

Parking Control Systems Using IoT Devices

ACIT4015-23V Internet of Things

Candidate Name: Akash Ray

Student ID: s371102

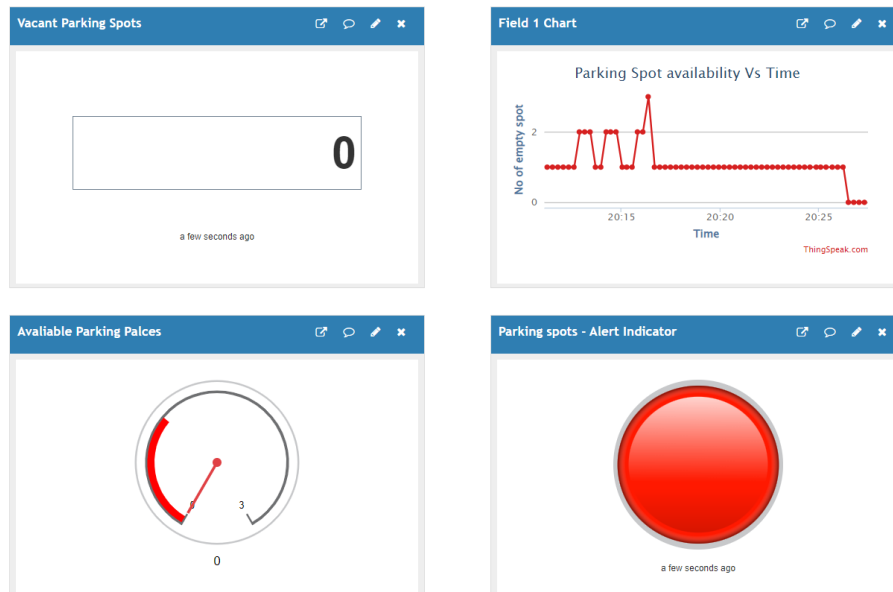


Figure 1 Available parking spot “zero” with RED lamp alert indicator for drivers at ThingSpeak website

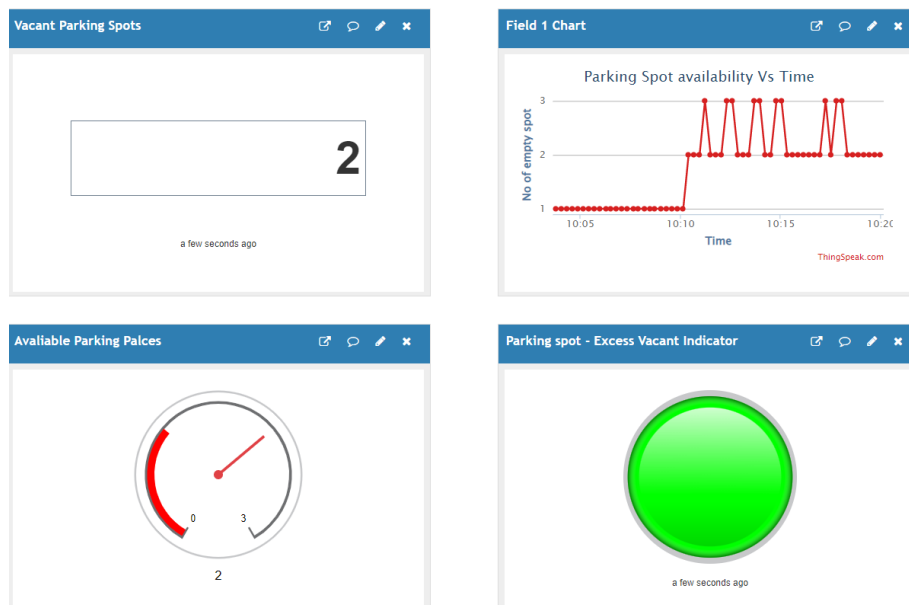


Figure 2 Sufficient vacant parking spot with GREEN lamp indicator for drivers at ThingSpeak website

Table of Contents

1. INTRODUCTION	3
2. PROBLEM DESCRIPTION	4
3. SOLUTION DESIGN	5
3.1 REQUIRED COMPONENTS	7
3.2 PROJECT WORKFLOW DIAGRAM.....	8
3.3 CIRCUIT CONNECTION.....	9
4. PROTOTYPE	9
4.1 ACHIEVED SOLUTION AND CODE PERFORMANCE	11
4.2 HIGHLIGHTS OF INTERESTING SUB-PROBLEM SOLVED	14
5. EVALUATION OF RESULTS	19
5.1 THINGSPEAK CLOUD PLATFORM RESULT ANALYSIS	24
6. CONCLUSION.....	28
7. BIBLIOGRAPHY	28

1. INTRODUCTION

The advent of the Internet of Things (IoT) has revolutionized the way we live and work. IoT describes as the interconnection of everyday smart devices such as sensors, vehicles, appliances, and other machines to the internet. This interconnectivity has enabled the development of smart systems that can enhance the efficiency and effectiveness of various services. One such application of IoT is the development of Smart Parking Control Systems.

