to indicate available parking spots, enhancing driver decision-making.
- Introduced scrolling text functionality on the LCD displays, offering dynamic, context-sensitive instructions to users, thereby elevating the parking experience.
- Undertook initial component and system-level testing to validate the integration efficacy, troubleshooting identified discrepancies to refine operational accuracy and responsiveness.

Week 4 (Mar 24 - 29): Advanced System Integration and User Interface Enhancement

- Seamlessly integrated the 16x2 LCD displays with the Arduino nodes to provide instantaneous feedback on parking slot distances and system statuses, utilizing custom characters and animations to enhance readability.
- Implemented asynchronous task management using the `millis()` function, thereby decoupling sensor reading, gate actuation, and feedback mechanisms from each other to prevent operational bottlenecks.
- Conducted exhaustive testing routines, simulating real-world parking scenarios to ensure the integrity of the vehicle detection logic, the precision of the feedback loops, and the reliability of the gate control system.

Week 5 (Apr 1-5): System Communication Enhancement, Optimization, and Deployment

- Enhanced the serial communication framework between the CCU and individual parking nodes, facilitating efficient data exchange and synchronization across the IPGS architecture.
- Developed and validated communication protocols, ensuring error-free transmission of occupancy data and operational commands.
- Systematically analyzed and optimized the software and hardware configurations, focusing on minimizing response times, enhancing data accuracy, and ensuring system stability under peak loads.
- Addressed and rectified all identified issues, refining the system's operational framework to guarantee flawless execution.

- Executed the final assembly of the IPGS, focusing on the strategic placement of components within the parking model to ensure operational efficiency and aesthetic coherence.
- Applied cosmetic enhancements to elevate the system's presentation, reinforcing the professional quality of the project.
- Compiled an exhaustive project dossier, detailing the system architecture, component specifications, and the developmental narrative, supplemented with schematics and source code.
- Prepared an engaging and informative presentation, distilling the project essence, challenges navigated, and the technological innovations introduced.
- Performed a final system walkthrough, demonstrating the IPGS's functionality, showcasing its features, and articulating the project's value proposition.
- Engaged in a critical review session, reflecting on project outcomes, insights gained, and potential avenues for future enhancements.

Week 6 (Apr 8-12): Serial Linking and Real-Life Stress Testing

- Implemented and refined the serial communication links between the Central Control Unit (CCU) and individual parking spot controllers to ensure robust and error-free data transmission across the system.
- Conducted comprehensive real-life stress testing scenarios to evaluate the system's performance under varied conditions, such as high vehicle inflow rates and simultaneous multiple spot changes.
- Troubleshooted and resolved issues related to data consistency and communication delays, optimizing firmware and hardware settings for enhanced reliability.
- Validated the system's capacity to manage and recover from potential failures, ensuring that it remains operational even under stressful conditions.
- Documented the outcomes and insights gained from testing, which provided critical data to further refine the system's operational protocols and user interface elements.

