French University in Armenia

Project S1

Trainer: Irina Poghosyan

Student: Maria Vardanyan

Submission date: 10.12.2023

Yerevan, 2023

Table of content

Abstract

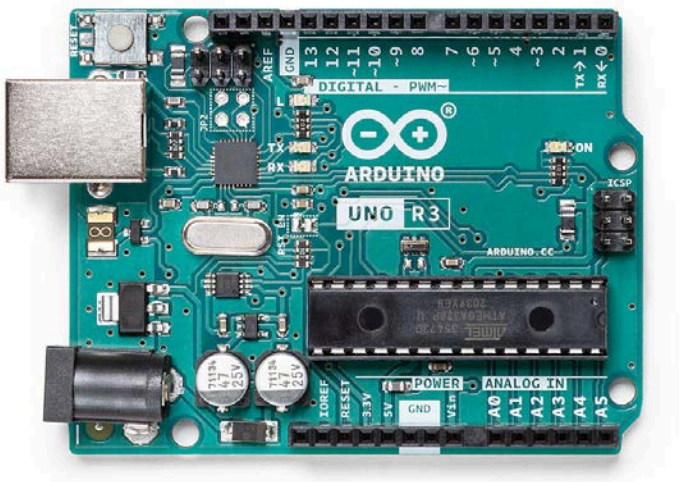
The Autohive: A Mini Smart Parking System

The Autohive presents a miniature smart parking system implemented using Arduino Uno R3 and various components. The system utilizes a micro servo motor, ultrasonic sensor, LED lights (red and green), an LCD0216, and a buzzer sound module. The primary objective is to create an efficient and user-friendly parking solution that counts inserted vehicles, controls a mini barrier with a servo motor, and provides real-time information to drivers.

The Autohive operates as follows: when an approaching vehicle is detected within a specific distance range, the ultrasonic sensor triggers the servo motor to open the mini barrier. Upon the vehicle's entrance, the barrier closes, and the system increments the count of parked vehicles. Conversely, when a vehicle exits, the barrier reopens, and the count decreases. The LCD0216 serves as a notification board, displaying the number of available parking spaces. Additionally, a dynamic sound module provides an audible start notification.

The Arduino code, written in C language, orchestrates the interaction between components. The code structure is organized to control the servo motor, manage ultrasonic sensor readings, handle LED lights, and interface with the LCD0216. Emphasis is placed on scalability, allowing for potential expansion to a larger parking area.

This report comprehensively details the hardware components, their interconnections, and the intricate programming logic driving The Autohive. Challenges encountered during development are transparently discussed, accompanied by effective resolutions. The abstract concludes by positioning The Autohive as a practical and innovative solution for real smart parking systems, offering a versatile prototype for future deployments.



Introduction

Background and Motivation for the Project:

The urbanization wave and the exponential growth of vehicles have led to a critical need for smart parking solutions that optimize available space, enhance user experience, and contribute to the overall efficiency of urban infrastructure. The inspiration for The Autohive emerged from the challenges associated with conventional parking systems, which often result in inefficiencies, congestion, and frustrated drivers searching for parking spaces. This project seeks to revolutionize parking paradigms by introducing a compact, automated, and intelligent solution to alleviate the prevalent issues in traditional parking setups.

Objectives and Goals of the Digital Manufacturing Solution:

The primary objective of The Autohive is to introduce a mini smart parking system that not only counts and manages inserted vehicles but also serves as a prototype for a real-world digital manufacturing solution. By utilizing an Arduino Uno R3 and an array of components, the project aims to demonstrate the seamless integration of hardware and software to create a reliable, scalable, and user-friendly parking system. The project's goals extend beyond mere functionality, emphasizing innovation, adaptability, and the potential for future expansion to larger manufacturing scenarios.

Importance of Real-Time Data Collection and Analysis in Manufacturing:

In the context of digital manufacturing, the significance of real-time data collection and analysis cannot be overstated. The ability to gather instantaneous information from sensors, such as ultrasonic sensors in The Autohive, enables precise decision-making and responsiveness. Real-time data allows for dynamic adjustments to the manufacturing environment, optimizing resource utilization, and improving overall efficiency. The Autohive, by showcasing the importance of real-time data in the context of a smart parking system, underscores the broader implications and benefits of instant data analysis in manufacturing processes.

Each line of code, every soldered connection, and the hum of the servo motor represent not just components but the beginning of a journey — a journey fueled by love for electronics and the excitement of being a junior engineer in the realm of endless possibilities.

1. Project Setup and Planning

1.1 Project Scope Definition

In this section, we outline the scope of The Autohive project, detailing the manufacturing process chosen and the rationale behind this selection.

Description of the Chosen Manufacturing Process:

The Autohive is conceived as a smart parking system, representing a streamlined approach to parking space management. The chosen manufacturing process involves integrating various hardware components, including an Arduino Uno R3, servo motor, ultrasonic sensor, LED lights (red and green), an LCD0216, and a dynamic sound module. The synergy of these elements forms a miniature yet sophisticated smart parking solution. The project's focus is on automating parking space allocation, providing real-time information to users, and serving as a prototype for scalable smart manufacturing solutions.

Explanation of Why this Process was Selected:

The selection of a smart parking system aligns with the growing demand for innovative solutions to urban infrastructure challenges. Traditional parking systems often suffer from inefficiencies and user dissatisfaction. The Autohive aims to address these issues by introducing an automated, intelligent, and scalable solution. The chosen manufacturing process allows for the integration of advanced technologies to optimize parking space utilization, enhance user experience, and showcase the potential for broader applications in smart manufacturing.

