**Chapter-1: Introduction**

**1.1 Problem Statement**

The exponential growth in number of cars has caused traffic bottleneck due to visitors searching for a parking space in metropolitan cities during the peak time. There is great analogy that when a driver enters any of the parking lot, he must look for some kind of information board that tells him about the status of the parking lot that whether it is fully occupied, partly occupied or vacant. Most of the times the drivers must circle around the parking area in search of the free parking space. This kind of problem mostly occur in cities near the shopping malls, hospitals etc., where the number of vehicles is greater as compared to the parking spaces. The process for searching the free parking space is time consuming and wastage of fuel. Most of the times the parking spaces remain unoccupied, however the total occupancy is low because of bad management of parking lot. This causes ineffective use of the parking area and results in traffic jams and congestion near the parking lots.

**1.2 Abstract**

In recent years, there has been a significant increase in the number of vehicles on the road, leading to a shortage of parking spaces in urban areas. This has led to several problems, including traffic congestion, pollution, and driver frustration.

To address these problems, there has been a growing interest in smart parking systems. Smart parking systems use technology to help drivers find available parking spaces more quickly and easily. They can also help to reduce traffic congestion and pollution by encouraging drivers to park in less congested areas.

This project proposes a smart parking system using Arduino Uno and ultrasonic sensors, servo motor and LCD display. This system will work as follows:

* Ultrasonic sensors will be placed at each parking spaces to detect the presence of a vehicle.
* The servo motor will be used to open and close the parking lot gate.
* The LCD display will be used to display the number of available parking spaces in the lot.

When a vehicle enters the parking lot, the ultrasonic sensors will detect its presence and send a signal to the Arduino Uno. The Arduino Uno will then open the parking lot gate and display the number of available parking spaces on the LCD display.

The driver can then park in any available space. When the driver exits the parking lot, the ultrasonic sensors will detect the absence of the vehicle and send a signal to the Arduino Uno. The Arduino will then open the parking lot gate and update the number of available parking spaces on the LCD display.

The threshold distance for the ultrasonic sensor to detect a vehicle is 5cm. this means that if an object is closer than 5cm to the sensor, the sensor will output a high signal. If the object is further away than 5cm, the sensor will output a low signal.

This smart parking system offers several benefits, including:

* Reduced traffic congestion
* Reduced pollution
* Increased driver satisfaction
* Improved efficiency of parking lots

The Arduino microcontroller coordinates these functionalities, ensuring seamless integration and effective automation. Users can customize critical parameters such as door open and close distances, slot occupancy thresholds, and servo motor angles based on their specific requirements. The project is designed to enhance parking management efficiency, reduce manual interventions, and provide a user-friendly experience for both door access and parking space availability.

**1.3 Objective**

* to identify the number of available parking space in parking lot.
* To display the number of empty parking space.
* To display the exact location of the empty parking space.
* To display the location of the nearest empty parking space.

**1.4 Scope and Future Prospects**

The Smart parking project, as implemented in the automated door and parking system, demonstrates a promising solution for addressing parking challenges in urban environments. The scope of the project extends beyond its current implementation, opening avenues for innovation and integration into larger smart city initiatives.

1. **Urban Mobility Enhancement:**

* Smart Parking can significantly contribute to the optimization of urban mobility by reducing traffic congestion and improving overall traffic flow.
* Integration with city-wide systems can provide real-time data on parking availability, guiding drivers to available spaces efficiently.

1. **Environmental Impact:**

* By minimizing the time spent searching for parking spaces, the project supports environmental sustainability by reducing vehicle emission and fuel consumption.
* Expansion of the system to include environmental sensors could further monitor air quality and contribute to a healthier urban environment.

1. **IoT Integration:**

* Integration with the Internet of Things(IoT) can enhance the project’s capabilities. For example, leveraging IoT platforms for data analytics could provide valuable insight into parking patterns and demand.
* Implementation of smart sensors and communication technologies can enable dynamic pricing models for parking, optimizing usage and revenue generation.

1. **Mobile Applications:**

* The development of dedicated mobile application can provide users with real-time information about parking availability, navigation to open spaces, and remote control over the automated door system.
* In-app features, such as reservation of parking and payment integration, can enhance user convenience.

1. **Smart City Integration:**

* Integration of the Smart Parking system into broader smart city initiatives can create synergies with other systems, such as traffic management, public transportation, and emergency services.
* Collaboration with municipal authorities can lead to policy recommendations and urban planning improvements based on data collected from the Smart Parking infrastructure.

1. **Advanced Security Measures:**

* The automated door systems can be further developed to include advanced security measures, such as facial recognition for authorized access.
* Integration with surveillance systems and alarms can enhance the overall security of parking areas.

1. **Data Analytics and Predictive Maintenance:**

* Implementing data analytics can provide insights into parking usage tends, helping authorities plan and optimize parking infrastructure.
* Predictive maintenance algorithms can be applied to the automated door and sensor systems to prevent failures and ensure continuous operation.

1. **Community Engagement:**

* Inclusion of community engagement features, such as user feedback mechanisms, social media integration, and community forums, can foster a sense of collaboration and participation among users.

In conclusion, the Smart Parking project’s scope extends far beyond its current implementation, presenting opportunities for scalability, integration, and innovation. The ongoing development and expansion of the system can contribute to the creation of more sustainable, efficient, and user-friendly urban environments.

**Chapter 2: Literate Review**

**2.1 Overview of Ultrasonic Sensors**

**2.1.1 Introduction**

Ultrasonic sensors have emerged as versatile devices in the realm of distance measurement and object detection, finding applications in various fields such as robotics, automotive, healthcare, and home automation. This section provides a comprehensive overview of ultrasonic sensor technology, delving into its principles of operation, characteristics, and common use cases.

**2.1.2 Principles of Operation**

Ultrasonic sensors operate on the principle of echolocation, akin to the sonar navigation used by bats and dolphins. These sensors emit ultrasonic waves, typically in the ultrasonic frequency range (20 kHz to 200 kHz), and measure the time taken for the waves to reflect off an object and return to the sensor. By calculating the time of flight, the sensor determines the distance to the object with high precision.

**2.1.3 Characteristics of Ultrasonic Sensors**

1. **Non-Contact Measurement:** One of the primary advantages of ultrasonic sensors is their non-contact nature, allowing for distance measurement without physical interaction with the object.
2. **Versatility:** Ultrasonic sensors are suitable for a wide range of materials, including liquids, solids, and powders, making them adaptable to diverse applications. Reliability: They exhibit robust performance in various environmental conditions, including darkness, dust, and moisture.
3. **Accuracy:** Ultrasonic sensors offer high accuracy in distance measurement, with resolutions that can be fine-tuned for specific requirements.

**2.1.4 Applications**

1. **Object Detection:** Ultrasonic sensors are extensively used for detecting the presence or absence of objects in automation and robotics.
2. **Distance Measurement:** Their precise distance measurement capabilities make them ideal for applications such as parking assistance in automotive systems. Liquid Level
3. **Measurement:** In industrial contexts, ultrasonic sensors are employed for monitoring and controlling liquid levels in tanks.
4. **Medical Imaging:** Ultrasonic sensors play a crucial role in medical imaging techniques like ultrasonography, facilitating non-invasive diagnostics.

**2.1.5 Recent Advancements**

Recent advancements in ultrasonic sensor technology involve miniaturization, improved signal processing algorithms, and integration with other sensing technologies. These developments have led to the creation of more compact, efficient, and intelligent ultrasonic sensors with enhanced capabilities**.**

**2.1.6 Challenges and Future Directions**

Despite their widespread use, challenges such as sensitivity to environmental factors and limited resolution in certain applications persist. The literature review will explore recent research efforts addressing these challenges and propose potential avenues for future improvements in ultrasonic sensor technology.

**2.2 Servo Motors in Automation**

**2.2.1 Introduction to Servo Motors**

Servo motors are critical components in automation systems, providing precise control over angular position, speed, and torque. These motors play a pivotal role in various applications, including robotics, manufacturing, and mechatronics.

