SpotterSense - Parking Spot Availability Monitor

Project Abstract

SpotterSense is an intelligent parking lot system that monitors and displays real-time parking spot availability. By using multiple Arduinos with ultrasonic sensors and Bluetooth communication, SpotterSense provides accurate and updated parking spot counts for users. One Arduino acts as a central hub that communicates with other Arduinos equipped with sensors to detect vehicle presence. This setup offers a seamless and feasible solution by providing real-time availability data and significantly improving user parking.

Project Ideas

1. **Overall Description**

   Our project, SpotterSense, aims to develop a smart and intuitive parking lot monitoring system that is designed to simplify the parking process for both drivers and parking lot owners. Parking availability within any parking lot has always been very ambiguous and scattered, leaving drivers to spend a significant amount of their time searching for open spots, perhaps even on multiple floors. SpotterSense will address this issue by displaying real-time availability to alert drivers directly to available spots, saving time and reducing frustration.

   SpotterSense's core function will be to detect and display the number of open spots open/free within a given parking lot. A main Arduino board (the "hub") will keep track of availability based on information from individual sensor Arduinos stationed at each parking spot. Customers and parking lot management will benefit from this system's immediate availability updates, improving the overall experience and efficiency of many lot operations.

2. Project Design

The SpotterSense system will consist of one primary Arduino, the central hub, and multiple secondary Arduinos that are each equipped with an ultrasonic sensor and placed at designated parking spot locations. For the purpose of demoing this project, there will be one Arduino acting as a sensor and another that serves as the central hub for data processing and displaying aggregated information to the user. The sensors will be used to measure the distance from the sensor to the car parked in front of it, detecting when a spot is occupied or vacant.

The central hub Arduino will receive data from the secondary Arduino(s), updating the count of available spots in real time. This data will then be formatted to be displayed on an LCD panel (screen) attached to the central hub Arduino, where both ‘drivers’ and ‘parking staff’ can view it. The Arduinos will use Bluetooth communication to send over the bytes of data to the central Arduino, which will allow for fast data transfers and real-time monitoring without the use of wired connections.

3. Plan for Arduinos

Each Arduino equipped with an ultrasonic sensor will frequently check its designated spot’s status. When a car comes within range of the Arduino, the ultrasonic sensor will pick up on this activity by measuring the distance to the vehicle. This distance is then translated into an occupied or vacant status. This status is then transmitted via Bluetooth to the central hub Arduino.

The central hub Arduino will take this data from each parking spot and dynamically update the total number of available spots. This information is formatted into a simple message displayed on a LCD screen. A Bluetooth HC-05 module for

Arduino boards will facilitate the system of transferring data efficiently while also being cost-effective, maintainable, scalable, and using minimal power.

4. **Expected Inputs and Outputs**

Inputs: The ultrasonic sensors will measure the distance between the sensor and the object (car) in front of it. Further than 280 cm (~100 in.), the spot will be considered as vacant. Conversely, if an object is detected within 150 cm (~60 in.), the spot is considered occupied.

Outputs: The individual Arduino(s) will output an occupied or vacant status to the central Arduino board via a Bluetooth connection. The central hub Arduino then outputs the total number of available spots on the LCD screen, updating automatically as cars enter or leave the lot.

5. **Original Work**

The original work we're attempting to implement for SpotterSense is a parking guide to help drivers park their cars. Taking advantage of the distance data from the ultrasonic sensors, we can help drivers gauge the distance their car is away from the curb. By measuring the distance using the sensors, combined with a RGB LED and a beeper, we can signal to the driver approximately how close they are to the curb.

6. **Building the Project**

Sensor Arduino:

Materials

- Arduino UNO
- HC-SR04 ultrasonic sensor
- HC-05 Bluetooth module
- 1x 220 Ohms resistor
- RGB LED