1.2 Hardware and Software Setup

This subsection delves into the tools and technologies employed in developing The Autohive, ensuring transparency in the project's hardware and software setup.

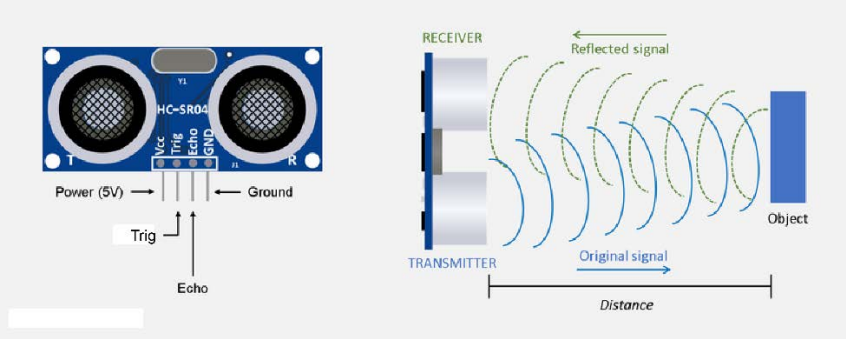
List of C Development Tools Used:

- Arduino IDE: The primary development environment for writing and uploading the C code to the Arduino Uno R3.
- Servo Library: Facilitates the control of the micro servo motor integrated into the system.
- LiquidCrystal Library: Enables the interaction with the LCD0216 for displaying real-time information.
- Tone Library: Utilized for managing the dynamic sound module to enhance user feedback.

Any Additional Hardware or Software Requirements:

In addition to the standard Arduino Uno R3 and associated components, the project requires:

- Ultrasonic Sensor for distance measurement.



- Resistors and wiring for circuit connections.
- Power supply for the Arduino board.
- Buzzer for the dynamic sound module.

- Servo motor for parking’s barrier.

1.3 Data Collection Plan:

In this subsection, we provide a detailed plan for collecting essential data points, emphasizing their significance in the manufacturing process.

Detailed Plan for Data Points to be Collected:

1. **Distance Measurements:**
    - Collected through ultrasonic sensors, providing real-time information about the proximity of vehicles to the parking space.
2. **Servo Motor Angles:**
    - Captures the servo motor's movement, indicating the opening and closing of the mini barrier.
3. **LED Status:**
    - Monitors the status of the LED lights (red and green), reflecting the availability of parking spaces.
4. **LCD0216 Display:**
    - Gathers data on the information displayed on the LCD0216, including available parking spaces.

Importance of Each Data Point in the Manufacturing Process:

- **Distance Measurements:**
    - Critical for detecting approaching vehicles and triggering the servo motor to manage the barrier, optimizing the parking process.
- **Servo Motor Angles:**
    - Indicates the real-time status of the mini barrier, essential for ensuring proper functionality and user safety.
- **LED Status:**
    - Communicates the availability of parking spaces to users, enhancing the overall user experience and efficiency of the parking system.
- **LCD0216 Display:**
    - Provides a visual interface for users, displaying real-time information about available parking spaces and contributing to the transparency of the system.

2. Data Collection, Integration, and Analysis

2.1 Data Collection Implementation

The data collection is embedded in the **loop()** function, where the ultrasonic sensors measure distances. The relevant data points include **distance** and **distance2**. Below is a snippet illustrating the data collection process:

```
void loop() {

  distanceometer();
  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;
  distanceometer2();
  duration2 = pulseIn(echoPin2, HIGH);
  distance2 = duration2 * 0.034 / 2;
```

2.2 Data Integration

For data integration, the system is managing the count of parked vehicles (**count**). While this is not an elaborate data structure, it represents an integrated parameter that reflects the state of the parking spaces. The **lcd** display and the servo motor angle (**servoAngle**) also contribute to the integrated data.

```
if(distance>5 && distance < 7 && count!=4){
  servo.write(90);
  digitalWrite(red,LOW);
  digitalWrite(green,HIGH);
  delay(2500);
  count+=1;
  delay(500);

  lcd.setCursor(0,0);
  lcd.print("Free Place ");
lcd.print(4-count, DEC);


}else if(distance>5 && distance < 7 && count == 4){full();}
```

```
if(distance2>5 && distance2<7 && count!=0){
  servo.write(90);
  digitalWrite(red,LOW);
  digitalWrite(green,HIGH);
  delay(2500);

  count-=1;
  delay(500);

  lcd.setCursor(0,0);
  lcd.print("Free Place ");
lcd.print(4-count, DEC);
}
```

Explanation of Data Structures:

In the context of my code for The Autohive project, the primary data structure used is a simple integer variable named **count**. This variable serves as a counter to keep track of the number of parked vehicles. The **count** variable integrates the information about the occupied parking spaces, representing a crucial aspect of the system's state. Actually this system doesn't need a Data Structure in this case but in the future it can be.