MATERIALS USED

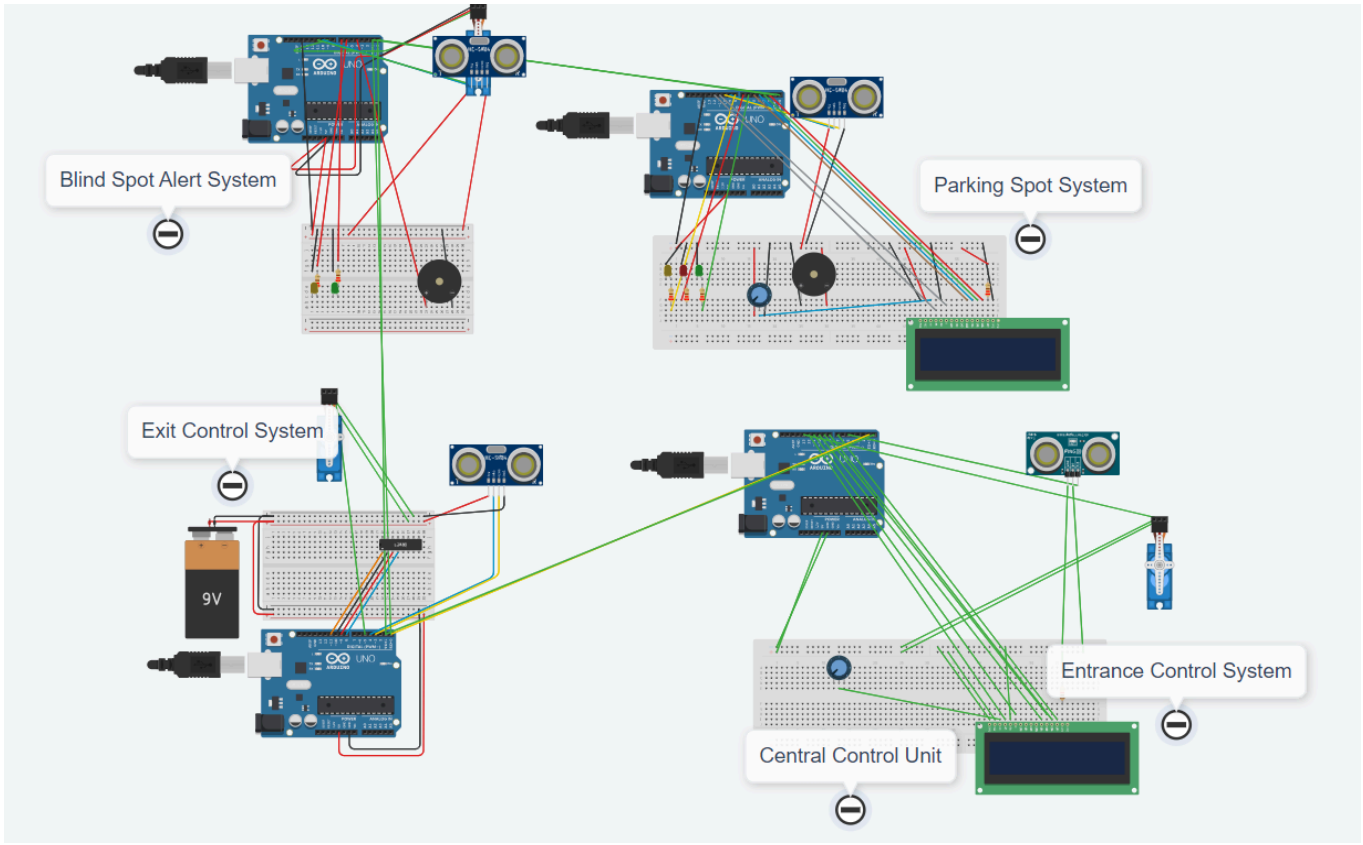
1. Arduino Uno R3
2. Bread Boards
3. LED’s

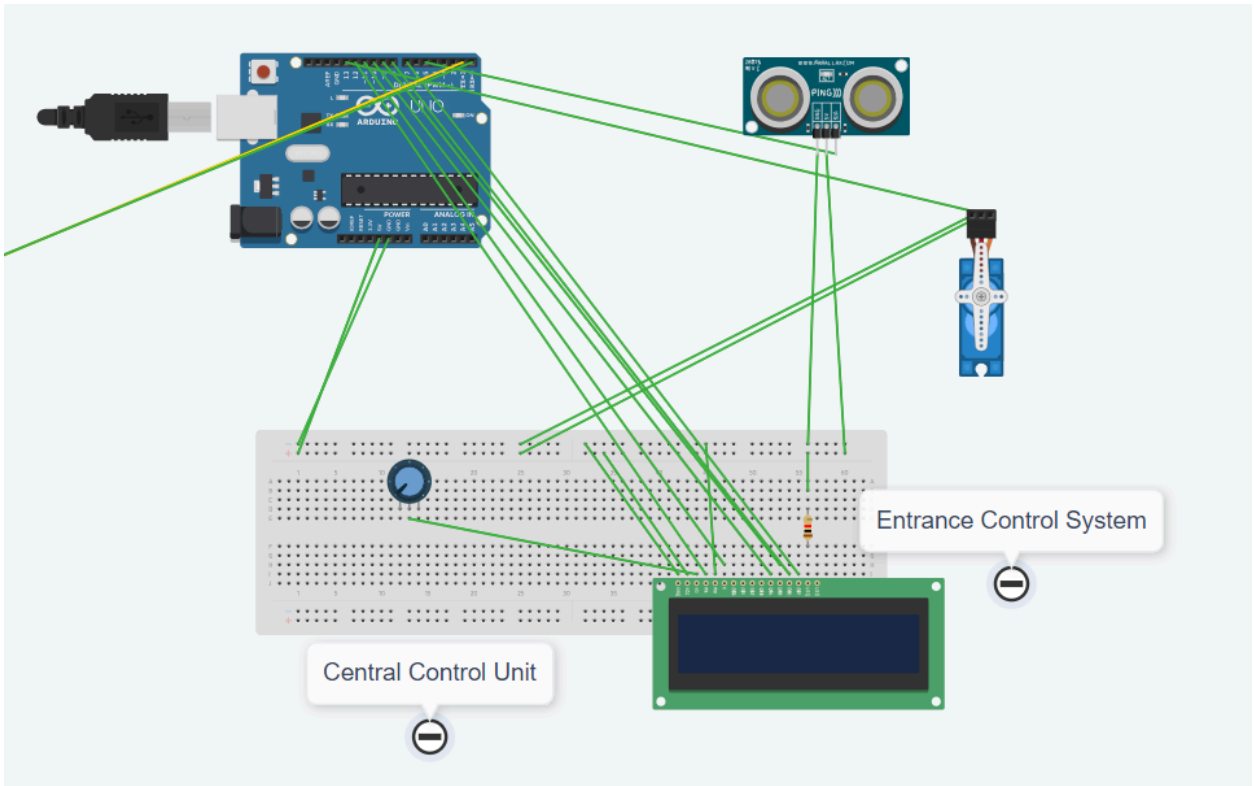
4. LCD 16x2 Display
5. Resistors
6. Buzzers
7. Jumper Cables
8. Servo Motors
9. UltraSonic Sensors