This report discusses the design, development, and implementation of a Smart Parking Control System using IoT devices and how efficiently it has solved our issues. The Smart Parking System is a solution to the growing problem of parking in urban areas. The traditional parking systems are inefficient, time-consuming, and often result in traffic congestion. The Smart Parking System aims to provide a more efficient and convenient way of parking by leveraging IoT devices such as sensors, Wi-Fi Modules, and cloud platforms.

This report presents the design and implementation of a Smart Parking System that utilizes IoT devices to provide real-time information about parking availability to drivers. The system includes various components such as parking sensors(Ultrasonic Sensors), Servo motors, LCD display, Arduino UNO R3, and a cloud-based platform for data storage and analysis. The sensors are placed at designated parking spots, and they detect the presence of vehicles. The data from these sensors is transmitted to the cloud platform via ESP8266 Wi-Fi Module , which processes it through Arduino UNO and provides real-time information about parking availability to drivers through the internet.

The Smart Parking System has several benefits, including reducing traffic congestion, improving parking efficiency, and enhancing the overall driving experience. This report discusses the challenges faced during the development of the system, such as sensor calibration, network connectivity, and data security. The report also presents the results of the system's performance in terms of accuracy, reliability, and scalability. (al, 2019)

The Smart Parking System is an innovative solution that utilizes IoT devices to address the parking challenges faced in urban areas. The system offers several benefits and has the potential to enhance the overall driving experience. This report provides a comprehensive overview of the design, development, and implementation of the system, and it presents the challenges faced and the results obtained.

2. PROBLEM DESCRIPTION

The world's population growth has reached 8 billion (as per World-O-Meter report). Most of them want to live in Urban areas for a better lifestyle, comfort, education, and healthcare facilities. There are plenty of problems that arise daily in these metropolitan highly developed urban cities. Parking vehicles in the correct place is one of them. As per the report by (Hedges & Company), there are 1.446 billion vehicles on our earth and every vehicle needs to have a proper parking place where it can be parked safely. Even though more parking facilities are being built in many areas, there are still several problems with their efficient utilization. A major problem involves the time wasted searching for parking places, which not only causes road congestion and irritates the drivers searching for empty parking spaces, but also has a big negative impact on environmental pollution, traffic congestion, theft concerns, and unnecessary paying more parking duties. (al, 2019)

As per research conducted by "SpotHero & Co", An average American driver wastes 17 hours in every year for searching parking his/her vehicle. That adds up to \$73 billion in lost time, fuel, and emissions searching for parking across all drivers nationally. Below are some data collected for how many hours

Parking Search Time

City	On-street search time (mins per trip)	Off-street search time (mins per trip)	Parking trips (per week)
New York City	15	13	10
San Francisco	12	11	9
Los Angeles	12	11	9
Washington D.C.	10	9	9
Chicago	9	8	8
Seattle	9	8	9
Boston	8	8	8
Atlanta	8	8	8
Dallas	8	8	8
Detroit	6	6	7

Figure 3 Parking search time in various cities (Data source by INRIX) (SpotHero, 2017)

Parking Search Cost

City	Annual search time (hours per driver per year)	Parking search cost per driver per year	Total per city per year
New York	107	\$2,243	\$4.3bn
Los Angeles	85	\$1,785	\$3.7bn
San Francisco	83	\$1,735	\$655m
Washington D.C.	65	\$1,367	\$329m
Seattle	58	\$1,205	\$490m
Chicago	56	\$1,174	\$1.3bn
Boston	53	\$1,111	\$262m
Atlanta	50	\$1,043	\$251m
Dallas	48	\$995	\$726m
Detroit	35	\$731	\$209m

Figure 4 Parking search cost in various cities(Data source by INRIX) (SpotHero, 2017)

To manage these parking issues efficiently, we need to design a model where the driver can easily get the details of parking spots available near to his/her location. As the availability of the spot, the driver can easily go and park his/her vehicle and most of issue can be resolved efficiently.