**2.2.2 Characteristics of Servo Motors**

Servo motors exhibit unique characteristics that make them well-suited for automation tasks: Precision: Servo motors offer high precision and accuracy in positioning, allowing for fine control over movements.

* **Feedback Systems:** Many servo motors incorporate feedback systems, such as encoders, to provide real-time information about the motor's position and speed.
* **High Torque:** Servo motors can deliver high torque at low speeds, making them suitable for applications requiring both strength and precision.
* **Speed Control:** The ability to control the speed of servo motors ensures dynamic and adaptable performance.

**2.2.3 Applications of Servo Motors in Automation**

Servo motors find extensive use in various automation applications, including:

* **Robotics:** Servo motors drive robotic joints, enabling precise and coordinated movements.
* **CNC Machines:** In Computer Numerical Control (CNC) machines, servo motors control the movement of cutting tools with high accuracy.
* **Automated Doors:** Servo motors are employed in automated door systems for controlled opening and closing mechanisms.

**2.2.4 Research and Innovations in Servo Motor Technology**

To date, research in servo motor technology has focused on:

* **Enhanced Feedback Systems:** Studies explore advanced feedback mechanisms to further improve positional accuracy.
* **Energy Efficiency:** Research aims to develop energy-efficient servo motors to reduce power consumption.
* **Miniaturization:** Ongoing efforts focus on miniaturizing servo motors for applications with space constraints.
* **Integration with IoT:** Exploring how servo motors can be integrated into IoT systems for enhanced automation capabilities.

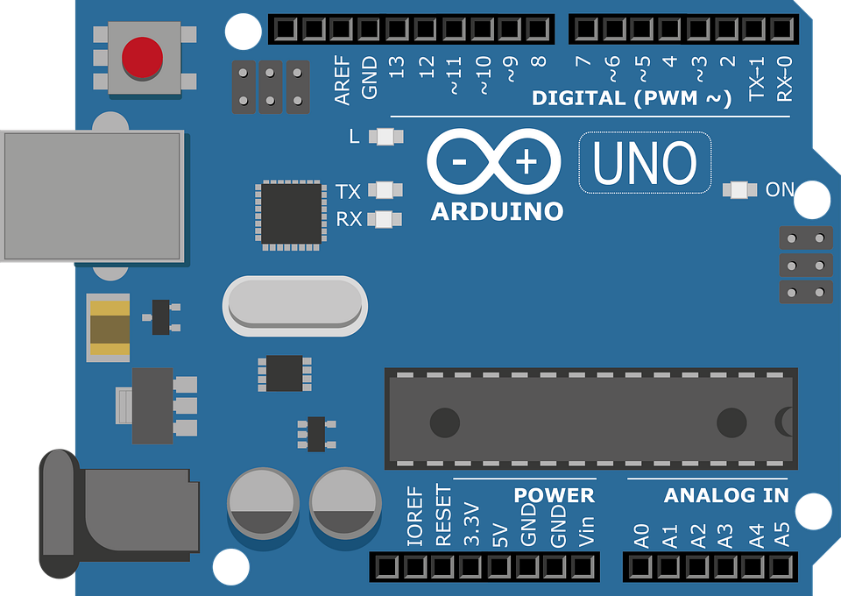
**2.2.5 Challenges and Future Directions**

While servo motors offer significant advantages, challenges persist, including:

* **Cost:** High-quality servo motors can be expensive, impacting the overall cost of automation systems.
* **Maintenance:** Regular maintenance is essential for ensuring the longevity and reliability of servo motors.
* **Integration Complexity:** Integrating servo motors into complex automation systems may pose challenges.

**2.3 Overview of Arduino Uno**

The Arduino Uno is a widely used open-source microcontroller board that has played a pivotal role in revolutionizing the landscape of electronics and embedded systems development.

Figure 2.1: Arduino Uno Board

Developed by the Arduino company, the Uno is part of the Arduino family of boards, each designed to cater to different needs within the maker and prototyping community.

**2.3.1 Design and Architecture**

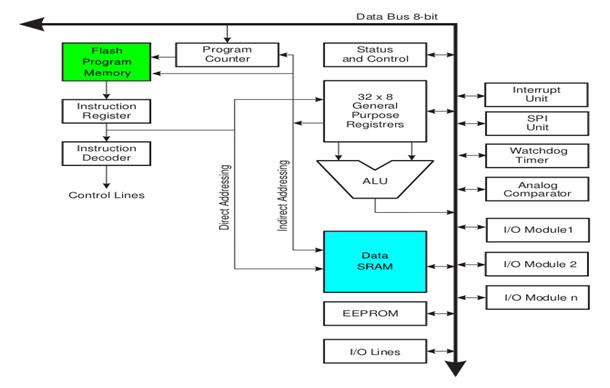
The Arduino Uno is based on the ATmega328P microcontroller, featuring a straightforward design that makes it accessible for both beginners and experienced developers. Figure 2.1 illustrates the basic architecture of the Arduino Uno.

Figure 2.2: Arduino Uno Architecture

The board incorporates a set of digital and analog input/output pins, allowing users to interface with a variety of sensors, actuators, and peripheral devices. The simplicity of its design, coupled with a user-friendly integrated development environment (IDE), has contributed to its popularity for a wide range of applications.

**2.3.2 Programming and IDE**

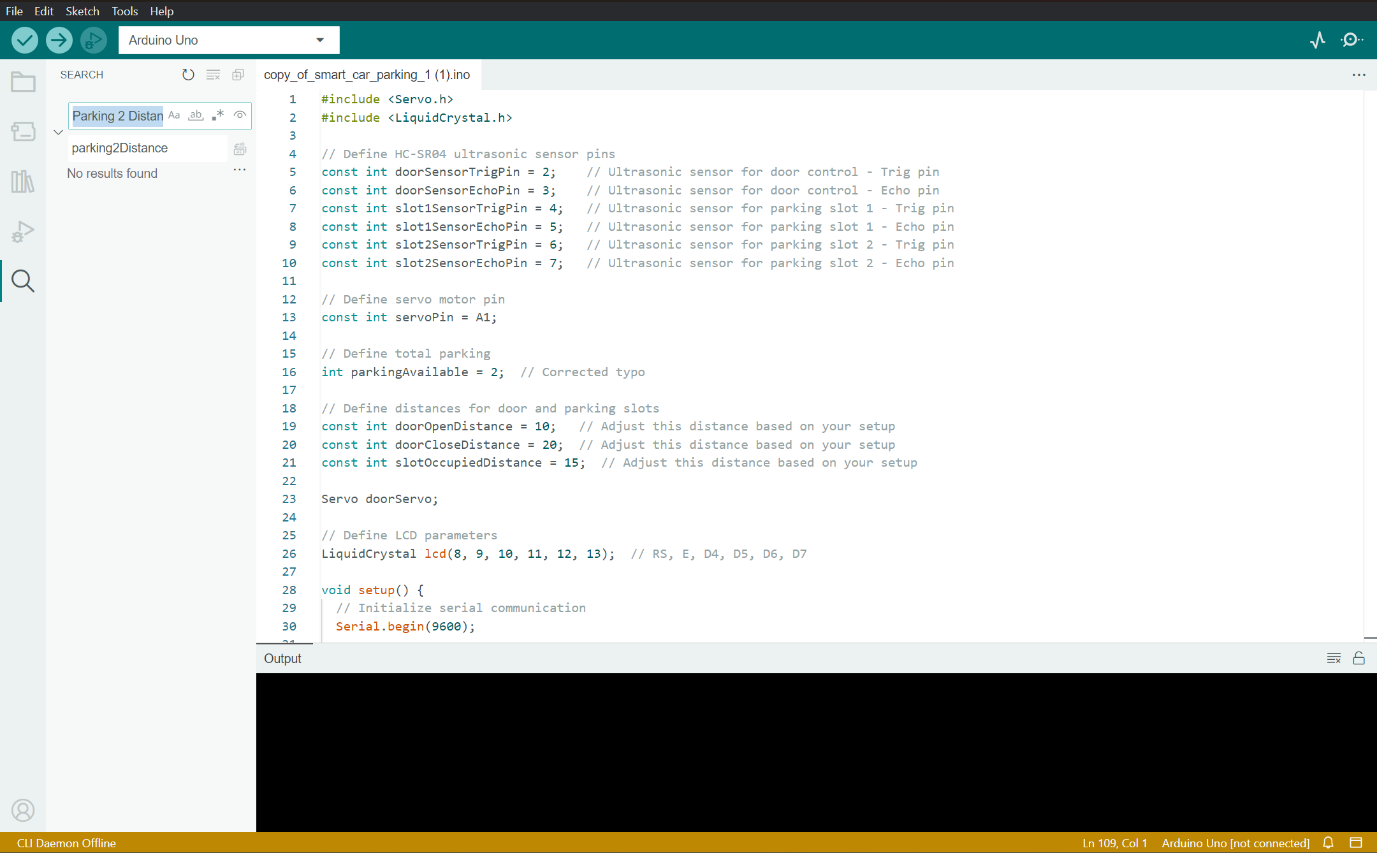
One of the defining features of the Arduino Uno is its ease of programming. The Arduino IDE provides a simplified environment for writing, compiling, and uploading code to the board. Figure 2.2 illustrates the Arduino IDE in action.