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

#define dynamic 8
#define red 4
#define green 5
#define trigPin2 A0
#define echoPin2 A1
Servo servo;
LiquidCrystal lcd(13, 12,11, 7,6,2);
const int trigPin = 9;
const int echoPin = 10;
float duration, duration2;
int distance, distance2, count=0;
```

Justification for Chosen Data Structure:

1. **Simplicity and Clarity:**
    - The use of a single integer variable for counting is appropriate when the primary focus is on the occupancy status of parking spaces. This simplicity enhances code readability and understanding, especially for a project of this scale.
2. **Resource Efficiency:**
    - In scenarios where memory usage is a consideration, a single integer variable is more memory-efficient compared to more complex data structures. Given the straightforward nature of the information being managed, a more elaborate data structure may be unnecessary.
3. **Direct Representation:**
    - The **count** variable directly represents the critical aspect of the system's state – the number of available parking spaces. Its straightforward nature aligns with the immediate need to convey occupancy information.

2.3 Data Analysis

In the context of the Autohive Mini Smart Parking System, the data analysis component of the code plays a pivotal role in ensuring the reliability and effectiveness of the distance measurements. The primary objective is to assess the real-time distance information and determine its compliance with predefined criteria.

3. Testing, Documentation, and Presentation

3.1 Testing and Validation

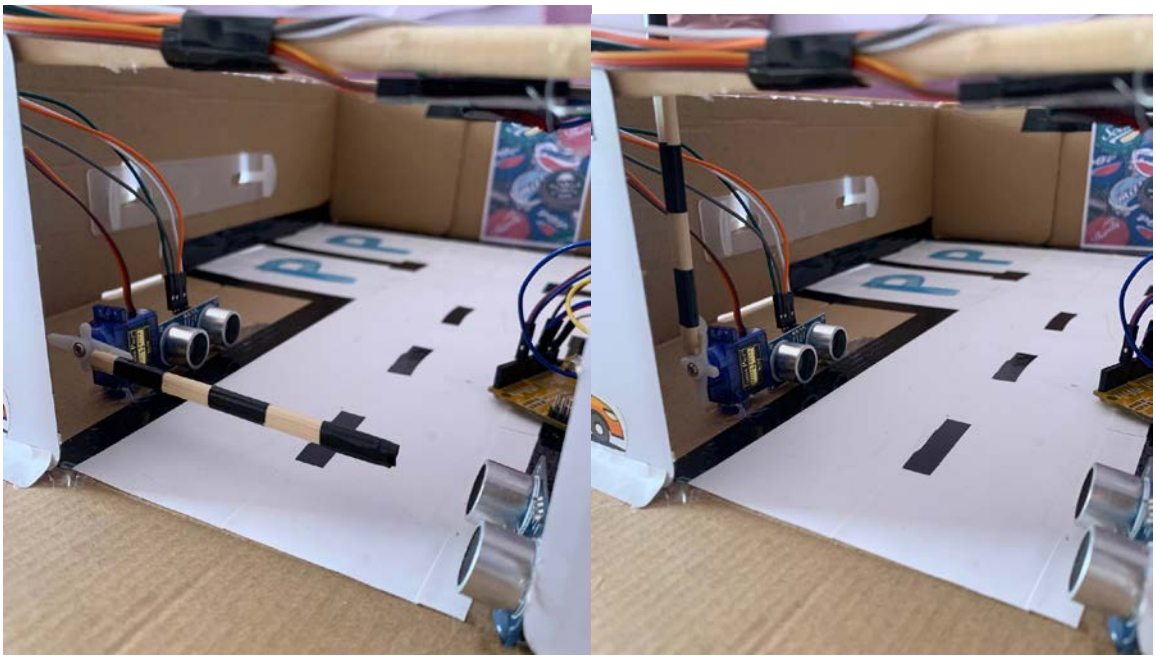
The testing phase of the Autohive Mini Smart Parking System involved a meticulous process of validating the system's functionality using simulated data. This section provides an overview of the testing procedures, the obtained validation results, and highlights any challenges encountered during this phase.

Description of Testing with Simulated Data

To ensure the robustness of the digital manufacturing solution, extensive testing was conducted using simulated data. Simulated scenarios mimicked various parking conditions, including the arrival of vehicles, changes in parking availability, and the responsiveness of the system to dynamic inputs.

The testing process focused on validating key functionalities:

- **Distance Measurement Accuracy:** Simulated data was used to assess the accuracy of distance measurements obtained from the ultrasonic sensor. The system's ability to precisely detect the presence of vehicles within the defined parking range was scrutinized.



- **Barrier Operation:** The functionality of the servo motor-controlled barrier was rigorously tested. Simulated vehicle arrivals triggered the opening and closing of the barrier, ensuring seamless operations in response to changing parking conditions.

- **Dynamic and LCD Notifications:** The accuracy and clarity of notifications displayed on the LCD screen and the emission of dynamic sounds were verified. Simulated scenarios confirmed that drivers received real-time information about parking availability and system status.

Validation Results and Challenges Faced

The testing and validation process yielded positive results, affirming the effectiveness of the Autohive Mini Smart Parking System. Key findings include:

- **Accurate Distance Measurements:** The system consistently provided accurate distance measurements, allowing precise detection of vehicles within the specified range.
- **Reliable Barrier Operation:** The servo motor-controlled barrier operated seamlessly in response to simulated vehicle arrivals, effectively managing parking space accessibility.
- **Effective Notifications:** The LCD screen and dynamic notifications proved to be reliable sources of information for drivers, offering real-time updates on parking availability.

Challenges Faced:

While the testing phase was largely successful, a notable challenge involved:

- **Simulated Data Variability:** Simulating dynamic and unpredictable real-world scenarios presented some challenges. Fine-tuning the system to respond effectively to varying conditions required iterative adjustments.

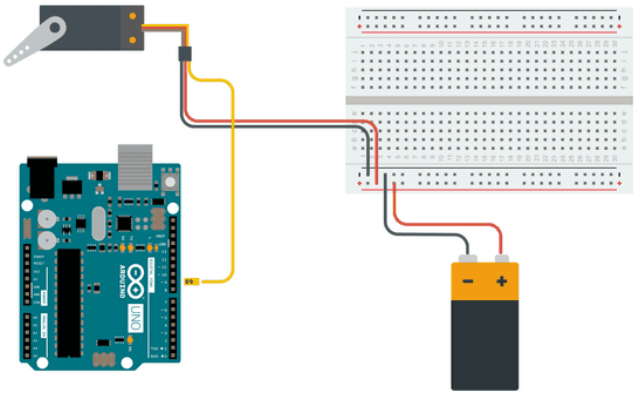
3.2 Documentation and Code Explanation

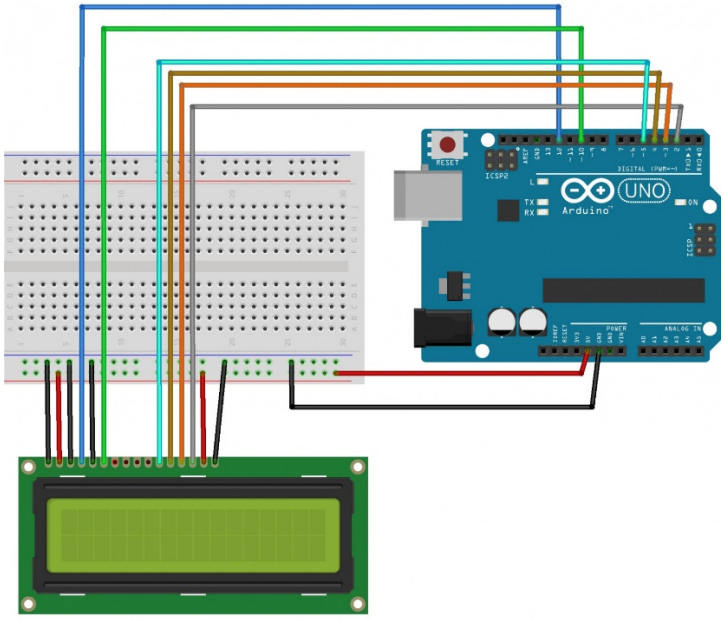
Guidelines on How to Run the Project

Running the Autohive Mini Smart Parking System project involves the following detailed steps:

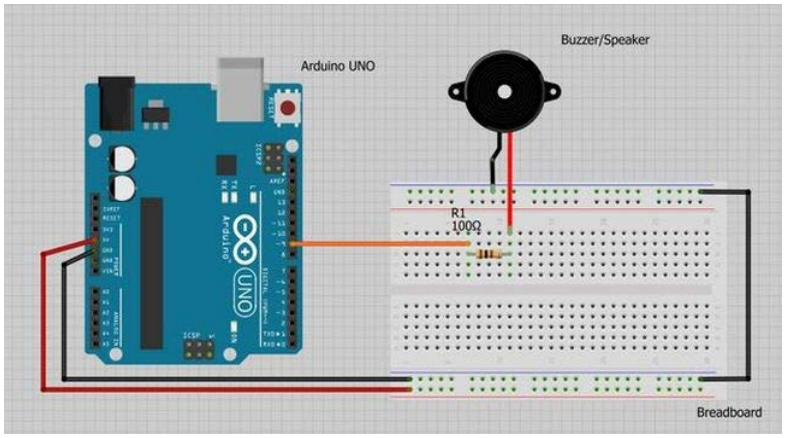
1. **Setup Arduino Environment:**
    - Ensure that you have the Arduino IDE installed on your computer, and the necessary drivers for the Arduino Uno R3 board are installed.