- Buzzer
- Wires
- Breadboard

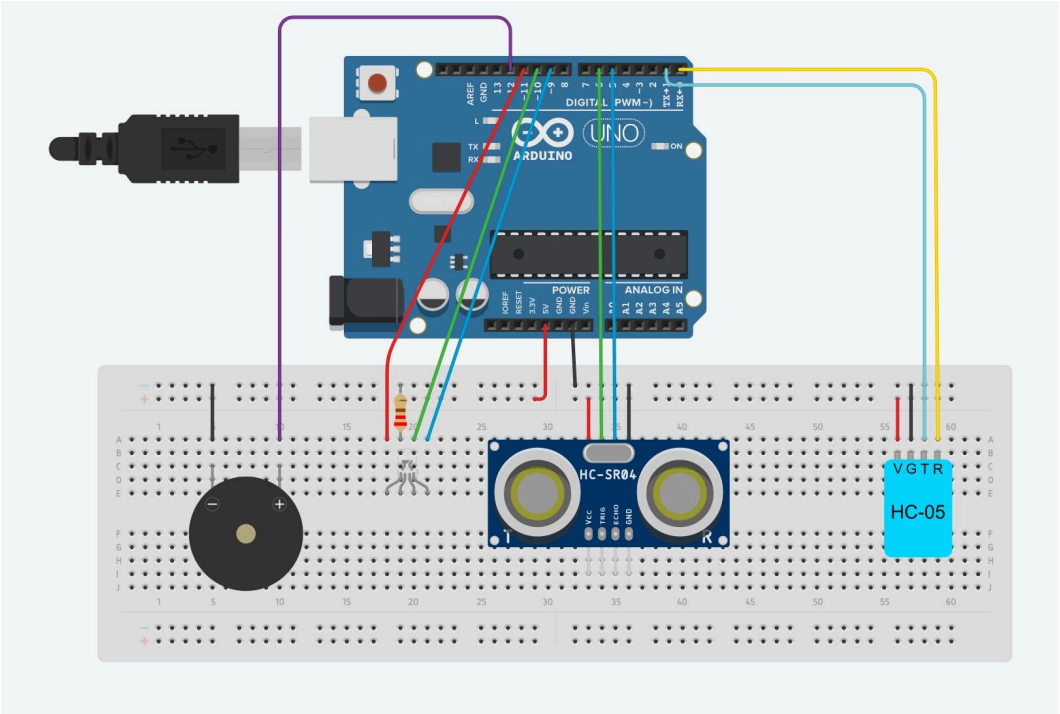

Diagram of the Sensor Arduino

Wiring

1. Arduino 5V and GND to positive power rail and negative power rail respectively
2. Positive and negative power rail to HC-05 VCC and GND pins respectively
3. Arduino D1 (TX) pin to HC-05 TX pin
4. Arduino D0 (RX) pin to HC-05 RX pin
5. Positive and negative power rail to HC-SR04 VCC and GND pins respectively
6. Arduino D6 pin to HC-SR04 TRIG pin
7. Arduino D5 pin to HC-SR04 ECHO pin
8. Arduino D11 pin to LED red pin
9. Arduino D10 pin to LED green pin
10. Arduino D9 pin to LED blue pin
11. Negative power rail to LED cathode pin with 220 Ohms resistor
12. Arduino D12 pin to buzzer positive pin