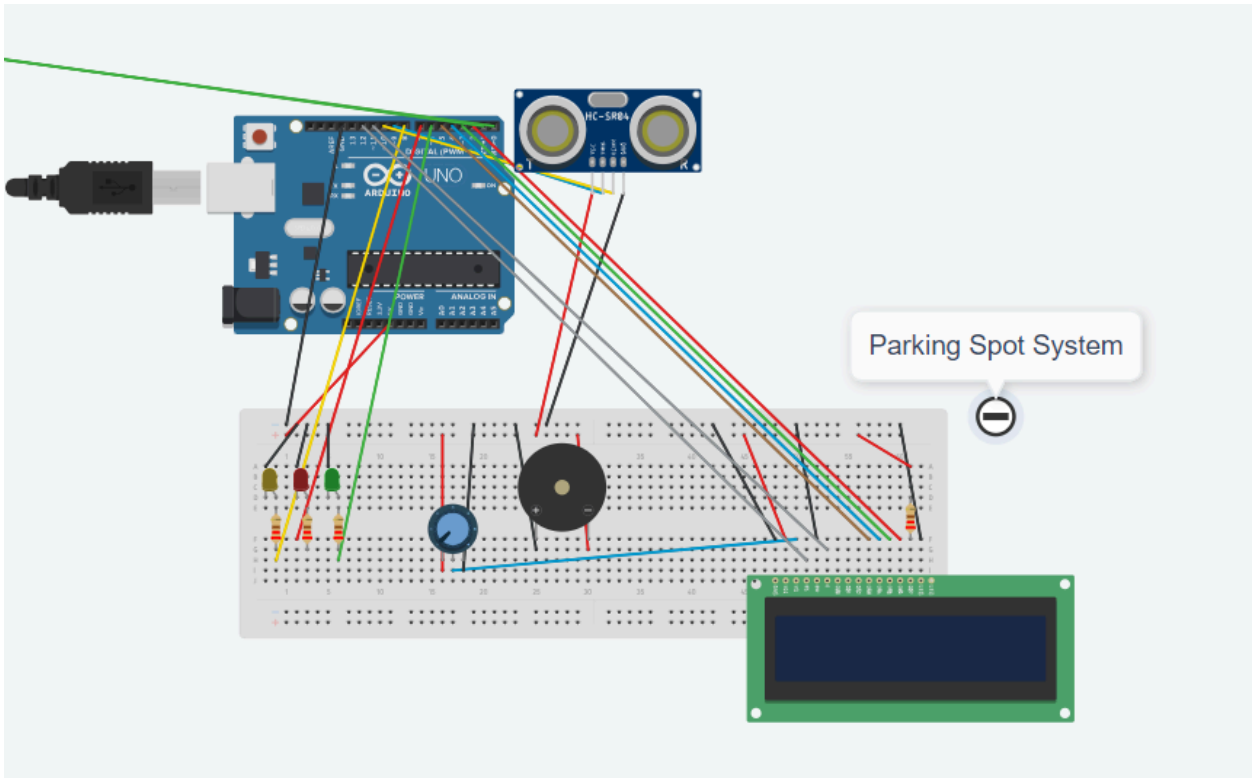
References

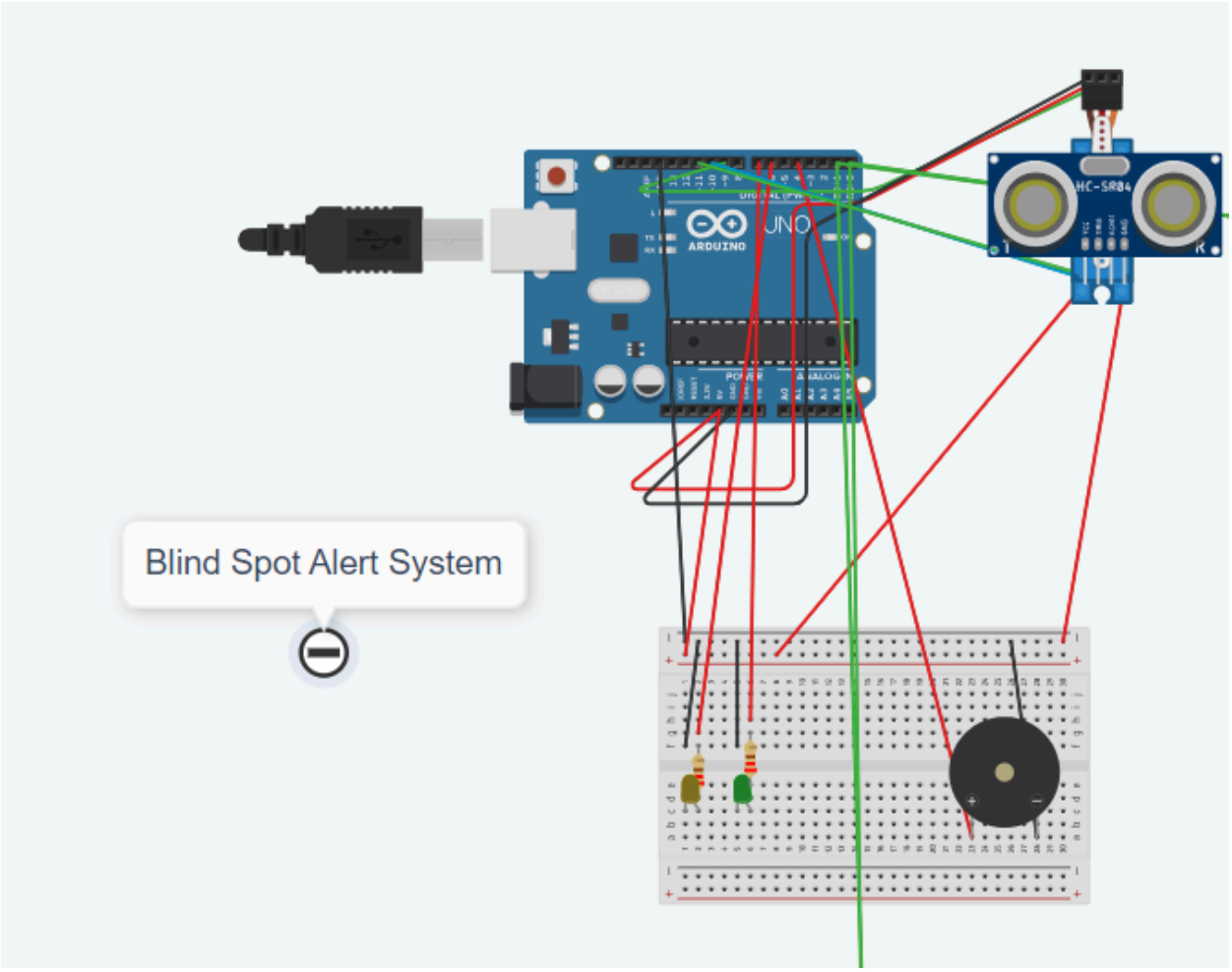
Adding more digital pins for the whole system
https://forum.arduino.cc/t/how-to-add-more-digital-pins-to-arduino-uno-solved/169356/15
Sample for a parking guidance system
https://forum.arduino.cc/t/parking-guidance-system-recommendation/993454
Using ultrasonic sensors for object detection
https://www.youtube.com/watch?v=HqiOpquTRW4
https://www.instructables.com/Controlling-a-Servo-With-an-Ultrasonic-Sensor-Usin/
https://www.youtube.com/watch?v=QbgTl6VSA9Y
Active Buzzer for Arduino
https://www.ardumotive.com/how-to-use-a-buzzer-en.html
Serial communication
https://www.youtube.com/watch?v=_QETdyeMyZw
Reference Connection for Arduino Ultrasonic Sensor
https://arduinointro.com/articles/projects/using-the-hc-sr04-ultrasonic-sensor-to-detect-objects
Reference Connection for Arduino RGB LED
https://projecthub.arduino.cc/semsemharaz/interfacing-rgb-led-with-arduino-b59902
Multiple device Serial Communication
https://arduino.stackexchange.com/questions/48360/serial-communication-between-multiple-devices-or-arduinos