Let's discuss the design of this project in the below segment.

3. SOLUTION DESIGN

Arduino UNO R3 is the brain of this project which would instruct the other components to execute the task and collect the data. Firstly, We need an I2C Module (Inter-Integrated circuit) which is used for the communication between the microcontrollers and other devices. It works on the principle of synchronous serial communication protocol with its two major segments called SDA(Serial Data Line) and SCL(Serial Clock Line). As we are using Arduino UNO R3, the SDA will connect to A4 and SCL will connect to the A5 pins of the Arduino board(ANALOG IN). The SDA line carries the data being transmitted, while the SCL line carries the clock signal that synchronizes the data transfer between the Arduino and I2C module devices. When we are connecting the I2C module with the 16X2 LCD display(by soldering its 16 pins), It has an inbuilt PCF8574 I2C chip that converts I2C serial data to parallel data for the LCD display. Ultimately, we can connect the Arduino to the LCD display and print the message at our convenience by validating and uploading the code in Arduino App. The use of the I2C adapter module will reduce the number of input/output pins connected to Arduino to four pins else we have to connect the sixteen pins to the LCD display which would lead to less number of available pins for other

components. Other than these two pins, I2C has another two pins called VCC and GND. VCC line would be connected to the 5V supply and GND will connect with the common ground of Arduino UNO. (Design, 2017)

In the next step, We need to use the sensors which would detect the vehicles parked in the parking spots. Here we are using the ultrasonic sensor module (HC-SR04). This has a similar working methodology as sonar and radar which first send the sound/radio waves respectively and once those echo waves will receive back, these sensors will evaluate the time duration. The time difference between the transmitted and received signals will help it to calculate the distance of the object present in front of the sensor. In an ultrasonic sensor, there are 4 pins. VCC, GND, TRIG (trigger), and ECHO. Trig and Echo pins are used to initiate the measurement of the distance to an object and to receive the echo pulse, respectively. This HC-SR04 ultrasonic sensor module generates ultrasonic sound at around 40KHz. The Trig terminal will initiate the signal and the echo terminal will receive the signal. They will connect with the Arduino UNO R3 and this will help to calculate the distance of the object from the ultrasonic sensor. On the other side, the VCC will connect with the 5V power supply and the GND will connect with the Ground. In this project, we are using 3 Ultrasonic sensors for checking the availabilities of the parking lots. We will discuss the circuit connection in detail in the next segments. (Mechatronics, 2019)

Now we need to send the data to the cloud for that we need a generic ESP8266 Wi-Fi module. This will help to get the internet connectivity as the name portrays it is a Wi-Fi module so once the user will provide the required SSID and password for the available Wi-Fi connection, then this device will connect to it and help the microcontroller (Arduino UNO R3) to connect to other devices through the internet. Technically, ESP8266 is a miniature microcontroller board, and we can control its functionality by executing the program code and can get the services that we need from it.

ESP8266 Wi-Fi module supports multiple protocols like TCP/IP, HTTP, DNS, UDP, UART, etc. Here it uses the UART (Universal Asynchronous Receiver and Transmitter) protocol to communicate with the Arduino UNO R3. It has 8 pins marked as VCC, RX, RESET, IO0, ENABLE, IO2, TX, and GROUND. We will connect the pins respectively with the Arduino UNO Pins to get the expected inputs and Outputs signals. The setup and connectivity will be discussed in the Implementation segment. (DevicePlusEditorialTeam, 2022)

As this ESP8266 module does not support or have any USB port via which we can compile and upload our executable codes so we need one programmer kit which will help us to do the task efficiently. In the Programmer kit, we need to attach the ESP8266 Wi-Fi module and then we can upload our codes. Before that, we need to sort the 2 pins of the programmer kit [GPIO0(IO0) and GROUND(GND)]. IO0 pin is used to make the ESP8266 module in boot mode. During normal operation, IO0 is set from high to enable the module's Wi-Fi functionality. However, when GPIO0 is pulled low (when connected to GND), the module enters firmware upload mode, allowing new firmware to be uploaded to the Wi-Fi module. By connecting IO0 to GND, the ESP8266 module will boot into firmware upload mode instead of normal operation mode. This is necessary to upload new firmware to the module. Once this is done, we can connect the programmer kit to the ESP8266 Wi-Fi module and upload the required codes for further functionalities. (DevicePlusEditorialTeam, 2022)

We need to have a “ThingSpeak” account (open-source cloud platform) which will help us to get the collected data by Arduino UNO R3 and we can do the required modification, generate the QR codes, provide access to the public, and monitor the data easily. Once We will create an account at ThingSpeak Cloud, We need to create a channel get the channel ID and the respective API keys for the channel. (ElectronicProjectHubTeam, 2020)

The last component required in this project is a Micro Servo motor - SG90. This will act as a parking barrier gate. There are 3 pins for this micro servo motor named power supply(VCC), Ground(GND), and signal(SIG). We will connect these pins to the respected Input/Output pins of Arduino UNO and control the incoming traffic to the parking area.