Figure2.3: Arduino IDE

The use of the C and C++ programming languages, combined with a vast library of pre-built functions, simplifies the coding process. This accessibility has made the Arduino Uno an ideal platform for educational purposes, enabling students and hobbyists to quickly grasp the fundamentals of embedded systems.

**2.3.3 Community and Open Source Ecosystem**

Arduino's strength lies not only in its hardware but also in the robust community that surrounds it. The open-source nature of Arduino has fostered a global community of enthusiasts, sharing projects, code snippets, and troubleshooting tips. This collaborative spirit has significantly contributed to the versatility and innovation associated with Arduino-based projects.

**2.3.4 Applications and Impact**

The Arduino Uno's impact extends beyond the realm of hobbyist projects, finding applications in diverse fields such as home automation, robotics, and prototyping for commercial products. Its versatility, cost-effectiveness, and simplicity make it an attractive choice for rapid prototyping and proof-of-concept development.

**2.3.5 Related Work**

Several projects and research efforts have utilized the Arduino Uno as a key component in their designs. These projects span various domains, including home automation, environmental monitoring, and educational tools. A comprehensive review of related work incorporating Arduino Uno will be presented in Section 2.3.

In summary, the Arduino Uno serves as a foundational element in the Smart Parking project, providing a reliable and accessible platform for hardware development and programming. Its widespread adoption and integration into countless projects underscore its significance in the realm of embedded systems and DIY electronics.

**Chapter 3: System Architecture**

**3.1 Hardware Components**

**3.1.1 Arduino Uno**

The Arduino UNO is a standard board of Arduino. Here UNO means 'one' in Italian. It was named as UNO to label the first release of Arduino Software. It was also the first USB board released by Arduino. It is considered as the powerful board used in various projects. Arduino.cc developed the Arduino UNO board.

Figure 3.1: Arduino Uno Board

Arduino UNO is based on an **ATmega328P** [microcontroller](https://www.javatpoint.com/microcontroller). It is easy to use compared to other boards, such as the Arduino Mega board, etc. The board consists of digital and analog Input/Output pins (I/O), shields, and other circuits.

The Arduino UNO includes 6 analog pin inputs, 14 digital pins, a [USB](https://www.javatpoint.com/usb-full-form) connector, a power jack, and an ICSP (In-Circuit Serial Programming) header. It is programmed based on IDE, which stands for Integrated Development Environment. It can run on both online and offline platforms.

The technical specifications of the Arduino UNO are listed below:

* There are 20 Input/Output pins present on the Arduino UNO board. These 20 pins include 6 PWM pins, 6 analog pins, and 8 digital I/O pins.
* The PWM pins are Pulse Width Modulation capable pins.
* The crystal oscillator present in Arduino UNO comes with a frequency of 16MHz.
* It also has a Arduino integrated Wi-Fi module. Such Arduino UNO board is based on the Integrated Wi-Fi ESP8266 Module and ATmega328P microcontroller.
* The input voltage of the UNO board varies from 7V to 20V.
* Arduino UNO automatically draws power from the external power supply. It can also draw power from the USB.

Some commonly used terms while addressing a Arduino Uno are listed below:

* **ATmega328 Microcontroller:** It is a single chip Microcontroller of the ATmel family. The processor code inside it is of 8-bit. It combines **Memory (SRAM, EEPROM, and Flash), Analog to Digital Converter, SPI serial ports, I/O lines, registers, timer, external and internal interrupts, and oscillator.**
* **ICSP pin:** The In-Circuit Serial Programming pin allows the user to program using the firmware of the Arduino board.
* **Power LED Indicator:** The ON status of LED shows the power is activated. When the power is OFF, the LED will not light up.
* **Digital I/O pins**: The digital pins have the value HIGH or LOW. The pins numbered from D0 to D13 are digital pins.
* **TX and RX LED's**: The successful flow of data is represented by the lighting of these LED's.
* **AREF:**The Analog Reference (AREF) pin is used to feed a reference voltage to the Arduino UNO board from the external power supply.
* **Reset button**: It is used to disconnecting and reconnecting the power supply**.**
* **USB**: It allows the board to connect to the computer. It is essential for the programming of the Arduino UNO board.
* **Crystal Oscillator**: The Crystal oscillator has a frequency of 16MHz, which makes the Arduino UNO a powerful board.
* **Voltage Regulator**: The voltage regulator converts the input voltage to 5V.
* **GND**: Ground pins. The ground pin acts as a pin with zero voltage.
* **Vin**: It is the input voltage.
* **Analog Pins**: The pins numbered from A0 to A5 are analog pins. The function of Analog pins is to read the analog sensor used in the connection. It can also act as GPIO (General Purpose Input Output) pins.

**3.1.2 HC-SR04 Ultrasonic Sensor (3 units)**

An HC-SR04 ultrasonic distance sensor actually consists of two [ultrasonic transducers](https://en.wikipedia.org/wiki/Ultrasonic_transducer). One acts as a transmitter that converts the electrical signal into 40 KHz ultrasonic sound pulses. The other acts as a receiver and listens for the transmitted pulses. When the receiver receives these pulses, it produces an output pulse whose width is proportional to the distance of the object in front. This sensor provides excellent non-contact range detection between 2 cm to 400 cm (~13 feet) with an accuracy of 3 mm.

Since it operates on 5 volts, it can be connected directly to an Arduino or any other 5V logic microcontroller.

The technical specification of HC-SR04 Ultrasonic Sensors are given below:

|  |  |
| --- | --- |
| **Operating Voltage** | DC 5V |
| **Operating Current** | 15mA |
| **Operating Frequency** | 40KHz |
| **Max. Range** | 4m |
| **Min. Range** | 2cm |
| **Ranging Accuracy** | 3mm |
| **Measuring Angles** | 15° |
| **Trigger Input Signal** | 10µS TTL pulse |

**3.1.2.1 HC-SR04 Ultrasonic Sensor Pinout**

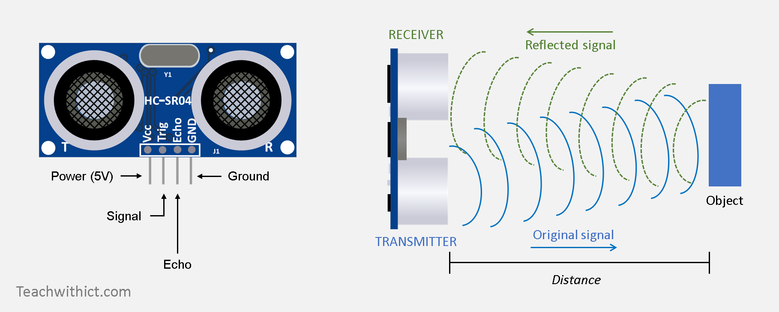
Let’s take a look at its pinout.