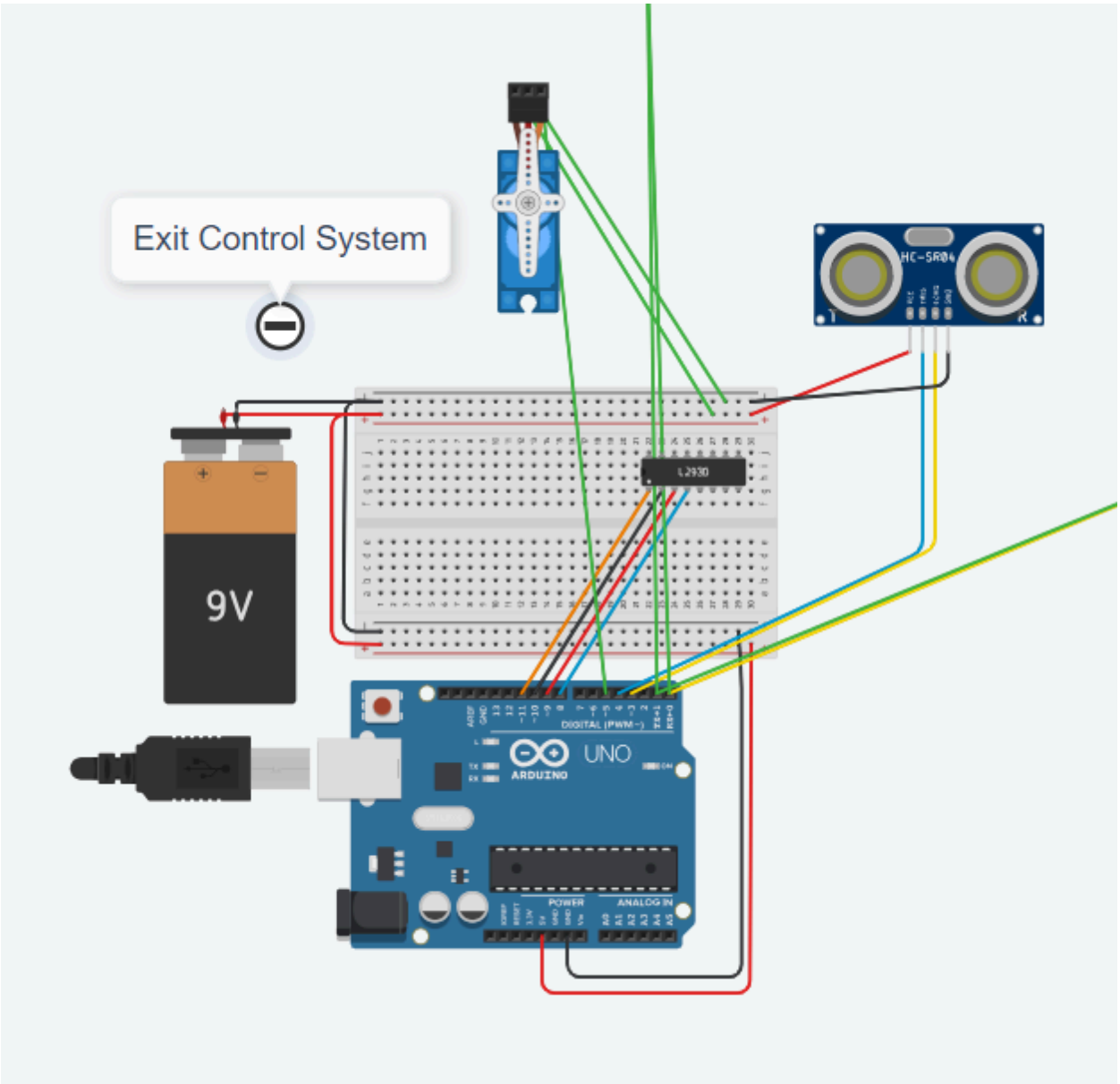
Inclusion of final diagram



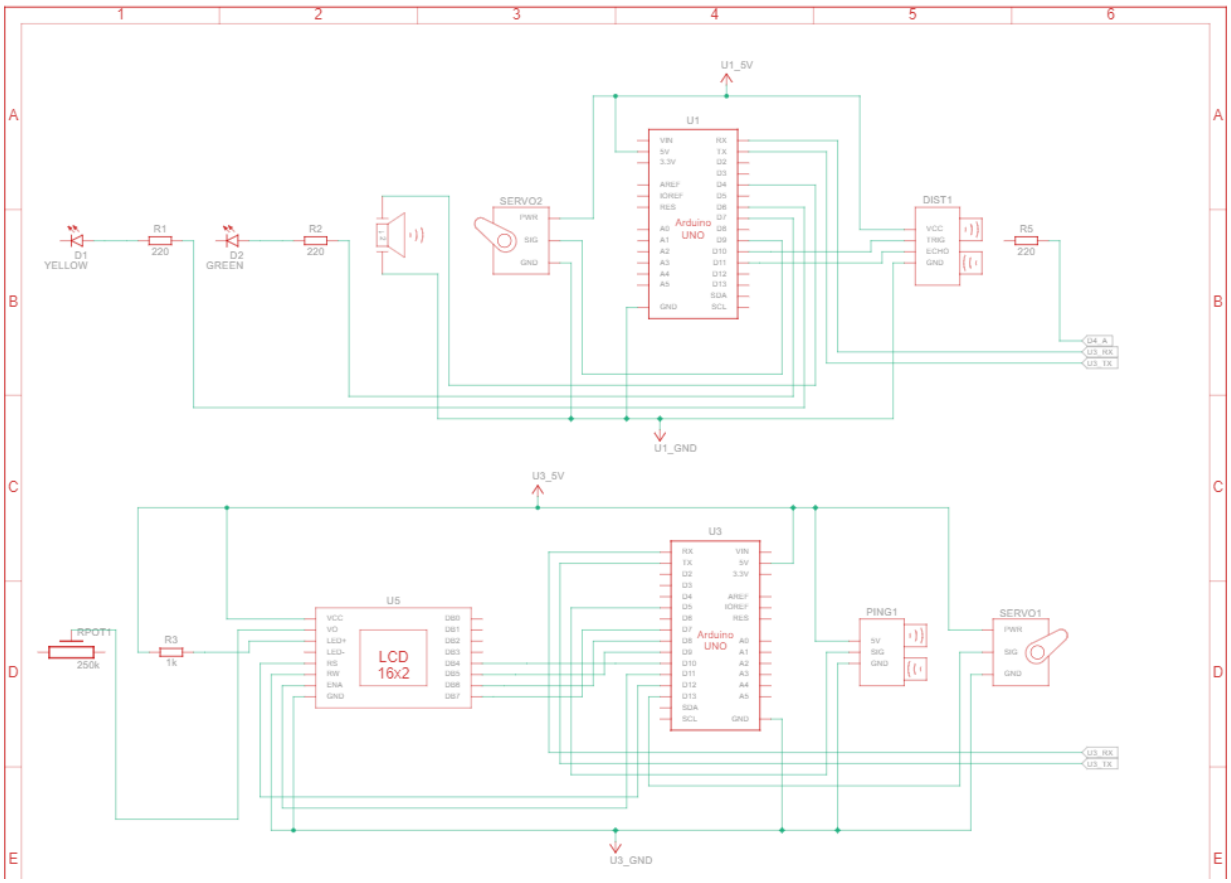



HC-SR04
UNO
ARDUINO
POWER
ANALOG IN
Parking Spot System


Blind Spot Alert System
ARDUINO
UNO
HC-SR04


Exit Control System
HC-SR04
L293D
9V
UNO
ARDUINO
DIGITAL (PWM~)
POWER
ANALOG IN

Schematics :-



Final Code Sketches

Code for Exit System

```
#include <LiquidCrystal.h>
#include <Servo.h>

const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

```
Servo entryBarrier;
const int motorPin = 8;
const int trigPin = 9;
const int echoPin = 10;
volatile bool pulseSent = false;
unsigned long pulseStartTime = 0;
const long pulseDuration = 10;  // Pulse duration in microseconds as a variable for tracking
bool gateOpen = false;
unsigned long gateOpenMillis = 0;
const long gateOpenDuration = 3000; // Duration for which the gate remains open

void setup() {
  Serial.begin(9600);
  lcd.begin(16, 2);
  entryBarrier.attach(motorPin);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  updateLCD("THANK U");
}

void updateLCD(String message) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(message);
}

void sendPulse() {
  digitalWrite(trigPin, HIGH);
  pulseStartTime = micros();
  pulseSent = true;
}

void loop() {
  unsigned long currentMillis = millis();
  unsigned long currentMicros = micros();
```

```
  if (Serial.available()) {
    String received = Serial.readStringUntil('\n');
    received.trim();
    if (received == "CRASH") {
      updateLCD("HIGH TRAFFIC");
      return;
    }
  }

  if (!pulseSent) {
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    sendPulse();
  } else if (pulseSent && (currentMicros - pulseStartTime >= pulseDuration)) {
    digitalWrite(trigPin, LOW);
    pulseSent = false;

    // Measure the distance immediately after sending pulse
    unsigned long duration = pulseIn(echoPin, HIGH);
    int distance = duration * 0.034 / 2;

     if (distance <= 5 && !gateOpen) {
      entryBarrier.write(90);
      gateOpenMillis = currentMillis;
      gateOpen = true;
    }
    if (gateOpen && currentMillis - gateOpenMillis >= gateOpenDuration) {
      entryBarrier.write(0);
      gateOpen = false;
      updateLCD("THANK U");
    }
  }
}
```

Code for entrance