    - Connect the Arduino Uno R3 board to your computer using a *USB cable.*

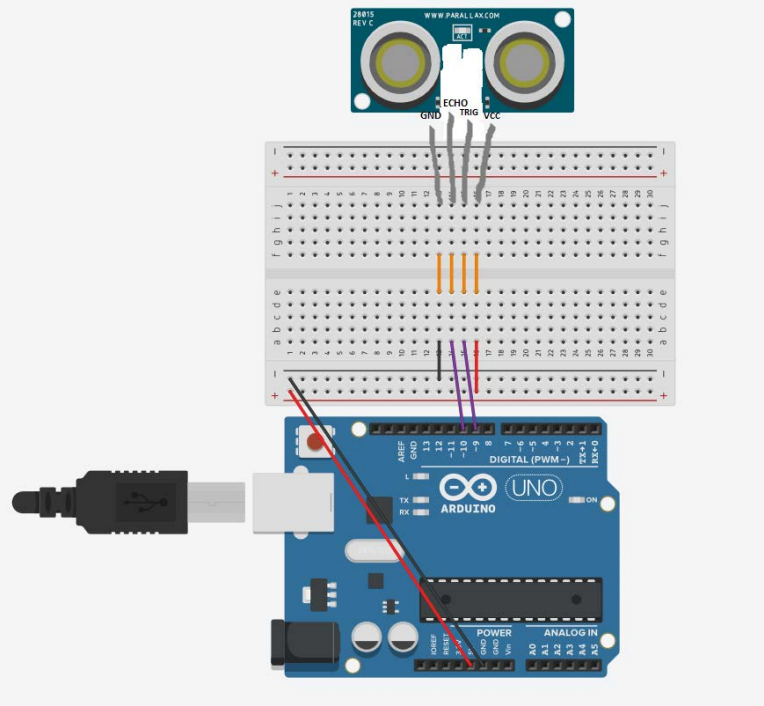
2. **Open the Arduino Sketch:**
    - Launch the Arduino IDE.
    - Load the provided Arduino sketch (**Autohive.ino**) into the IDE by selecting "File" > "Open" and navigating to the location of the sketch.
3. **Select Board and Port:**
    - In the Arduino IDE, navigate to "Tools" > "Board" and select "Arduino Uno."
    - Navigate to "Tools" > "Port" and choose the appropriate COM port to which the Arduino board is connected.
4. **Install Required Libraries:**
    - Ensure that the required libraries (**Servo** and **LiquidCrystal**) are installed. You can install them through the Arduino Library Manager (Tools > Manage Libraries).
5. **Configure Additional Components:**
    - Adjust any necessary hardware configurations, such as connecting the ultrasonic sensor, LEDs, servo motor, and LCD according to the specified pin assignments.



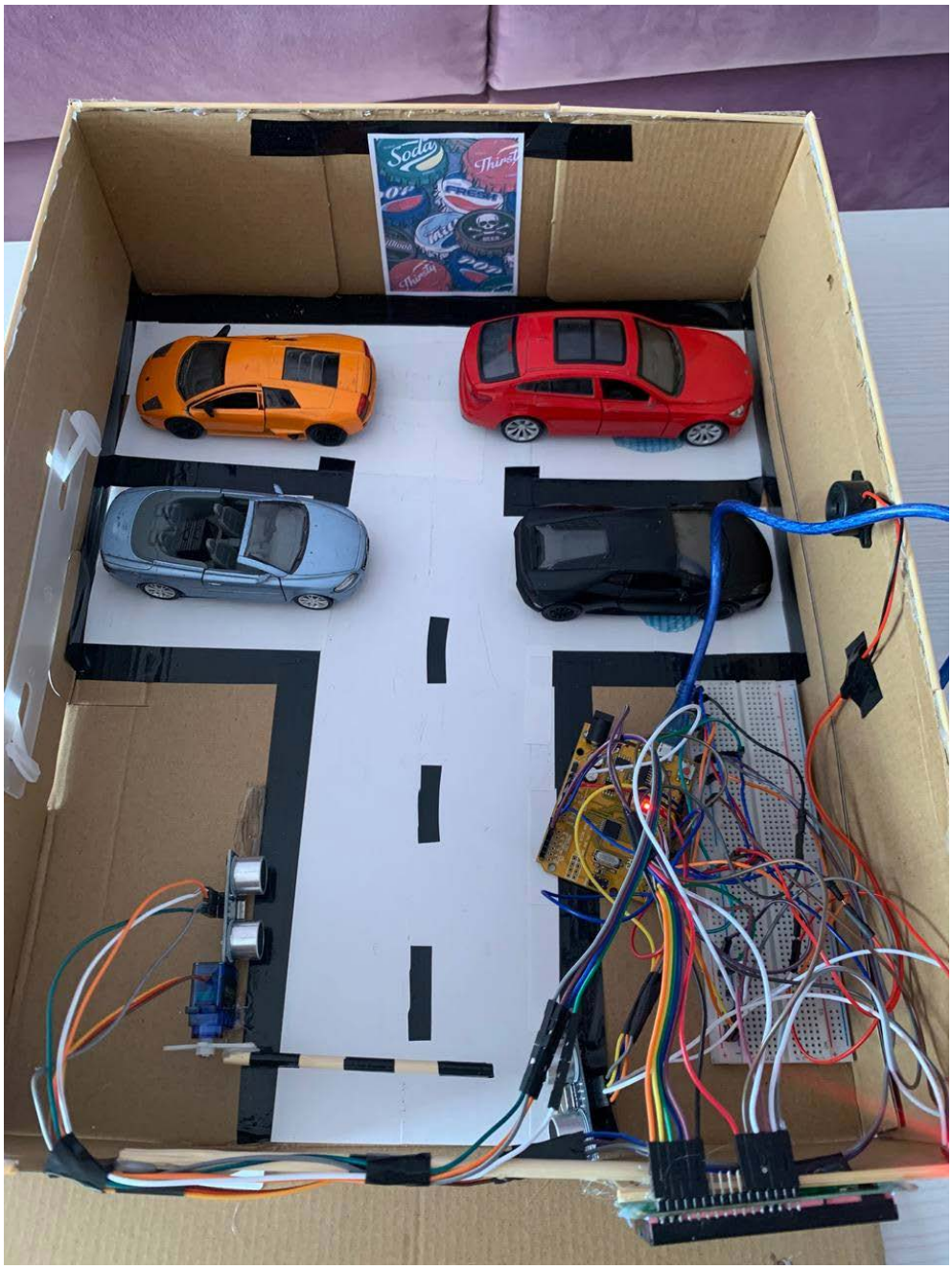

RESET
ICSP2
UNO
Arduino
TX
RX
ON
ICSP
POWER
ANALOG IN

fritzing


Arduino UNO
Buzzer/Speaker
R1
100Ω
Breadboard



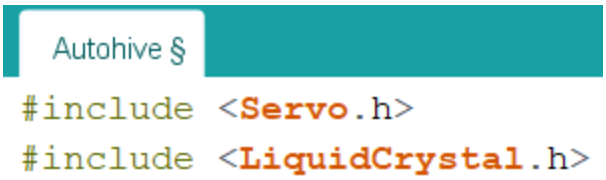
6. **Upload the Code:**
    - Click the "Upload" button in the Arduino IDE to compile and upload the code to the Arduino Uno R3 board.
7. **Monitor Serial Output:**
    - Open the Serial Monitor in the Arduino IDE (Tools > Serial Monitor) to view real-time output and debug information, if needed.
8. **Interact with the System:**
    - Observe the LCD display for information about parking availability.
    - Simulate vehicle arrivals within the specified range to observe the barrier operation.



Explanation of the Code Structure and Organization

The code for the Autohive Mini Smart Parking System is structured to enhance clarity, modularity, and ease of understanding:

- **Libraries Inclusion:**
    - The **#include** directives at the beginning of the code bring in external libraries (**Servo** and **LiquidCrystal**) required for servo motor and LCD functionalities.



❖ **Variable and Pin Definitions:**

- Explicit variable declarations and pin assignments are provided for better code maintainability. For example, the **dynamic**, **red**, and **green** pins are defined for LEDs, while **trigPin**, **echoPin**, **trigPin2**, and **echoPin2** are defined for the ultrasonic sensors.