13. Negative power rail to buzzer negative pin

Code

```
// Javid Uddin, muddi7
// Jason Liang, jlian21
// Project Name: SpotterSense - Parking Lot Availability Sensor
// (1/2) SENSOR ARDUINO

#include <SoftwareSerial.h>

// sensor & parking lot ID
const int id = 1;

// Initialize a software serial connection for BT communication
SoftwareSerial bt(1, 0); // RX, TX

// Ultrasonic sensor pins
const int trigPin = 6, echoPin = 5;

// Variables for timing
unsigned long prevPulseTime = 0, prevLoopTime = 0, prevUpdateTime = 0,
prevChangeTime = 0, prevAlertTime = 0;

// RGB LED pins and buzzer pin
const int redPin = 11, greenPin = 10, bluePin = 9, buzzerPin = 12;

// Alert interval for buzzer sound (in milliseconds)
int alertInterval = 0;

// Distance-related variables
int objDistance, prevDistance = -1; // Object distance and previously
recorded distance
bool alertsOn = false; // Tracks whether alerts (LED/buzzer) are active

// Function to wait for a specific number of microseconds
void wait_micros(unsigned long interval) {
```

```
while (micros() - prevPulseTime < interval) {} // Wait until the interval
has elapsed
prevPulseTime = micros(); // Update the previous pulse timestamp
}

// Function to calculate the distance using the ultrasonic sensor
void calculate_dist() {
  digitalWrite(trigPin, LOW); // Ensure trigger pin is low
  wait_micros(2); // Wait for 2 microseconds

  digitalWrite(trigPin, HIGH); // Send a 10-microsecond pulse
  wait_micros(10);
  digitalWrite(trigPin, LOW); // Stop the pulse

  // Measure the duration of the echo pulse
  unsigned long duration = pulseIn(echoPin, HIGH);

  // Convert the pulse duration to distance (in cm)
  objDistance = duration * 0.034 / 2;
}

// Function to set the RGB LED color
void write_color(int r, int g, int b) {
  analogWrite(redPin, r);   // Set red component
  analogWrite(greenPin, g); // Set green component
  analogWrite(bluePin, b);  // Set blue component
}

// Function to determine alert behavior based on distance
void set_alerts() {
  if (objDistance >= 182) {
    // Turn off alerts if the object is too far
    alertsOn = false;
    disable_alerts();
  }
  else if (objDistance > 60) {
    // Blue LED indicates medium proximity
    // Normal buzzer interval
    write_color(0, 0, 255);
    alertInterval = 1500;
```

```
  }
  else if (objDistance > 30) {
    // Green LED indicates close proximity
    // No buzzer
    write_color(0, 255, 0);
    alertInterval = -1;
  }
  else {
    // Red LED indicates very close proximity
    // Fast buzzer interval
    write_color(255, 0, 0);
    alertInterval = 1000;
  }
}

// Function to disable alerts (turn off LED and buzzer)
void disable_alerts() {
  write_color(0, 0, 0); // Turn off RGB LED
  alertInterval = -1;   // Disable buzzer
}

// Function to send parking status via Bluetooth
void send_status() {

  char status[10];

  if (objDistance > 280) {
    // Send "available" status if the object is far away
    snprintf(status, "%d:0\n", id);
  }
  else if (objDistance < 200) {
    // Send "occupied" status if the object is close
    snprintf(status, "%d:1\n", id);
  }

  bt.write(status);
}

void setup() {
  // Initialize ultrasonic sensor pins
```

```
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  // Initialize RGB LED pins
  pinMode(redPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  pinMode(greenPin, OUTPUT);

  // Start Bluetooth communication
  bt.begin(9600);
}

void loop() {
  unsigned long currTime = millis(); // Get the current time

  // Update distance calculation every second
  if (currTime - prevLoopTime >= 1000) {
    calculate_dist(); // Calculate the distance using the ultrasonic
sensor

    // Check if the distance has remained stable
    if (abs(objDistance - prevDistance) < 10) {
      // If the distance hasn't changed for 5 seconds, disable alerts
      if (currTime - prevChangeTime > 5000) {
        alertsOn = false;
        disable_alerts();
      }
    } else {
      // If the distance has changed, enable alerts and update the
previous change time
      alertsOn = true;
      set_alerts();
      prevChangeTime = millis();
    }

    prevLoopTime = millis(); // Update the loop timestamp
  }

  currTime = millis();
```

```
  // Send the parking status every 5 seconds
  if (currTime - prevUpdateTime > 5000) {
    if (abs(objDistance - prevDistance) > 10) {
      prevDistance = objDistance; // Update the previous distance if it
has changed
    }

    send_status(); // Send the current parking status
    prevUpdateTime = millis(); // Update the update timestamp
  }

  // Handle alerts (buzzer and LED)
  if (alertsOn) {
    currTime = millis();

    // Trigger the buzzer based on the alert interval
    if (alertInterval > 0 && currTime - prevAlertTime >= alertInterval) {
      tone(buzzerPin, 50, 1000); // Play a tone on the buzzer
      prevAlertTime = millis(); // Update the alert timestamp
    }
  }
}
```

Explanation

The sensor Arduino's main role is to detect the distance of objects through the HCSR-04 ultrasonic sensor. In simplest terms, the ultrasonic sensor functions by sending out ultrasonic pulses through the TRIG pin for 10 microseconds, then listening for reflections using the ECHO pin. By reading the time it took to receive a reflection pulse, we're able to calculate the distance an object is away from the sensor. This measurement is then done once every second.

Once the distance is calculated, it is compared with the distance measured from 5 seconds ago. This comparison is done to calculate when to turn off our parking guides, the RGB LED and buzzer. If the current distance and previous distance is within 10 cm, the LED and buzzer are

turned off. Otherwise, according to the distance measured, the LED and buzzer will be active with a different color and buzz interval respectively.

The final major role of the sensor Arduino is to communicate its spot's status back to the central hub. Like previously mentioned, this is done via a HC-05 Bluetooth module. Every 5 seconds, a simple string containing the sensor's ID and spot status is sent.