Figure 3.2: HC-SR04 Ultrasonic Sensor

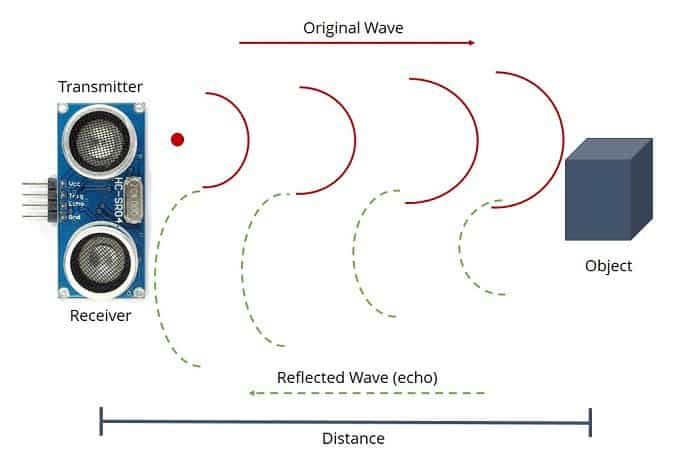
* **Vcc:** Supplies power to the HC-SR04 ultrasonic sensor. You can connect it to the 5V output from the Arduino.
* **Trig (Trigger):** Trigger pin is used to trigger ultrasonic sound pulses. By setting this pin to HIGH for 10µs, the sensor initiates an ultrasonic burst.
* **Echo:** This pin hoes HIGH when the ultrasonic burst is transmitted and remains high until the sensor receives an echo, after which it goes low. By measuring the time the Echo pin stays high, the distance can be calculated.
* **GND:** This is the ground pin. Connect it to the ground of the Arduino.

**3.2.2 Working of HC-SR04 Ultrasonic Distance Sensor**

It all starts when the trigger pin is set HIGH for 10µs. In response, the sensor transmits an ultrasonic burst of eight pulses at 40 kHz. This 8-pulse pattern is specially designed so that the receiver can distinguish the transmitted pulses from ambient ultrasonic noise.

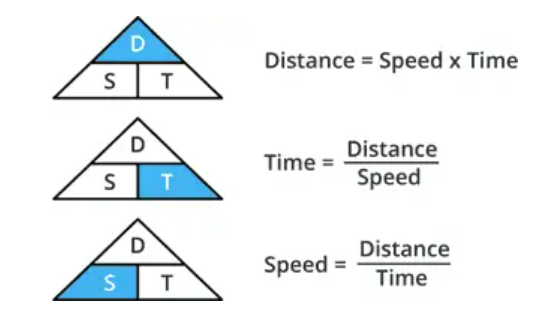
These eight ultrasonic pulses travel through the air away from the transmitter. Meanwhile the echo pin goes HIGH to initiate the echo-back signal.

If those pulses are not reflected back, the echo signal times out and goes low after 38ms (38 milliseconds). Thus, a pulse of 38ms indicates no obstruction within the range of the sensor.

Figure 3.3: Ultrasonic Sensor Transmitting and Receiving Signal

If those pulses are reflected back, the echo pin goes low as soon as the signal is received. This generates a pulse on the echo pin whose width varies from 150 µs to 25 ms depending on the time taken to receive the signal.

**3.1.2.3 Calculating the Distance**

The width of the received pulse is used to calculate the distance from the reflected object. This can be worked out using the simple distance-speed-time equation we learned in high school. An easy way to remember the equation is to put the letters in a triangle.

Let us take an example to make it more clear. Suppose we have an object in front of the sensor at an unknown distance and we receive a pulse of 500µs width on the echo pin. Figure 3.4: Distance measurement

Now let’s calculate how far the object is from the sensor. For this we will use the below equation.

Distance = Speed x Time

Here we have the value of time i.e. 500 µs and we know the speed. Of course, it’s the speed of sound! It is 340 m/s. To calculate the distance, we need to convert the speed of sound into cm/µs. It is 0.034 cm/μs. With that information we can now calculate the distance!

Distance = 0.034 cm/µs x 500 µs

But we’re not done yet! Remember that the echo pulse indicates the time it takes for the signal to be sent and reflected back. So, to get the distance, you have to divide your result by two.

Distance = (0.034 cm/µs x 500 µs) / 2

Distance = 8.5 cm

Now we know that the object is 8.5 cm away from the sensor.

**3.1.3 Servo Motor (SG90)**

Servo motors are part of a closed-loop system and are comprised of several parts namely a control circuit, servo motor, shaft, potentiometer, drive gears, amplifier and either an encoder or resolver. A servo motor is a self-contained electrical device, that rotate parts of a machine with high efficiency and with great precision. The output shaft of this motor can be moved to a particular angle, position and velocity that a regular motor does not have. The Servo Motor utilizes a regular motor and couples it with a sensor for positional feedback. The controller is the most important part of the Servo Motor designed and used specifically for this purpose.

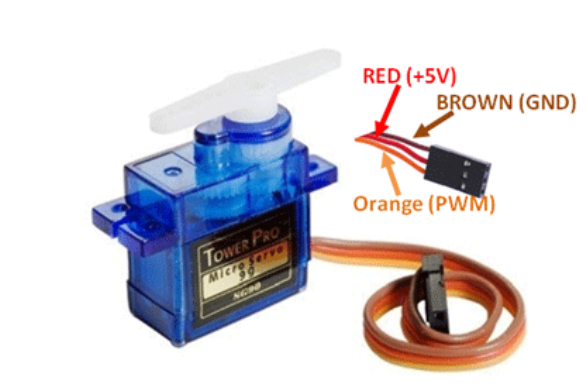
****The servo motor is a closed-loop mechanism that incorporates positional feedback to control the rotational or linear speed and position. The motor is controlled with an electric signal, either analog or digital, which determines the amount of movement which represents the final command position for the shaft. A type of encoder serves as a sensor providing speed and position feedback. This circuitry is built right inside the motor housing which usually is fitted with gear system.

Figure 3.5: Servo Motor

|  |  |
| --- | --- |
| **Wire**  **Colour** | **Description** |
| Brown | Ground wire connected to the ground of system |
| Red | Powers the motor typically +5V |
| Orange | PWM signal is given in through this wire to drive motor |

### **3.1.3.1 Servo Motor (SG-90) Features**

* Operating Voltage is +5V
* Torque is 2.5kg/cm
* Operating Speed is 0.1s/60°
* Gear Type: Plastic
* Rotation: 0°-180°
* Weight of motor: 9gm

**3.1.4 Liquid Crystal Display (LCD)**

The term [LCD stands for liquid crystal display](https://www.elprocus.com/difference-alphanumeric-display-and-customized-lcd/). It is one kind of electronic display module used in an extensive range of applications like various circuits & devices like mobile phones, calculators, computers, TV sets, etc. These displays are mainly preferred for multi-segment [light-emitting diodes](https://www.elprocus.com/light-emitting-diode-led-working-application/) and seven segments. The main benefits of using this module are inexpensive; 16X2 LCD simply programmable, 16X2 LCD animations, and there are no limitations for displaying custom characters, special and even animations, etc.

Figure 3.6: 16x2 LCD Display

### **3.1.4.1 LCD 16×2 Pin Diagram**

The 16×2 LCD pinout is shown below.

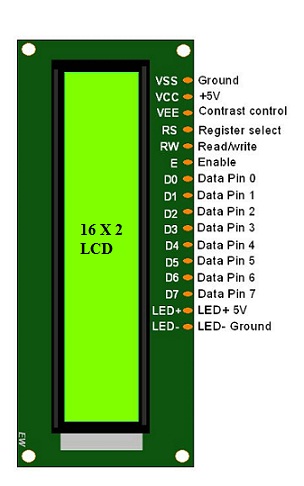
* **Pin1 (Ground/Source Pin):** This is a GND pin of display, used to connect the GND terminal of the microcontroller unit or power source.
* **Pin2 (VCC/Source Pin):** This is the voltage supply pin of the display, used to connect the supply pin of the power source.
* **Pin3 (V0/VEE/Control Pin):** This pin regulates the difference of the display, used to connect a changeable POT that can supply 0 to 5V.