3.1 REQUIRED COMPONENTS

SL NO	Components	Description	Quantity
1	Arduino Uno	R3 model	1
2	I2C Module	HD44780 Compatible	1
3	LCD Display	16X2	1
4	Ultrasonic Sensors	HC - SR 04	4
5	ESP8266 Wi-Fi Module	1M baud upload speed	1
6	ESP8266 programmer Kit	USB Cable Enable	1
7	Servo Motor	5.5V - 9V	1
8	Power cable	USB Cable	1
9	Toy cars	Regular Size	4
10	Jumper wires	Male to Female	10
11	Jumper wires	Male to Male	8

Table 1 Components required for the project

3.2 PROJECT WORKFLOW DIAGRAM

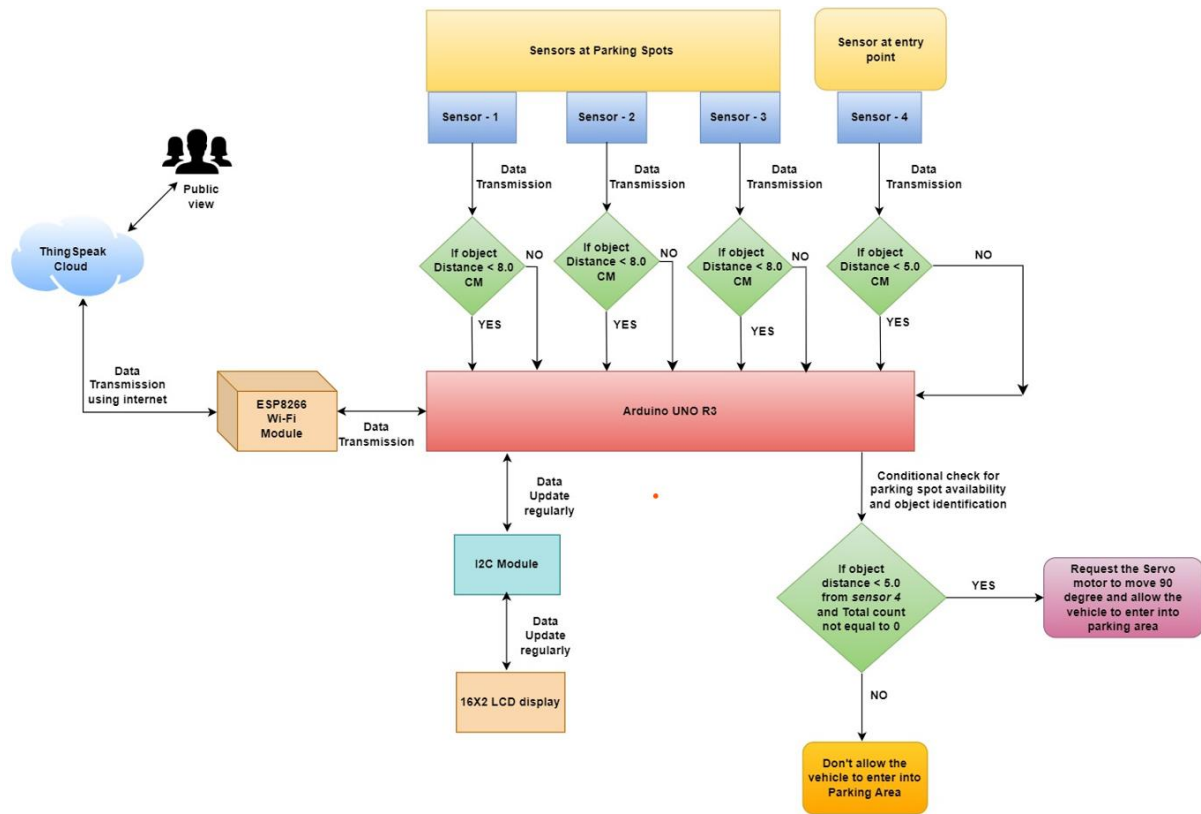


Figure 5 Flowchart diagram

3.3 CIRCUIT CONNECTION

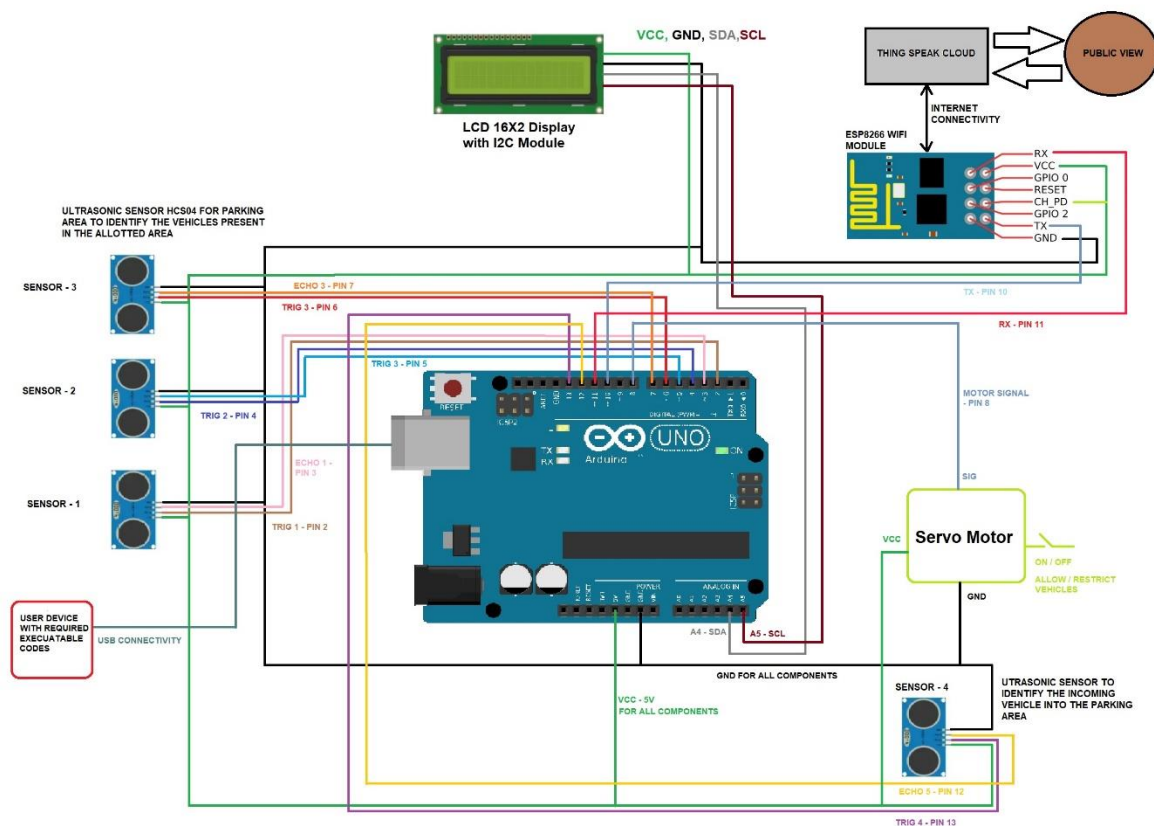


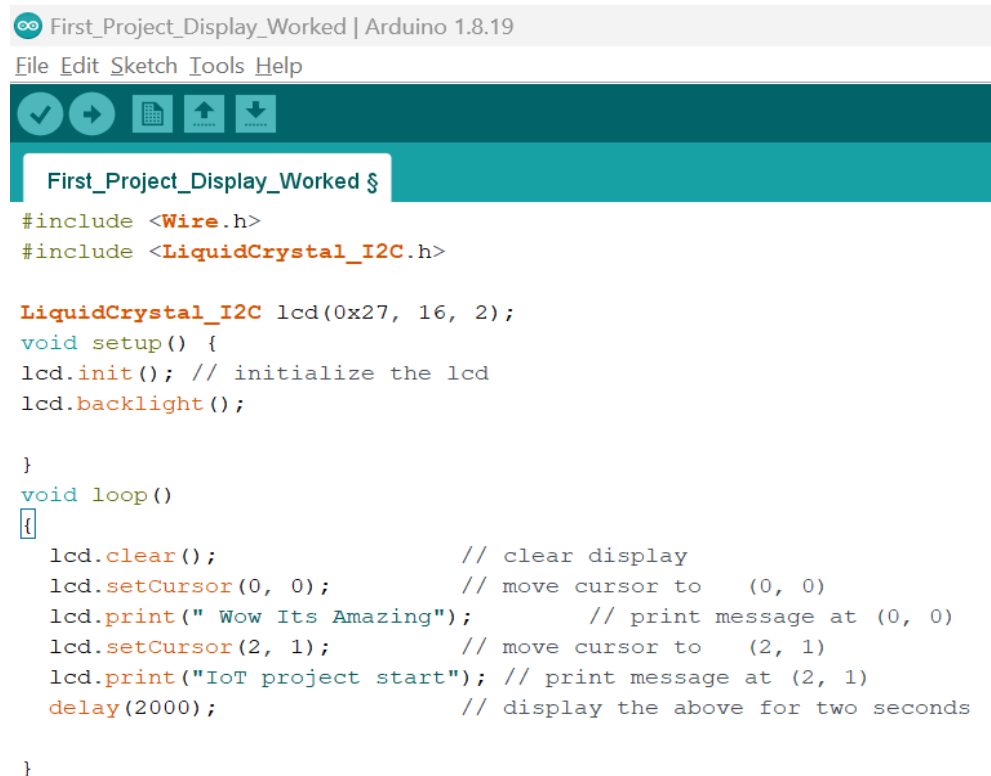
Figure 6 Complete Circuit diagram

4. PROTOTYPE

As per the circuit diagram, we need to connect the respective components to the Arduino kit. Now it is the time to understand how this works, sends and receives data from the different devices, and connects to each other. For this project, I have taken assistance from multiple sources which I have mentioned in the references.