Central Arduino:

Materials

- Arduino UNO
- HC-05 Bluetooth module
- 1x 220 Ohms resistor
- 2x 10k Ohms resistor
- 2x Pushbuttons
- Potentiometer
- 16x2 LCD screen
- Wires
- Breadboard

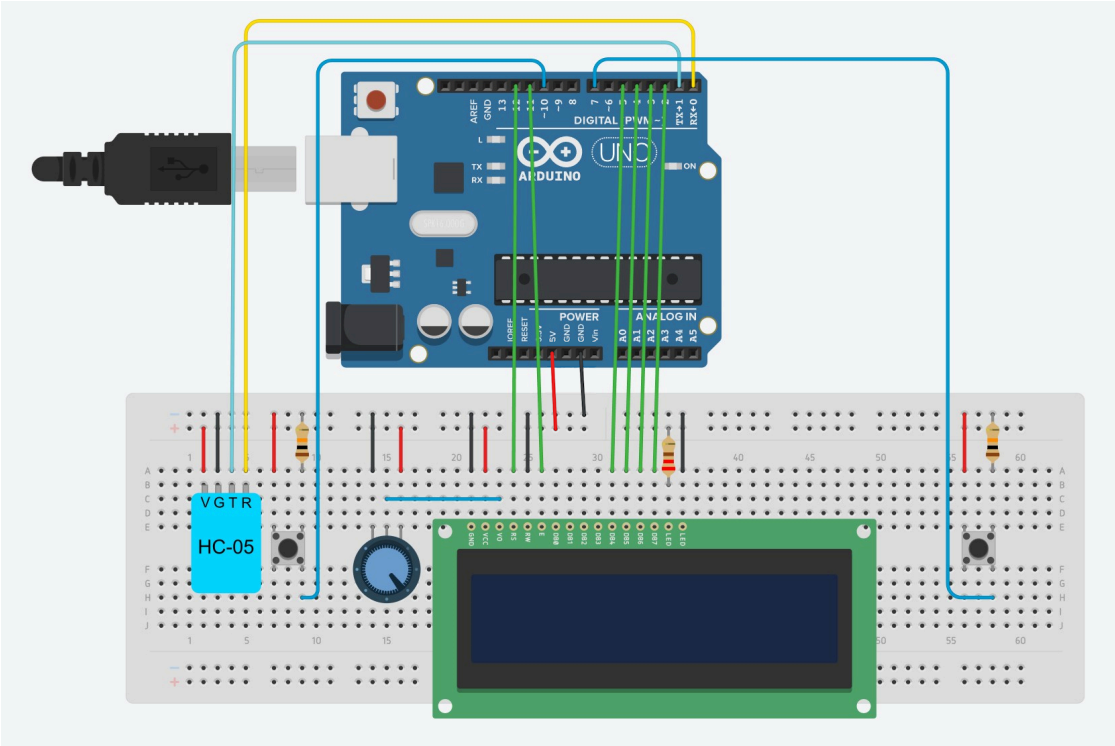

Diagram of the Central Arduino

Wiring

1. Arduino 5V and GND to positive power rail and negative power rail respectively
2. Positive and negative power rail to HC-05 VCC and GND pins respectively
3. Arduino D1 (TX) pin to HC-05 TX pin
4. Arduino D0 (RX) pin to HC-05 RX pin
5. Potentiometer left pin to negative power rail
6. Potentiometer right pin to positive power rail
7. Potentiometer middle pin to LCD VO
8. LCD A(anode) to to positive power rail with 220 Ohms resistor
9. LCD K(cathode) to negative power rail
10. LCD GND and RW to negative power rail
11. LCD VCC to positive power rail
12. LCD RS to Arduino D12
13. LCD E to Arduino D11
14. LCD DB4 to Arduino D5
15. LCD DB5 to Arduino D4

16. LCD DB6 to Arduino D3
17. LCD DB7 to Arduino D2
18. Both buttons:
    a. Side A pin 1 to positive power rail
    b. Side A pin 2 to negative power rail with 10k Ohms resistor
    c. Side B pin 2 (opposite of Side A pin 2) to Arduino D10
        i. Only for one of the button, the other button will connect to D7