```
#include <LiquidCrystal.h>
#include <Servo.h>
#include<SoftwareSerial.h>

SoftwareSerial DELTASerial(13, 7); // RX pin = 13, TX pin = 7
LiquidCrystal lcd(12, 11, 10, 9, 8, 7);
Servo entryBarrier;
const int motorPin = 13;
const int trigPin = 5;
const int echoPin = 6;
int freeSpots = 3;
unsigned long previousMillis = 0;
const long interval = 1000;
String message = "Intelligent Parking Guidance System    ";
int position = 0;
bool carDetected = false;
unsigned long gateOpenMillis = 0;
const long gateOpenDuration = 3000; // Duration for which the gate remains open

char serialBuffer[32]; // Buffer for serial data
int bufferPosition = 0; // Position in buffer

int measureDiff=10000;

void setup() {
  Serial.begin(9600);
  lcd.begin(16, 2);
  entryBarrier.attach(motorPin);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  updateLCD();
}
```

```
void updateLCD() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(message.substring(0, 16));
  lcd.setCursor(0, 1);
  lcd.print("Free Spots: ");
  lcd.print(freeSpots);
}


void loop() {
  unsigned long currentMillis = millis();

  // Read serial data in a non-blocking manner
  if (Serial.available()) {
        String received = Serial.readStringUntil('\n');
        received.trim();
        Serial.print("Received command: ");
        Serial.println(received);
        if (received == "Spot occupied"){
          if (freeSpots==3){
          freeSpots--;}
          updateLCD();
        }
        if (received == "Spot empty"){
          if (freeSpots==2){
          freeSpots++;}
          updateLCD();
        }

        // if (received == "TESTING 3") {
        //     lcd.setCursor(0, 1);
        //     lcd.print("TEST 3 Received ");
    // }
  }
```

```
  // Scrolling text logic
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    if (!carDetected) {
      scrollText();
    }
  }

  // Car detection logic
  checkForCar(currentMillis);
}


void scrollText() {
  if (carDetected) return;

  lcd.setCursor(0, 0);
  lcd.print(message.substring(position, min(position + 16, message.length())) + "                ");
  position++;
  if (position > message.length() - 16) {
    position = 0;
  }
}

void checkForCar(unsigned long currentMillis) {
  long duration, distance;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;
```

```
  if (distance > 0 && distance < 5 && !carDetected) {
    carDetected = true;
    lcd.setCursor(0, 0);
    lcd.print("Gate Opening    ");
    lcd.setCursor(0, 1);
    lcd.print("Please Wait     ");
    entryBarrier.write(90); // Open the gate
    gateOpenMillis = currentMillis;
    Serial.println("Hi hello");  // This line sends the message "Hi hello"

  }

  if (carDetected && currentMillis - gateOpenMillis > gateOpenDuration) {
    entryBarrier.write(0); // Close the gate
    carDetected = false;
    updateLCD();
    position = 0;
    scrollText();
  }
}
```

Code for Parking Spot

```
#include <LiquidCrystal.h>

// Initialize the LCD library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
#define trigPin 9
#define echoPin 10
#define redLED 7
#define yellowLED 8
#define greenLED 6
#define buzzer 13

unsigned long previousMillis = 0;
unsigned long previousSensorMillis = 0;
unsigned long interval = 500; // Interval at which to blink (milliseconds)
```

```
unsigned long sensorInterval = 50; // Interval at which to read the sensor

int spotsOccupied = 0;
int spotsEmpty = 0;

String currentStatus = "";
String lastStatus = "";

void setup() {
  lcd.begin(16, 2);
  pinMode(redLED, OUTPUT);
  pinMode(yellowLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(9600);
}

void loop() {
    unsigned long currentMillis = millis();

    if (Serial.available()) {
        String received = Serial.readStringUntil('\n');
        received.trim();
        Serial.print("Received command: ");
        Serial.println(received);

        // if (received == "TESTING 3") {
        //     lcd.setCursor(0, 1);
        //     lcd.print("TEST 3 Received ");
        // }
    }

    if (currentMillis - previousSensorMillis >= sensorInterval) {
        previousSensorMillis = currentMillis;
```