Figure 3.7: Pin diagram

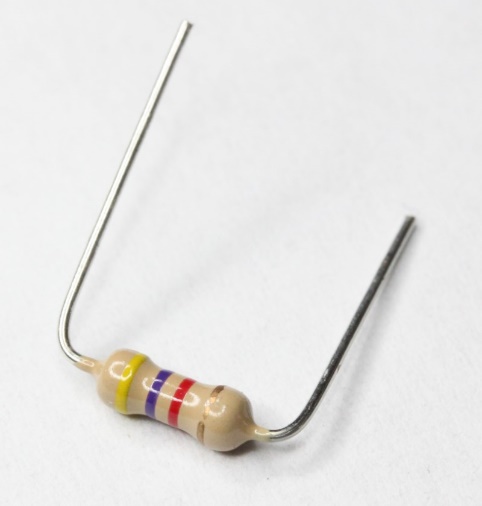
* **Pin4 (Register Select/Control Pin):** This pin toggles among command or data register, used to connect a microcontroller unit pin and obtains either 0 or 1(0 = data mode, and 1 = command mode).
* **Pin5 (Read/Write/Control Pin):** This pin toggles the display among the read or writes operation, and it is connected to a microcontroller unit pin to get either 0 or 1 (0 = Write Operation, and 1 = Read Operation).
* **Pin 6 (Enable/Control Pin)**: This pin should be held high to execute Read/Write process, and it is connected to the microcontroller unit & constantly held high.
* **Pins 7-14 (Data Pins):** These pins are used to send data to the display. These pins are connected in two-wire modes like 4-wire mode and 8-wire mode. In 4-wire mode, only four pins are connected to the microcontroller unit like 0 to 3, whereas in 8-wire mode, 8-pins are connected to microcontroller unit like 0 to 7.
* **Pin15 (+ve pin of the LED):** This pin is connected to +5V
* **Pin 16 (-ve pin of the LED):** This pin is connected to GND.

### **3.1.4.2 Features of LCD 16x2**

The features of this LCD mainly include the following.

* The operating voltage of this LCD is 4.7V-5.3V
* It includes two rows where each row can produce 16-characters.
* The utilization of current is 1mA with no backlight
* Every character can be built with a 5×8-pixel box
* The alphanumeric LCDs alphabets & numbers
* Is display can work on two modes like 4-bit & 8-bit
* These are obtainable in Blue & Green Backlight
* It displays a few custom generated characters

**3.1.5 Resistor**

****A resistor is an electrical component that limits or regulates the flow of electrical [current](https://www.techtarget.com/whatis/definition/current) in an electronic circuit. Resistors can also be used to provide a specific voltage for an active device such as a [transistor](https://www.techtarget.com/whatis/definition/transistor).

All other factors being equal, in a direct-current ([DC](https://www.techtarget.com/whatis/definition/DC-direct-current)) circuit, the current through a resistor is inversely proportional to its [resistance](https://www.techtarget.com/whatis/definition/resistance), and directly proportional to the voltage across it. This is the well-known [Ohm's Law](https://www.techtarget.com/whatis/definition/Ohms-Law). In alternating-current ([AC](https://www.techtarget.com/whatis/definition/alternating-current-AC)) circuits, this rule also applies as long as the resistor does not contain inductance or [capacitance](https://www.techtarget.com/whatis/definition/capacitor-capacitance).

Figure 3.8: Resistor

## **3.1.5.1 Types of resistors**

Resistors can be fabricated in a variety of ways.

* The most common type in electronic devices and systems is the **carbon-composition resistor**. Fine granulated carbon (graphite) is mixed with clay and hardened. The resistance depends on the proportion of carbon to clay; the higher this ratio, the lower the resistance.
* Another type of resistor is made from winding Nichrome or similar wire on an insulating form. This component, called a **wire wound resistor**, is able to handle higher currents than a carbon-composition resistor of the same physical size. However, because the wire is wound into a coil, the component acts as an [inductor](https://www.techtarget.com/whatis/definition/inductor)s as well as exhibiting resistance. This does not affect performance in DC circuits, but can have an adverse effect in AC circuits because inductance renders the device sensitive to changes in [frequency](https://www.techtarget.com/whatis/definition/frequency).

**3.1.6 Breadboard**

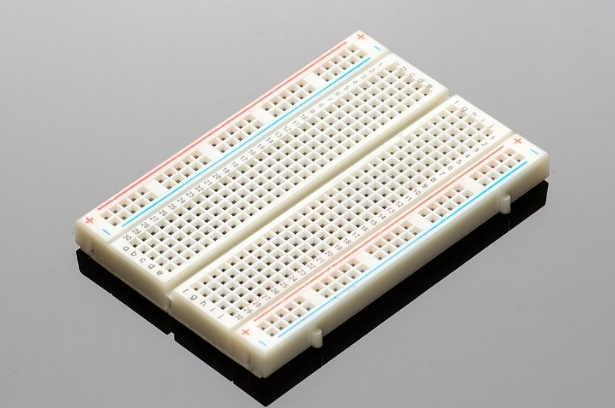
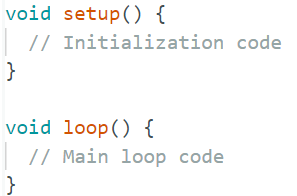
Breadboards are temporary work boards for [electronic circuits](https://www.sciencedirect.com/topics/engineering/networks-circuits). The general shape of a breadboard is shown in Fig. 6.3. Compatible with most breadboards, 24-gauge wire is used to connect circuits; solid wire, not stranded. Sometimes, kits may be available with various colours of fixed lengths to specifically fit breadboards. These are a nice convenience.

Figure 3.9: Breadboard

* 1. **Software Component**

The software component of the Smart Parking project plays a critical role in orchestrating the interaction between different hardware components, processing sensor data, and controlling the system's behaviour. The software is implemented as an Arduino sketch, which is a program written in the Arduino programming language based on C and C++. Let's delve into the key aspects of the software component:

**3.2.1 Arduino Sketch Structure**

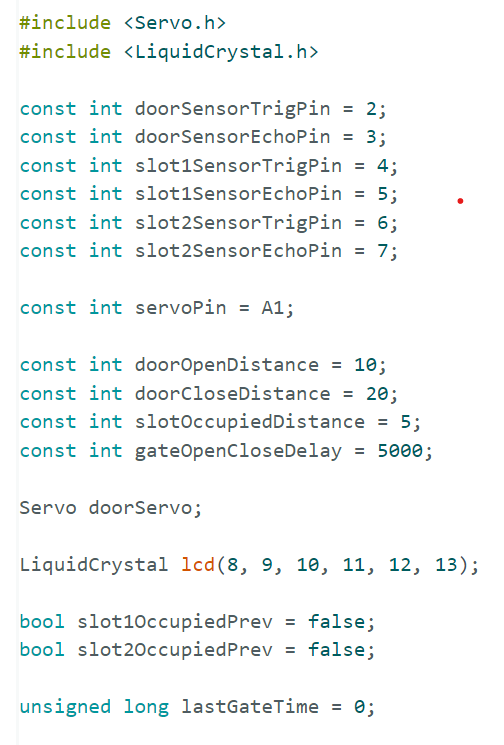
The Arduino sketch follows a structured format, including setup and loop functions. The setup function is executed once during the initialization phase, while the loop function runs continuously.

**3.2.2 Library Inclusion**

The software relies on specific libraries to extend the capabilities of the Arduino platform. In the Smart Parking project, two essential libraries are included:

* **Servo Library (<Servo.h>):** This library facilitates the control of servo motors, allowing precise positioning based on input angles.
* **Liquid Crystal Library (<LiquidCrystal.h>):** This library is used to interface with LCD displays, enabling the project to convey parking status information visually.

**3.2.3 Global Variable Declarations:**

Key variables are declared at the beginning of the sketch to store information such as pin assignments, distance thresholds, and the status of parking slots. These variables are crucial for maintaining the state of the system.