Code

```
// Javid Uddin, muddi7
// Jason Liang, jlian21
// Project Name: SpotterSense - Parking Lot Availability Sensor
// (2/2) CENTRAL ARDUINO

#include <LiquidCrystal.h> // Library for controlling the LCD
#include <SoftwareSerial.h> // Library for software-based serial
communication
#include <string.h>         // Library for string manipulation

// LCD pin assignments (RS, Enable, D4, D5, D6, D7)
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

// Button pin assignments
const int buttonPin1 = 10; // Button to move left in the menu
const int buttonPin2 = 7;  // Button to move right in the menu

// Array to track parking spot availability (0 = available, 1 =
unavailable)
int spots[5] = {0, 0, 0, 0, 0};

// Current index of the selected parking spot
int currentIndex = -1;

// LCD text lines
String topLine;
String bottomLine;
```

```
// Start screen variables
const String scrollMessage = "Press any button to continue"; // Scrolling message on the start screen
int startScreenVisible = 1; // Flag to indicate if the start screen is being displayed
unsigned long scrollStart = 0; // Timestamp for scrolling message
const unsigned long scrollInterval = 500;  // Time interval between scroll updates (ms)
int scrollPosition = 0; // Current position in the scroll message

// Button state variables
int buttonState = 0; // Current state of the buttons
int prevButtonTime = 0; // Timestamp of the last button press
int prevButtonState = 0; // Previous button state

// Bluetooth serial communication (RX, TX)
SoftwareSerial btSerial(1, 0); // Connect RX of Bluetooth module to pin 10, TX to pin 9

// Function to calculate the total number of parking spots
int getSpotsCount() {
  return sizeof(spots) / sizeof(spots[0]);
}

// Handles button presses to navigate the parking spots
void handleBtn(int buttonReading) {
  if (currentIndex == -1) {
    // If no spot is selected, start at the first spot
    currentIndex = 0;
  } else if (currentIndex + buttonReading >= 0 && currentIndex + buttonReading < getSpotsCount()) {
    // Move to the next/previous spot if within bounds
    currentIndex += buttonReading;
  } else if (currentIndex + buttonReading < 0) {
    // Wrap around to the last spot if moving left from the first spot
    currentIndex = getSpotsCount() - 1;
  } else if (currentIndex + buttonReading >= getSpotsCount()) {
    // Wrap around to the first spot if moving right from the last spot
    currentIndex = 0;
```

```
  }
  displayStatus(); // Update the LCD display
}

// Updates the LCD display to show the current parking spot status
void displayStatus() {
  topLine = "Parking Spot #" + String(currentIndex + 1); // Display the
spot number
  bottomLine = spots[currentIndex] == 0 ? "Available" : "Unavailable"; //
Display availability status

  lcd.clear(); // Clear the LCD
  lcd.setCursor(0, 0);
  lcd.print(topLine); // Display the top line
  lcd.setCursor(0, 1);
  lcd.print(bottomLine); // Display the bottom line
}

// Displays the initial start screen with a scrolling message
void showStartScreen() {
  if (millis() - scrollStart >= scrollInterval) { // Check if it's time to
update the scroll
    lcd.clear();

    lcd.setCursor(0, 0);
    lcd.print("SpotterSense"); // Display the project name
    scrollStart = millis();  // Reset scroll timer

    // Display a portion of the scrolling message on the bottom line
    lcd.setCursor(0, 1);
    lcd.print(scrollMessage.substring(scrollPosition, scrollPosition +
16));

    // Update the scroll position and wrap around if the end is reached
    scrollPosition++;
    if (scrollPosition > scrollMessage.length() - 16) {
      scrollPosition = 0;
    }
  }
}
```

```
void setup() {
  pinMode(buttonPin1, INPUT); // Set button 1 as input
  pinMode(buttonPin2, INPUT); // Set button 2 as input
  lcd.begin(16, 2); // Initialize the LCD with 16x2 dimensions
  btSerial.begin(9600); // Initialize Bluetooth serial communication at
9600 baud
}

void loop() {
  if (startScreenVisible) {
    showStartScreen(); // Display the start screen if it is visible
  }

  // Read the button states
  int buttonReading1 = digitalRead(buttonPin1);
  int buttonReading2 = digitalRead(buttonPin2);
  int buttonReading = buttonReading2 - buttonReading1; // Determine button
direction (-1, 0, 1)

  // Handle button presses with debounce logic
  if (millis() - prevButtonTime >= 200) { // 200 ms debounce
    if (buttonReading != prevButtonState) {
      prevButtonTime = millis(); // Update the debounce timer
      startScreenVisible = 0; // Hide the start screen after any button
press
      handleBtn(buttonReading); // Handle the button action
    }
  }
  prevButtonState = buttonReading; // Update the previous button state

  // Check for Bluetooth input and update parking spot statuses
  if (btSerial.available() > 0) {
    String input = btSerial.readStringUntil('\n'); // Read input until a
newline
    input.trim(); // Remove any leading/trailing whitespace

    // Parse the input for spot number and status
    int spotNumber = input[0] - '0' - 1; // Convert the first character to
a spot index
```

```
    int spotStatus = input[2] - '0'; // Convert the third character to a
status (0/1)
    spots[spotNumber] = spotStatus; // Update the spot status

    if (!startScreenVisible) {
      displayStatus(); // Update the LCD if the start screen is not
visible
    }
  }
}
```