```
#define dynamic 8
#define red 4
#define green 5
#define trigPin2 A0
#define echoPin2 A1
Servo servo;
LiquidCrystal lcd(13, 12,11, 7,6,2);
const int trigPin = 9;
const int echoPin = 10;
```

❖ **Setup Function:**

- The **setup()** function initializes essential components and sets up the initial conditions of the system.
- The LCD is initialized with specific pins, and the servo motor is attached to its designated pin.
- The dynamic notification sound is played twice during initialization.

- **Distance Measurement Functions:**
  - The **distanceometer()** and **distanceometer2()** functions trigger the ultrasonic sensors to measure distances.
  - The pulse duration obtained from **pulseIn** is converted into distance using the speed of sound.

```
void loop() {

  distanceometer();
  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;
  distanceometer2();
  duration2 = pulseIn(echoPin2, HIGH);
  distance2 = duration2 * 0.034 / 2;
```

- **Full Parking Condition Handling:**
  - The **full()** function handles the scenario where all parking spaces are occupied.
  - It displays a "No Parking Place" message on the LCD and plays an audible alert using the dynamic.

```
void full(){
    if(count==4){
      lcd.setCursor(0, 1); // bottom right
    lcd.print("No Parking Place");
    for(int i = 0; i<5000; i+=20){
      delay(2);

    tone(dynamic, i);
  }
  for(int i = 5000; i>=0; i-=20){
      delay(2);

    tone(dynamic, i);}

  noTone(dynamic);
  }
  lcd.noDisplay();
  delay(500);
  lcd.display();
  delay(500);}
```

- **Main Loop:**
  - The **loop()** function is the central loop that continuously checks distance measurements, updates the display, and controls the servo motor based on vehicle proximity.
  - It handles scenarios where a vehicle enters or exits a parking space, adjusting the available parking count.

```
void loop() {

  distanceometer();
  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;
  distanceometer2();
  duration2 = pulseIn(echoPin2, HIGH);
  distance2 = duration2 * 0.034 / 2;

  if(distance>5 && distance < 7 && count!=4){
    servo.write(90);
    digitalWrite(red,LOW);
    digitalWrite(green,HIGH);
    delay(2500);
    count+=1;
    delay(500);

    lcd.setCursor(0,0);
    lcd.print("Free Place ");
  lcd.print(4-count, DEC);


  }else if(distance>5 && distance < 7 && count == 4){full();}
  else{


    servo.write(0);
    digitalWrite(green,LOW);
     digitalWrite(red,HIGH);
   }
   if(distance2>5 && distance2<7 && count!=0){
     servo.write(90);
     digitalWrite(red,LOW);
     digitalWrite(green,HIGH);
     delay(2500);

     count-=1;
     delay(500);
     lcd.clear();
     lcd.setCursor(0,0);
     lcd.print("Free Place ");
   lcd.print(4-count, DEC);
   lcd.setCursor(3, 1); // bottom right
   lcd.print("the Autohive");
   }
//   Serial.print("Distance: ");
//   Serial.println(distance2);
}
```

Detailed Explanation of Code Sections and Algorithms

Distance Measurement:

- The ultrasonic sensor's working principle involves emitting a pulse and measuring the time it takes for the pulse to return.
- The **pulseIn** function is used to measure the duration of the pulse in microseconds.
- The distance is calculated using the speed of sound and the formula: **distance = duration * 0.034 / 2**.