**3.2.4 Function Definitions**

Several functions are defined to encapsulate specific functionalities, promoting code modularity and readability. The essential functions in the Smart Parking project include:

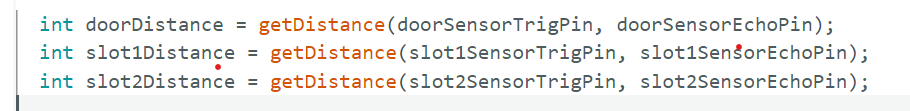
* **‘getDistance’:** Calculates the distance from an ultrasonic sensor based on the time taken for sound waves to travel.
* **‘openDoor’ and ‘closeDoor’:** Control the servo motor to open and close the door based on the distance from the door sensor.
* **‘updateLCD’:** Updates the LCD display with parking status information.
* **‘carLeavesSlot’:** This function checks if a car has left a parking slot. It returns true if the slot was occupied in the previous iteration but is now unoccupied based on the current distance measured by an ultrasonic sensor.
* **‘updateSlotStatus’:** This function updates the occupancy status of a parking slot based on the current distance measured by an ultrasonic sensor. If a car leaves the slot, and a certain delay has passed since the last gate action, it opens and closes the door.
* The **‘setup()’** function initializes the system, and the ‘**loop()’** function continuously reads sensor distances, controls the door, updates the LCD, and checks the status of each parking slot.

**3.2.5 Main Loop Logic**

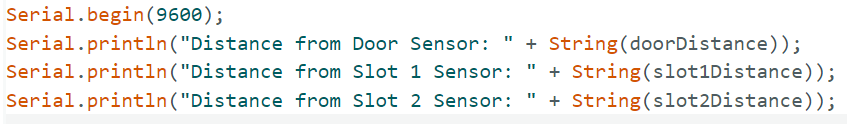
The main loop logic in this code is responsible for continuously reading the distances from ultrasonic sensors, controlling the door based on the door distance, updating the LCD with parking status, and updating the status of each parking slot.

**3.2.6 Distance Calculation Function (‘getDistance’)**

The getDistance function calculates the distance from an ultrasonic sensor based on the time it takes for the emitted pulse to return. This function is crucial for determining the proximity of obstacles and making decisions accordingly.



**3.2.7 Serial Communication for Debugging**

The Serial functions are employed for serial communication between the Arduino and the computer, facilitating debugging by printing relevant information to the Serial Monitor in the Arduino IDE.

**3.2.8 Delay to Avoid Rapid Sensor Readings**

A small delay is introduced in the main loop to avoid rapid and unnecessary sensor readings, promoting stability and efficiency in the system. …

**Chapter 4: Implementation**

The implementation of the Smart Parking system involves the integration of hardware components and the development of software logic to achieve automated door control and real-time parking status indication. The primary components include the Arduino Uno microcontroller, HC-SR04 ultrasonic sensors, a servo motor, and a Liquid Crystal display (LCD).

**4.1 Hardware Setup**

**4.1.1 Arduino Uno Integration**

* Connect the Arduino Uno to the computer and upload the Arduino sketch (program) using the Arduino IDE. Ensure that the necessary libraries, such as Servo and Liquid Crystal, are included in the sketch.

**4.1.2 Ultrasonic Sensors Configuration**

* Connect the Trig and Echo pins of each HC-SR04 ultrasonic sensor to their designated digital pins on the Arduino Uno. Use jumper wires for connections. Adjust the sensor placement to cover the door area and parking slots.

**4.1.3 Servo Motor Connection**

* Connect the servo motor to the designated pin on the Arduino Uno (defined as ‘servoPin’). Ensure proper power and ground connections. The servo motor controls the movement of the door.

**4.1.4 Liquid Crystal Display Integration**

* Connect the RS, E, D4, D5, D6, and D7 pins of the LCD to their respective digital pins on the Arduino Uno. Connect the backlight and adjust the contrast using the potentiometer.

**4.1.5 Power Supply**

* Power the Arduino Uno using an appropriate power source, such as a battery pack or an external power adapter. Ensure that the power supply can provide sufficient current for all connected components.

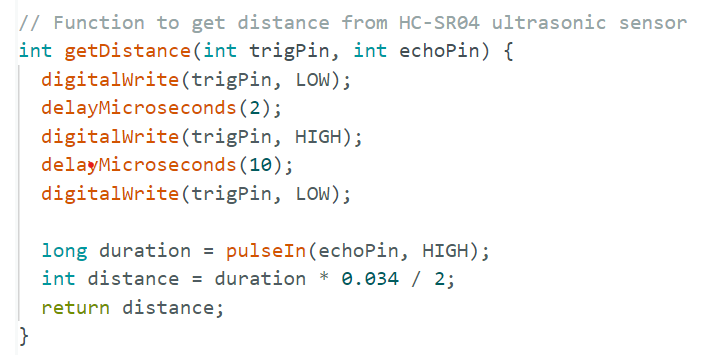
**4.1.6 Breadboard Setup**

* Use a breadboard to organize and interconnect the components, reducing the need for soldering. Ensure that all connections are secure and follow the circuit diagram provided in the project documentation.

**4.2 Software Implementation**

**4.2.1 Arduino Sketch**

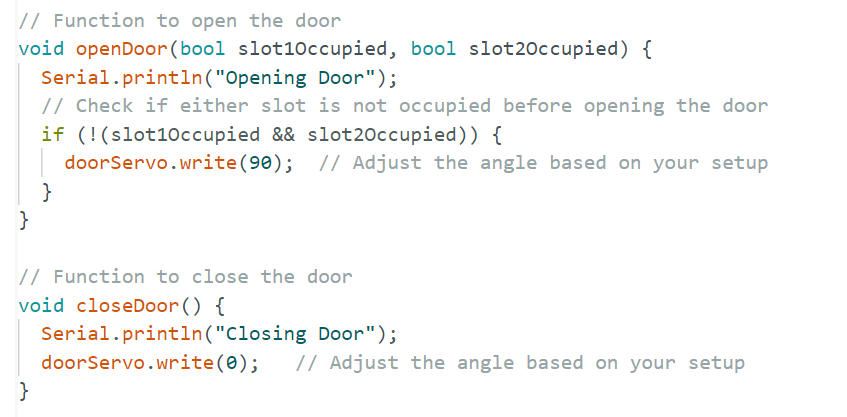
* Develop the Arduino sketch that includes functions for reading ultrasonic sensor data, controlling the servo motor, updating the LCD display, and implementing the logic for door control and parking status indication.

**4.2.2 Ultrasonic Sensor Logic**

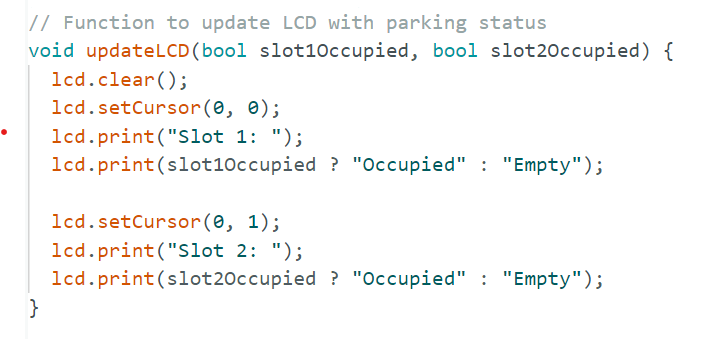
This function **‘getDistance’** is responsible for obtaining the distance measured by the ultrasonic sensor. Here's a step-by-step explanation:

1. **Trigger Pulse:** Set the **‘trigPin’** (trigger pin) LOW for 2 microseconds, then HIGH for 10 microseconds, and finally LOW again. This sequence generates a pulse to trigger the ultrasonic sensor.
2. **Measure Echo:** Use the **‘pulseIn’** function to measure the duration of the pulse on the echoPin (echo pin). This duration corresponds to the time taken for the ultrasonic wave to travel to the object and back.
3. **Calculate Distance:** Convert the pulse duration (**‘duration’**) to distance in centimetres using the formula **‘distance = duration \* 0.034 / 2’**. The factor of 0.034 is used to convert the time into centimetres (speed of sound in air is approximately 343 meters per second).