Explanation

The main role of the Arduino server is to act as the central hub in the SpotterSense parking lot availability system. It handles user interaction, displays parking spot statuses, and communicates with any other sensor-equipped Arduinos via Bluetooth (in our case, there was only 1 Sensor Arduino).

When powered on, the Arduino server begins with a start screen on the LCD display. This screen scrolls a welcome message, prompting the user to press any button to continue. Two buttons are used for user input:

1) Button 1: Switches to the *previous* parking spot.
2) Button 2: Switches to the *next* parking spot.

The **handleBtn()** function ensures the user can navigate through parking spots, wrapping around if the start or end of the list is reached. The parking spot data is stored in an array, where each index represents a spot, and its value indicates whether the spot is available (0) or unavailable (1). The **displayStatus()** function updates the LCD to show the selected parking spot number and its availability (e.g., Available or Unavailable).

The Arduino server communicates with the sensor-equipped Arduinos using a SoftwareSerial Bluetooth module. It listens for status updates sent by the sensor Arduinos, which

include the spot number and availability status. For example, the message "1:1" indicates that parking spot 1 is unavailable. These updates are parsed and stored in the spots array. The server immediately updates the display if a user actively views the updated spot. Additionally, while the start screen is visible, the **showStartScreen()** function scrolls the welcome message. This makes the user interface more engaging while waiting for input. Once the user exits the start screen by pressing a button, the server displays real-time parking spot information based on the data it automatically receives from the sensor Arduinos.

Bluetooth Pairing:

While the project could theoretically function through wired serial communication, Bluetooth or some sort of wireless communication is required in order to maximize the flexibility of the project. This could be done if the Arduino boards used support native wireless communication, but our project uses boards that do not, hence the inclusion of the HC-05 modules.

Every sensor Arduino must be equipped with the module and said module must be set as a slave, while the central Arduino is set as the master. Our modules are paired by following a tutorial and is listed in our references.

7. User Guide

The expected usage of SpotterSense is quite simple. Place a sensor Arduino at each parking spot and have the central Arduino at the entrance of the parking lot. Each sensor Arduino should have an ID number ranging from 1 to the total number spots. Currently, our prototype supports up to 5 parking spots, but it can be increased by simply changing the size of the **spots[]**

array. Once all sensor Arduinos are placed and connected with the central hub, all setup is complete. When powered, the sensors would be able to detect if any cars are in a spot, and statuses could be seen on the central hub's LCD screen.

The LCD screen can be controlled with 2 buttons that traverse in opposite directions. The display will start with a start message. Pressing either button will display the status of a spot (first or last spot depending on which button was pressed). Then using the buttons once again, the user can navigate through all spots. The start screen should not be displayed again until powered off and on again.

Development Timeline

- ➔ Week of 11/1: Began and finalized Arduino circuit design.
- ➔ Week of 11/8: Begin assembling and coding individual Arduino parking sensors; implement and test ultrasonic sensor measurements.
- ➔ Week of 11/15: Finalize Bluetooth communication protocol between Arduinos
- ➔ Week of 11/22: Design Presentation (11/22); test Bluetooth communication and LCD output.
- ➔ Week of 11/29: Final testing and fine-tuning of detection and update timings.

References

- Bluetooth pairing - https://lastminuteengineers.com/hc05-master-slave-arduino-tutorial/
- HC-05 module debugging - https://forum.arduino.cc/t/sending-a-value-between-arduinos-using-hc-05-bluetooth-modules/859376/22
- Ultrasonic sensor usage - https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/
- RGB LED usage - https://arduinogetstarted.com/tutorials/arduino-rgb-led
- LCD Display setup - https://docs.arduino.cc/learn/electronics/lcd-displays/

- SoftwareSerial - https://docs.arduino.cc/learn/built-in-libraries/software-serial/
- TinkerCad Documentation - https://www.tinkercad.com/learn