Firstly, we need to solder the I2C module with the 16X2 LCD display which both have 16 pins each. This has multiple benefits we can control the LCD display via the I2C module with only 4 pins instead of the 16 pins occupied in the Arduino board. Moreover, the synchronous serial communication protocol concept that the I2C module follows has much better performance and continuity along with time. We have connected the I2C module's 4 pins with the Arduino UNO R3. The SDA and SCL have connected with the Analog input pins (A4 and A5) respectively. Whereas the VCC is connected to a 5V power supply and the GND is connected to the Ground pin of the Arduino. It works simply without any internet connectivity. We can execute the code and upload the same using the Arduino App and get the expected result to print over the LCD Display.

Below is a small example where I have executed a few lines of code then compiled them and uploaded it. After that, we could see the result on the LCD screen. (Ansar, 2019)



```
First_Project_Display_Worked | Arduino 1.8.19
File Edit Sketch Tools Help

First_Project_Display_Worked $
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);
void setup() {
  lcd.init(); // initialize the lcd
  lcd.backlight();
}
void loop()
{
  lcd.clear(); // clear display
  lcd.setCursor(0, 0); // move cursor to (0, 0)
  lcd.print(" Wow Its Amazing"); // print message at (0, 0)
  lcd.setCursor(2, 1); // move cursor to (2, 1)
  lcd.print("IoT project start"); // print message at (2, 1)
  delay(2000); // display the above for two seconds
}
```

Figure 7 LCD display program using Arduino App

We need the “Wire.h” and “LiquidCrystal_I2C” libraries for this program to be executed in our Arduino App. After this, we can initiate the LCD and backlight in the void setup() and in the void loop(), we can print the requested text on the LCD screen which will display continuously.