**4.2.3 Door Control Functions**

* The **‘openDoor’** function is responsible for opening the door. It checks if either parking slot is not occupied before opening the door. If at least one of the slots is empty, it commands the servo motor (connected to pin **A1**) to rotate to an angle of 90 degrees. You may need to adjust the angle based on your setup to ensure the door opens to the desired extent.
* The **‘closeDoor’** function is responsible for closing the door. It commands the servo motor to rotate back to an angle of 0 degrees, closing the door. Again, you may need to adjust the angle based on your setup.

**4.2.4 LCD Update Function**

This function uses the **‘LiquidCrystal’** library to control an LCD display. It clears the LCD, sets the cursor position, and then prints the status of each parking slot based on whether it is occupied or empty. The LCD display has two lines, and each line is used to display the status of one parking slot. The status is determined by the **‘slot1Occupied’** and **‘slot2Occupied’** parameters passed to the function. If a slot is occupied, it prints "Occupied"; otherwise, it prints "Empty".

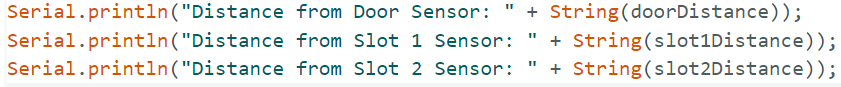
**4.2.5 Main Loop**

 In the main loop, continuously read distances from ultrasonic sensors, control the door, update the LCD, and introduce a delay to avoid rapid sensor readings.

**4.3 Testing and Troubleshooting**

**4.3.1 Serial Monitoring**

Use the Serial Monitor in the Arduino IDE for debugging. Print relevant variables and sensor readings to observe the system's behaviour.



**4.3.2 Verify Door Movements**

Manually verify that the servo motor responds correctly to changes in the door distance. Ensure that the door opens and closes smoothly.

**4.3.3 Check LCD Output**

Confirm that the LCD accurately reflects the parking status based on the distances from the ultrasonic sensors.

**4.3.4 Optimize Distances**

Adjust the threshold distances for door control and parking status indication to suit the specific environment and setup.

**Chapter 5: Conclusion**

The Smart Parking project successfully implements an automated door and parking status indication system using an Arduino Uno microcontroller and various hardware components. The integration of HC-SR04 ultrasonic sensors, a servo motor, and a Liquid Crystal display facilitates an efficient solution for managing parking spaces. The Arduino sketch orchestrates the system's behaviour, processing sensor data, controlling the servo motor for door movement, and updating the LCD to reflect real-time parking status.

The project not only addresses the immediate challenge of automating door control and parking status indication but also opens avenues for broader applications. The scalable nature of the system allows for integration with smart city initiatives, contributing to enhanced urban mobility, reduced environmental impact, and improved community engagement. The collaborative and open-source nature of the Arduino platform, coupled with a robust community, positions the Smart Parking project as a versatile and accessible solution for addressing parking challenges in urban environments.

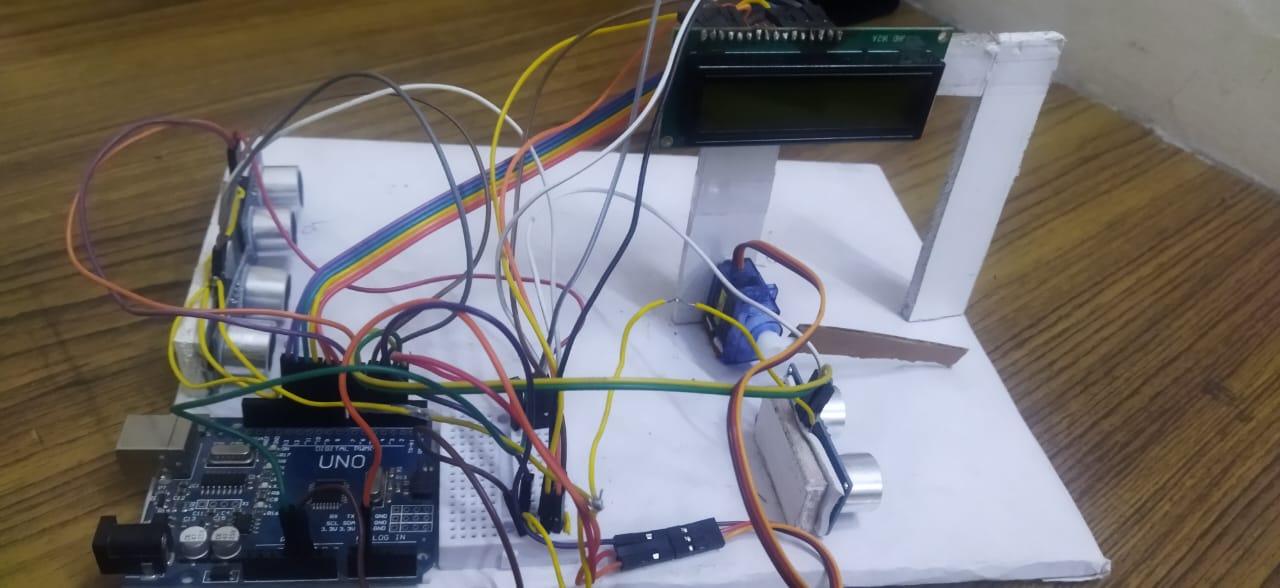
As technology continues to evolve, the Smart Parking system can benefit from advancements such as Internet of Things (IoT) integration, machine learning algorithms for predictive parking analytics, and enhanced security features. The successful implementation of this project serves as a foundation for future enhancements and innovations in the realm of smart parking solutions.

Fig 5.1: Finished project

**Chapter 6: References**

1. https://www.researchgate.net/publication/341870728\_Smart\_Parking\_System\_based\_on\_IOT
2. Arduino - Home. <https://www.arduino.cc/>
3. Servo Library. <https://www.arduino.cc/en/Reference/Servo>
4. Liquid Crystal Library. <https://www.arduino.cc/en/Reference/LiquidCrystal>
5. <https://www.tinkercad.com/things/3AnOiHO1LXu-copy-of-smart-parking/editel?returnTo=%2Fdashboard>
6. https://www.sensingthecity.com/how-to-make-an-arduino-toll-gate-with-ultrasonic-sensor-servo-and-lcd-screen/

**Chapter 7: Appendices**

**Appendix A:** **Arduino Sketch**

The Arduino sketch (program) for the Smart Parking project is provided in this appendix. It includes the complete code for the project, covering the setup, main loop, and functions for sensor readings, door control, and LCD updates.

#include <Servo.h>

#include <LiquidCrystal.h>

// Define HC-SR04 ultrasonic sensor pins

const int doorSensorTrigPin = 2; // Ultrasonic sensor for door control-Trig pin

const int doorSensorEchoPin = 3; // Ultrasonic sensor for door control-Echo pin

const int slot1SensorTrigPin = 4;// Ultrasonic sensor for parking slot 1-Trig pin

const int slot1SensorEchoPin = 5;// Ultrasonic sensor for parking slot 1-Echo pin

const int slot2SensorTrigPin = 6;// Ultrasonic sensor for parking slot 2-Trig pin

const int slot2SensorEchoPin = 7;// Ultrasonic sensor for parking slot 2-Echo pin

// Define servo motor pin

const int servoPin = A1;

// Define distances for door and parking slots

const int doorOpenDistance = 5;   // Adjust this distance based on your setup

const int doorCloseDistance = 5;  // Adjust this distance based on your setup

const int slotOccupiedDistance = 5;  // Adjust this distance based on your setup

// Define gate open and close delay

const int gateOpenCloseDelay = 5000; // 5 seconds

Servo doorServo;

// Define LCD parameters

LiquidCrystal lcd(8, 9, 10, 11, 12, 13);  // RS, E, D4, D5, D6, D7

bool slot1OccupiedPrev = false;

bool slot2OccupiedPrev = false;

unsigned long lastGateTime = 0; // Variable to store the last time the gate was opened or closed