```
        measureDistance();
    }

    // Flashing yellow LED logic for "OBJECT MOVING"
    if (currentStatus == "OBJECT MOVING" && (currentMillis - previousMillis >= interval)) {
        previousMillis = currentMillis; // Save the last time we toggled the LED
        digitalWrite(yellowLED, !digitalRead(yellowLED)); // Toggle LED state
    } else if (currentStatus != "OBJECT MOVING") {
        digitalWrite(yellowLED, LOW); // Make sure it's off if not in the correct state
    }
}

void measureDistance() {
    // Trigger the ultrasonic sensor
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);//modify the message being sent to the main use millis to
calculate 10 seconds for parking occupancy and if spot is occupied send message to the
control unit
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    long duration = pulseIn(echoPin, HIGH);
    long distance = (duration / 2) / 30; // Calculate the distance in cm

    updateStatus(distance);
}

void updateStatus(long distance) {
    // Determine the current status based on the distance
    if (distance < 4) { // Less than 4 cm
        currentStatus = "TOO CLOSE";
        spotsOccupied = 1;
        spotsEmpty = 0;
        digitalWrite(buzzer, HIGH);
    } else {
        digitalWrite(buzzer, LOW); // Buzzer off when not too close
```

```
    if (distance >= 4 && distance < 8) {
      currentStatus = "PERFECT POSITION";
      spotsOccupied = 1;
      spotsEmpty = 0;
      Serial.println("Spot occupied");
    } else if (distance >= 8 && distance < 12) {
      currentStatus = "OBJECT MOVING";
      spotsOccupied = 0;
      spotsEmpty = 0;
    } else if (distance >= 12) {
      currentStatus = "EMPTY SPOT";
      spotsOccupied = 0;
      spotsEmpty = 1;
      Serial.println("Spot empty");
    }
  }

  // Update the LCD only if the status has changed
  if (currentStatus != lastStatus) {
    lastStatus = currentStatus;
    lcd.clear();
    lcd.setCursor(0, 1);
    lcd.print(currentStatus);
  }

  // Always display the distance on the first line
  lcd.setCursor(0, 0);
  lcd.print("Dist: ");
  lcd.print(distance);
  lcd.print(" cm   "); // Clear the previous text with extra spaces

  // LED logic
  digitalWrite(redLED, currentStatus == "PERFECT POSITION");
  digitalWrite(greenLED, currentStatus == "EMPTY SPOT");
}
```

Code for BlindSpot Alert System

```
#include <Servo.h>
#define TRIGGER_PIN 9
#define ECHO_PIN 10
#define BUZZER_PIN 7
#define RED_LIGHT_PIN 12
#define YELLOW_LIGHT_PIN 13
#define SERVO_PIN 6

unsigned long previousMillis = 0;
unsigned long servoMillis = 0;
unsigned long yellowLightMillis = 0;
const long interval = 100; // Interval for checking distance
const long yellowInterval = 100; // Interval for yellow light flashing
const long servoInterval = 500; // Interval for servo movement
Servo myservo;
int angle = 0;
bool yellowState = false; // State of yellow light

void setup() {
  myservo.attach(SERVO_PIN);
  pinMode(TRIGGER_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(RED_LIGHT_PIN, OUTPUT);
  pinMode(YELLOW_LIGHT_PIN, OUTPUT);
  digitalWrite(RED_LIGHT_PIN, HIGH);
  myservo.write(0);
  Serial.begin(9600); // Initialize serial communication at 9600 baud rate
}
```

```
void loop() {
  unsigned long currentMillis = millis();
  float duration, distanceCm, distanceInch;

  // Trigger the ultrasonic sensor
  digitalWrite(TRIGGER_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);

  duration = pulseIn(ECHO_PIN, HIGH);
  distanceCm = (duration / 2) / 29.1;
  distanceInch = distanceCm * 0.393701;

  if (distanceInch <= 6) {
    digitalWrite(RED_LIGHT_PIN, LOW);
    digitalWrite(BUZZER_PIN, HIGH);
    Serial.println("CRASH");
    // Send signal to master Arduino

    // Handle yellow light blinking
    if (currentMillis - yellowLightMillis >= yellowInterval) {
      yellowLightMillis = currentMillis;
      yellowState = !yellowState; // Toggle yellow light state
      digitalWrite(YELLOW_LIGHT_PIN, yellowState ? HIGH : LOW);

    }
  } else {
    digitalWrite(RED_LIGHT_PIN, HIGH);
    digitalWrite(YELLOW_LIGHT_PIN, LOW);
    digitalWrite(BUZZER_PIN, LOW);
    yellowState = false; // Reset yellow light state
  }
```

```
  // Servo rotation logic for faster movement
  if (currentMillis - servoMillis >= servoInterval) {
    servoMillis = currentMillis;
    if (angle == 0) {
      myservo.write(90);
      angle = 90;
    } else {
 myservo.write(0);
      angle = 0;
    }
  }
}
```