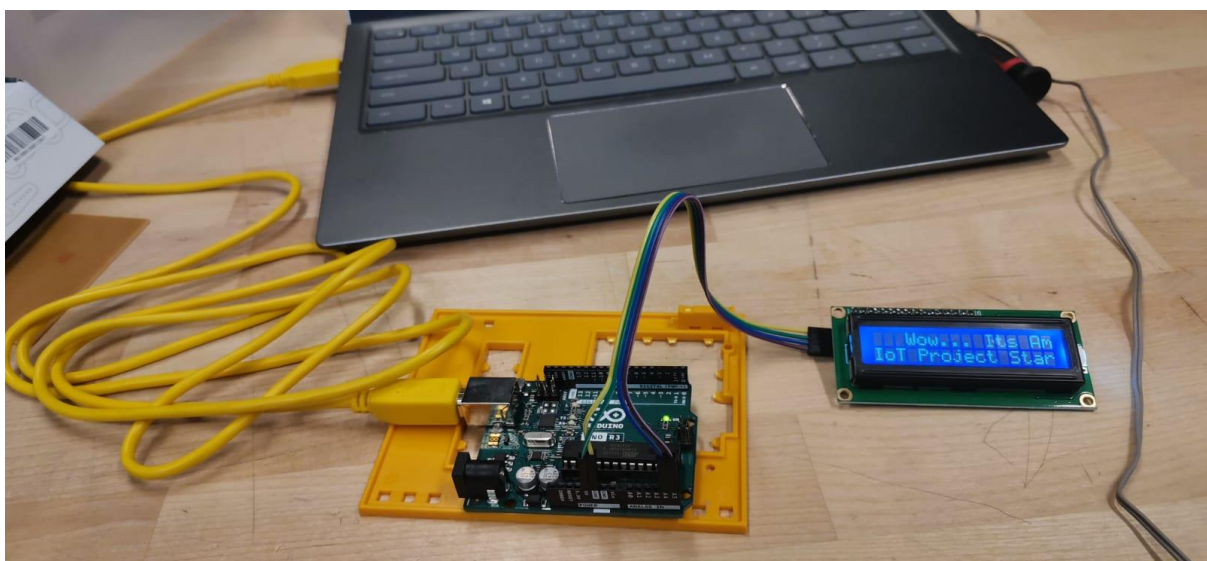


Figure 8 Expected Output on the LCD Screen

Now once this setup has worked fine, we can move to the next component setup which is the ultrasonic sensors setup. We need to connect the ultrasonic sensors HC - SR04 with the respective pins of the Arduino board as mentioned in the circuit diagram. All the VCC and GND need to be connected with the 5V supply pin and Ground of the Arduino and the Trigger (TRIG) and Echo(ECHO) pins will connect to the input and output pins of the Arduino UNO R3 board.

We have already discussed that the TRIG will transmit high-frequency sound waves and once they bounce back from the obstacle the ECHO pin will capture the sound waves. The difference between the duration of these two actions will help to calculate the distance of the object toward the ultrasonic sensor.

4.1 ACHIEVED SOLUTION AND CODE PERFORMANCE

First, we need to declare the input and output of the ultrasonic sensors(trig and echo) as constants and define them as integers (Arduino board pin location) in our program with respect to the circuit diagram which we have already defined in the above paragraph.

```
const int trig_1 = 2;
const int echo_1 = 3;
const int trig_2 = 4;
const int echo_2 = 5;
const int trig_3 = 6;
const int echo_3 = 7;
const int trigPin = 13;
const int echoPin = 12;
```

There are some other attributes that we need to define like the distance of the object from the sensor, expected distance, Time, parking lot positions, and timer_count. As the vehicle can be present at any distance from the sensor so we have defined it in the "float" with the distance as 8 cm. Along with that, all the values as defined below have the initial value of "0".

```
float distance_in_CM_01 = 0, result_in_CM_01 = 0;
float distance_in_CM_02 = 0, result_in_CM_02 = 0;
float distance_in_CM_03 = 0, result_in_CM_03 = 0;
long Time_1, Time_2, Time_3;
float car_1, car_2, car_3;
float Distance_01 = 8.0, Distance_02 = 8.0, Distance_03 = 8.0;
int total = 0, Timer_Count = 0;
```

Now we need to initialize the variables, pin modes, and other necessary activities for this project. We have used the void setup() function call where we have put the required initializers. Like first we set the baud rate for the Arduino UNO board, then we defined the pin mode (as inputs and outputs) for all trig and echo pins. After that, it defined the servo motor signal pin and initiated the servo motor to write which would set that to

its initial position. Then, in the program, we declared the LCD initialization and a welcome message would reflect as per the cursor positions ("IoT Car Parking Monitoring System").

```
void setup()
{
  mySerial.begin(115200);
  /* Function call in the Arduino programming language that sets the baud rate
  for serial communication between the Arduino board and another device*/
  pinMode(trig_1, OUTPUT); /* Initialize the pin mode for all trig and echo connection*/
  pinMode(trig_2, OUTPUT);
  pinMode(trig_3, OUTPUT);
  pinMode(echo_1, INPUT);
  pinMode(echo_2, INPUT);
  pinMode(echo_3, INPUT);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  // Initialize the servo motor
  myServo.attach(servoPin);
  // Set the servo motor to its initial position
  myServo.write(0);
  digitalWrite(trig_1, LOW);
  digitalWrite(trig_2, LOW);
  digitalWrite(trig_3, LOW);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("  IoT CAR PARK");
  lcd.setCursor(0, 1);
  lcd.print(" MONITOR SYSTEM");
  delay(2000);
  lcd.clear();
}
```

Now, we will move towards the void loop() section which this program will iterate infinitely times till the power supply is uninterrupted or there would be any change in the program.