```
void loop() {

  distanceometer();
  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;
  distanceometer2();
  duration2 = pulseIn(echoPin2, HIGH);
  distance2 = duration2 * 0.034 / 2;
```

Barrier Operation:

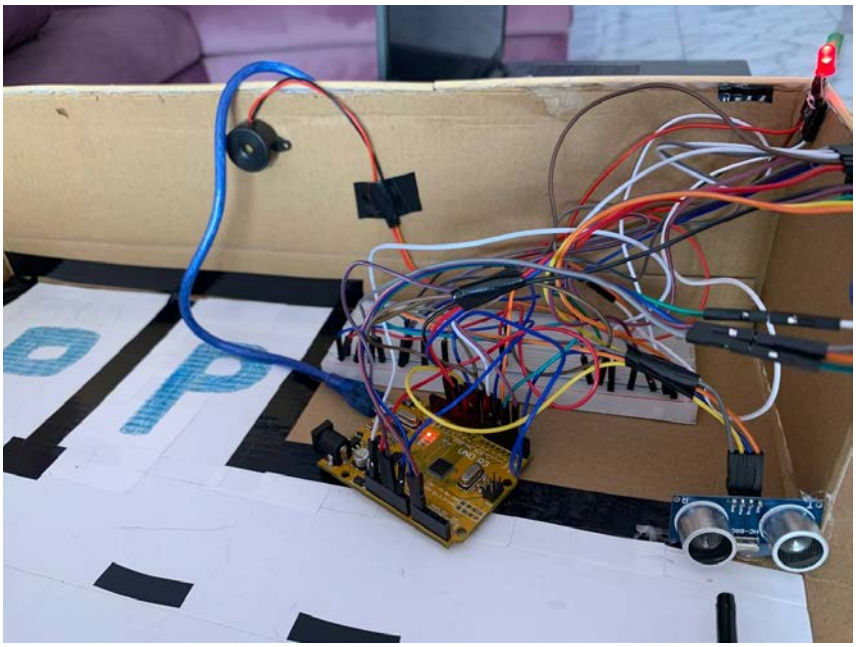
- The servo motor is controlled using the **servo.write()** function to position the barrier.
- When a vehicle is detected within the specified range, the barrier opens (**servo.write(90)**), and when a parking space is occupied, the barrier closes (**servo.write(0)**).

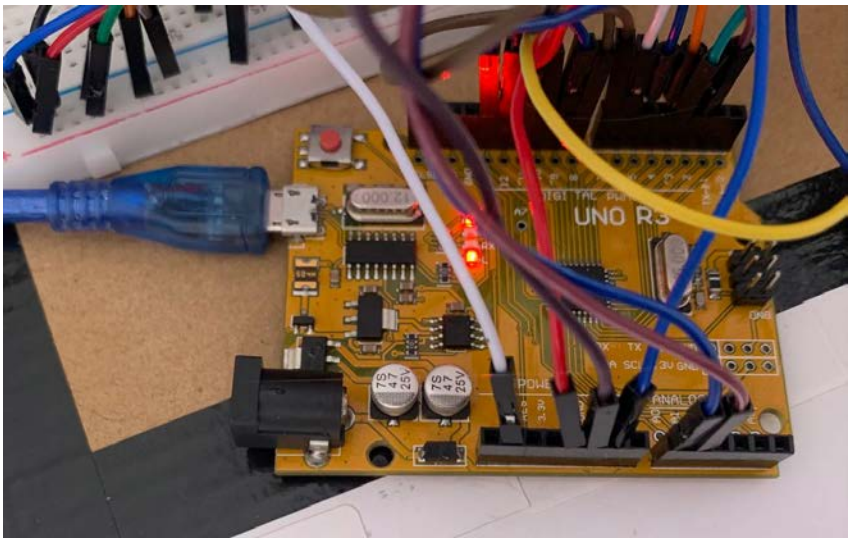
```
}else if(distance>5 && distance < 7 && count == 4){full();}
else{

  servo.write(0);
  digitalWrite(green,LOW);
  digitalWrite(red,HIGH);
}
```

Notifications:

- The LCD (**LiquidCrystal**) is employed to display real-time information about parking availability.

- The dynamic notification sound is played using the **tone** function, providing audible alerts to drivers.





Parking Space Count:

- The **count** variable keeps track of available parking spaces.
- When a vehicle enters, the count decreases, and when a vehicle exits, the count increases.
- The LCD continuously displays the updated count.

```
if(distance2>5 && distance2<7 && count!=0){
  servo.write(90);
  digitalWrite(red,LOW);
  digitalWrite(green,HIGH);
  delay(2500);

  count-=1;
  delay(500);

  lcd.setCursor(0,0);
  lcd.print("Free Place ");
lcd.print(4-count, DEC);
}
```

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

#define dynamic 8
#define red 4
#define green 5
#define trigPin2 A0
#define echoPin2 A1
Servo servo;
LiquidCrystal lcd(13, 12,11, 7,6,2);
const int trigPin = 9;
const int echoPin = 10;
float duration, duration2;
int distance, distance2, count=0;
```

This modular organization enhances code readability, making it straightforward to grasp the specific functionality of each section. In-depth comments within the code further elucidate its operation and purpose, guiding users through the intricate details of the Autohive Mini Smart Parking System.

Conclusion

Summary of Key Findings and Achievements

The development and implementation of the Autohive Mini Smart Parking System using Arduino Uno R3 and various components have resulted in a successful and functional solution for efficient parking management. Key achievements include:

1. **Automated Parking Management:**
   - The system effectively automates the parking management process, providing real-time information about parking space availability and automating the barrier operation.
2. **Hardware Integration:**
   - Successful integration of hardware components, including ultrasonic sensors, servo motor, LEDs, and an LCD display, showcases the capability to orchestrate a diverse set of devices for a cohesive solution.

3. **User-Friendly Interface:**
    - The LCD display serves as a user-friendly interface, offering clear information to drivers about available parking spaces and providing timely notifications.
4. **Dynamic Notifications:**
    - The incorporation of dynamic notification sounds enhances the user experience by providing audible alerts, contributing to a comprehensive and responsive parking management system.