// Function to get distance from HC-SR04 ultrasonic sensor

int getDistance(int trigPin, int echoPin) {

  digitalWrite(trigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);

  int distance = duration \* 0.034 / 2;

  return distance;

}

// Function to open the door

void openDoor(bool slot1Occupied, bool slot2Occupied) {

  Serial.println("Opening Door");

  // Check if either slot is not occupied before opening the door

  if (!(slot1Occupied && slot2Occupied)) {

    doorServo.write(90);  // Adjust the angle based on your setup

  }

}

// Function to close the door

void closeDoor() {

  Serial.println("Closing Door");

  doorServo.write(0);   // Adjust the angle based on your setup

}

// Function to update LCD with parking status

void updateLCD(bool slot1Occupied, bool slot2Occupied) {

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print("Slot 1: ");

  lcd.print(slot1Occupied ? "Occupied" : "Empty");

  lcd.setCursor(0, 1);

  lcd.print("Slot 2: ");

  lcd.print(slot2Occupied ? "Occupied" : "Empty");

}

// Function to check if a car leaves a slot

bool carLeavesSlot(bool previousOccupied, int currentDistance, int occupiedDistance) {

  return previousOccupied && !(currentDistance < occupiedDistance);

}

// Update the slot status and open the gate if necessary

void updateSlotStatus(bool &slotOccupiedPrev, int currentDistance, int occupiedDistance) {

  bool slotOccupied = currentDistance < occupiedDistance;

  if (carLeavesSlot(slotOccupiedPrev, currentDistance, occupiedDistance)) {

    unsigned long currentTime = millis();

    if (currentTime - lastGateTime >= gateOpenCloseDelay) {

      // Open the gate

      openDoor(false, false);

      delay(gateOpenCloseDelay);

      // Close the gate

      closeDoor();

      lastGateTime = currentTime;

    }

  }

  // Update previous occupied status

  slotOccupiedPrev = slotOccupied;

}

void setup() {

  // Initialize serial communication

  Serial.begin(9600);

  // Attach servo to its pin

  doorServo.attach(servoPin);

  // Set sensor pins as inputs and outputs

  pinMode(doorSensorTrigPin, OUTPUT);

  pinMode(doorSensorEchoPin, INPUT);

  pinMode(slot1SensorTrigPin, OUTPUT);

  pinMode(slot1SensorEchoPin, INPUT);

  pinMode(slot2SensorTrigPin, OUTPUT);

  pinMode(slot2SensorEchoPin, INPUT);

  // Initialize LCD

  lcd.begin(16, 2);

}

void loop() {

  // Read distances from ultrasonic sensors

  int doorDistance = getDistance(doorSensorTrigPin, doorSensorEchoPin);

  int slot1Distance = getDistance(slot1SensorTrigPin, slot1SensorEchoPin);

  int slot2Distance = getDistance(slot2SensorTrigPin, slot2SensorEchoPin);

  // Control the door based on the door distance

  if (doorDistance < doorOpenDistance) {

    unsigned long currentTime = millis();

    if (currentTime - lastGateTime >= gateOpenCloseDelay) {

      // Open the gate

      openDoor(slot1OccupiedPrev, slot2OccupiedPrev);

      delay(gateOpenCloseDelay);

      // Close the gate

      closeDoor();

      lastGateTime = currentTime;

    }

  }

  // Update LCD with parking status

  updateLCD(slot1OccupiedPrev, slot2OccupiedPrev);

  // Update slot 1 status and open gate if necessary

  updateSlotStatus(slot1OccupiedPrev, slot1Distance, slotOccupiedDistance);

  // Update slot 2 status and open gate if necessary

  updateSlotStatus(slot2OccupiedPrev, slot2Distance, slotOccupiedDistance);

  // Add a delay to avoid rapid sensor readings

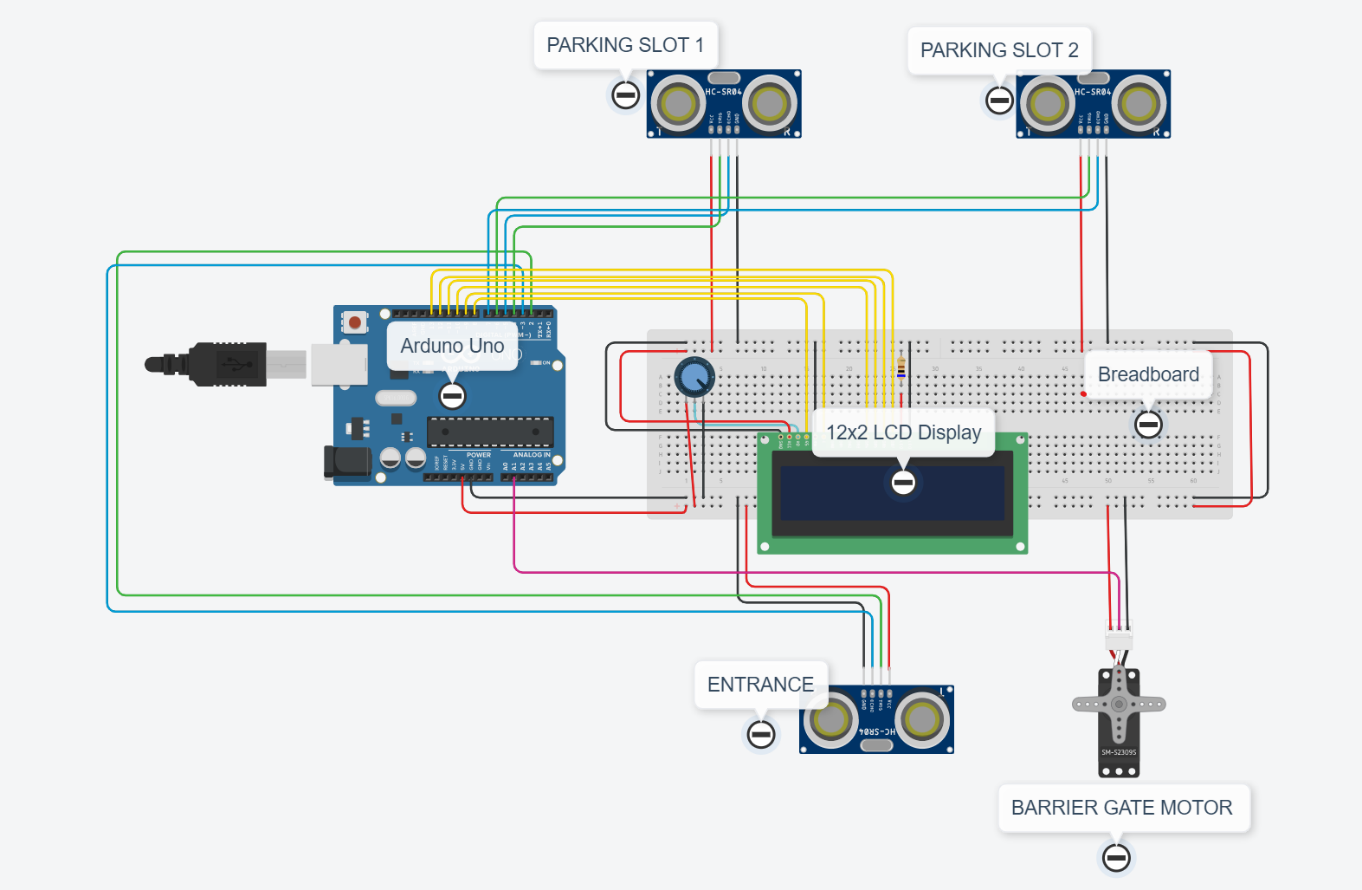
  delay(100);

}

**Appendix B: Circuit Diagram**

A detailed circuit diagram illustrating the connections between the Arduino Uno, ultrasonic sensors, servo motor, LCD, and other components is included in this section. The diagram provides a visual representation of the project's hardware configuration.

Figure 7.1: Circuit Diagram



Top of Form