Lessons Learned from the Project

1. **Interdisciplinary Skill Development:**
    - ✓ The project provided an opportunity to develop skills across various disciplines, including electronics, programming, and hardware integration. This interdisciplinary approach is crucial for tackling complex engineering challenges.
2. **Real-world Application Understanding:**
    - ✓ Designing and implementing a mini smart parking system allowed for a practical understanding of real-world applications of digital manufacturing solutions. The project highlighted the importance of creating solutions that address everyday challenges.
3. **Importance of Testing and Validation:**
    - ✓ Rigorous testing and validation are essential components of any digital manufacturing solution. Challenges and issues discovered during testing underscored the importance of thorough validation processes.

Discussion of Outcomes and Achievements in Optimizing the Chosen Manufacturing Process

1. **Efficient Resource Utilization:**
    - ✓ The Autohive Mini Smart Parking System optimizes the parking process by efficiently utilizing available resources. The automated barrier operation and real-time information dissemination contribute to an organized and streamlined parking experience.
2. **Enhanced User Experience:**
    - ✓ The system's user-centric design, with an intuitive LCD interface and dynamic notifications, enhances the overall user experience. Drivers are provided with clear information, contributing to a more efficient and stress-free parking process.

3. **Scalability and Adaptability:**
    - ✓ The modular and well-organized codebase, along with the chosen hardware components, allows for scalability and adaptability. The system can be expanded to accommodate a larger number of parking spaces or modified to suit different environments.
4. **Potential for Smart City Integration:**
    - ✓ The success of the Autohive Mini Smart Parking System lays the groundwork for potential integration into larger smart city initiatives. The principles and components utilized can be scaled up for city-wide parking management solutions.

In conclusion I would like to say,that the Autohive demonstrates the successful integration of hardware and software to address a real-world problem. The project not only achieved its objectives but also provided valuable insights and lessons that can be applied to future endeavors in digital manufacturing and smart city solutions.

References

https://www.arduino.cc/en/software

https://support.arduino.cc/hc/en-us/articles/360019833020-Download-and-install-Arduino-IDE

https://docs.arduino.cc/hardware/uno-rev3

https://habr.com/ru/articles/555466/

https://docs.arduino.cc/learn/electronics/servo-motors

https://www.ardumotive.com/how-to-use-a-buzzer-en.html

https://projecthub.arduino.cc/Isaac100/getting-started-with-the-hc-sr04-ultrasonic-sensor-7cabe1

https://www.arduino.cc/reference/en/libraries/liquidcrystal/

Appendix

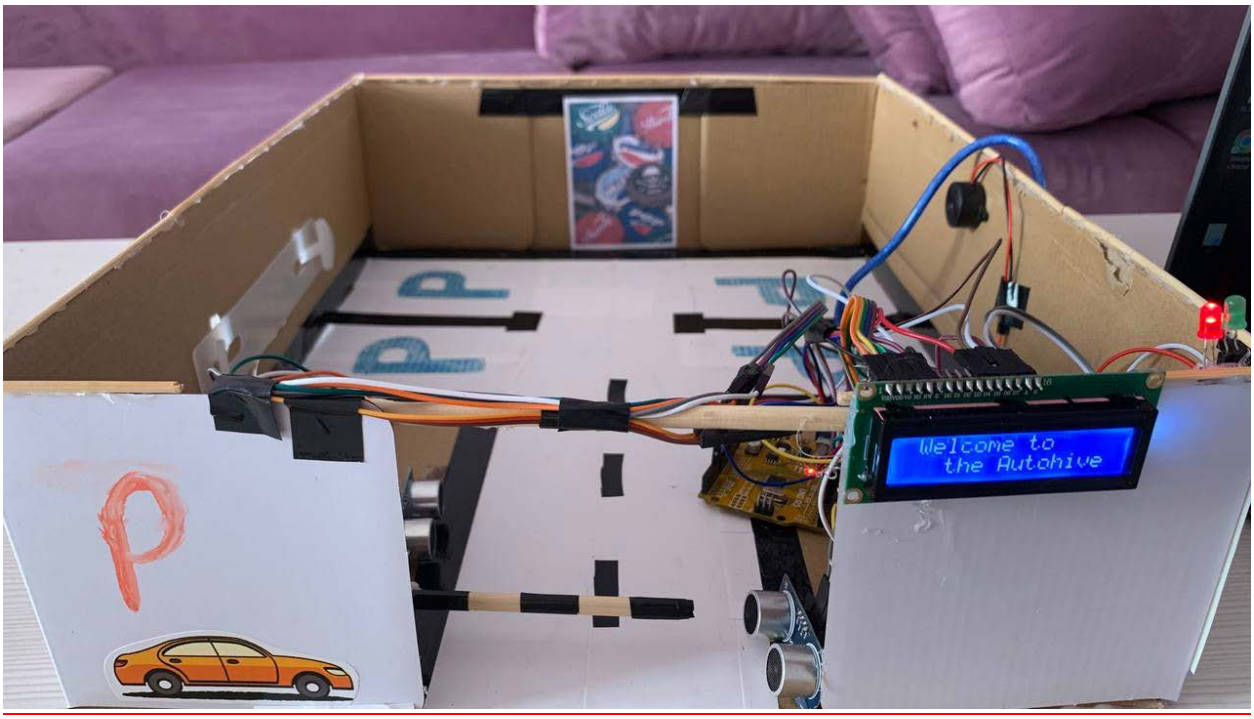
You can download **LiquidCrystal.h** library from
https://www.arduino.cc/reference/en/libraries/liquidcrystal/

Download Servo.h library from

https://www.arduino.cc/reference/en/libraries/servo/

See and use the source code in Github

The link of GitHub - https://github.com/mey17/theAutohive



Thank you for your time!