Here first we have initiated some values like the total parking spot as "0" and we have defined the three vehicle positions with the sensors respectively.

```
void loop()
{
  total = 0;
  car_1 = sensor_1();
  car_2 = sensor_2();
  car_3 = sensor_3();
```

Here comes the main logic behind our codes how the sensor will validate the vehicle position and collect the data with respect to which we will assume that weather the parking spot is free or occupied.

```
  lcd.setCursor(0, 0);
  lcd.print("CAR1:");
  if (car_1 <= Distance_object_01)
  {
    lcd.print("OK ");
  }
  else
  {
    total += 1;
  }
  if (car_1 > Distance_object_01)    lcd.print("NO ");
```

→ "If-else" condition for identification for parking vehicle at certain distance at the parking spot - 1

We have already defined in the void setup() function as the Distance_object_01 as "8.0" so If the value of car_1 is less than or equal to "8.0" then the LCD will print as OK (Means parking spot occupied) besides the CAR1. This signified that the car parking spot - 1 is full and the total value would automatically add 1 to the original value. On the other side if there is no object then it will print NO(which means the parking spot is available).

We have used the same logic for the other two parking spot as mentioned in the below codes.

```

lcd.print("CAR2:");
if (car_2 <= Distance_object_02)
{
    lcd.print("OK ");
}
else
{
    total += 1;
}
if (car_2 > Distance_object_02)    lcd.print("NO ");
lcd.setCursor(0, 1);
lcd.print("CAR3:");
if (car_3 <= Distance_object_03)
{
    lcd.print("OK ");
}
else
{
    total += 1;
}
if (car_3 > Distance_object_03)    lcd.print("NO ");

```

Conditions for parking spot - 2

Condition for parking spot -3

After these conditions, we need to have another field where it will show how many total vacant parking spots are available in the parking area. For that, we have used the below logic.

```

lcd.print("FREE:");
lcd.print(total);
if (timer_count >= 50)
{
    mySerial.print('*');
    mySerial.print(total);
    mySerial.println('#');
    timer_count = 0;
}
timer_count += 1;

```

After this, We need to send the ultrasonic signal to ensure the distance of the object from the sensors. Initially, we need to set this digitalWrite function with the trigger pin as low (deactivate mode) after 2 microseconds delay, we will make this pin HIGH (Ultrasonic signal sent). Then we need to give a delay of 10 microseconds after which we need to make it LOW as the bounce back signal needs to be captured by the Echo pins.

```

// Send an ultrasonic signal
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);

// Measure the distance using the ultrasonic sensor
long duration = pulseIn(echoPin, HIGH);
int distance = duration / 58;

```

For measuring the distance, we need to define a function called `pulseIn` that measures the length of a pulse on a specified digital input pin (`echoPin`). The `HIGH` parameter indicates that the function should wait for a `HIGH` pulse on the `echoPin` before timing how long the pulse lasts. Moreover, we can calculate the distance in centimeters from the time in microseconds when we divide the duration by 58.

4.2 HIGHLIGHTS OF INTERESTING SUB-PROBLEM SOLVED

Once these sensor parts are done, we will work on the servo motor part where we can allow and restrict the car from entering the parking area. As we have already mentioned we are using one ultrasonic sensor(SR_04 as per the circuit diagram) which will detect the vehicle and pass the information to the Arduino Uno after which as per the parking spot availability, our designed program will request the Arduino to move the servo motor. Let's explore this with the below codes:

```

// If the distance is less than 5cm, move the servo motor
if (distance < 5 && total != 0) {
  // Move the servo motor to 90 degrees
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("  Welcome  ");
  lcd.setCursor(0, 1);
  lcd.print("  To Parking  ");
  myServo.write(90);

  delay(5000);
  total--;
  // Move the servo motor back to its original position
  myServo.write(0);
  delay(1000);
}else {
  //Set the servo motor to its initial position
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("  Sorry  ");
  lcd.setCursor(0, 1);
  lcd.print("Parking is Full");
  myServo.write(0);
}

```

As we can see from the conditions that if the ultrasonic sensor – 4 would detect an object below 5 cm distance and the total available parking spot is not “zero” then the LCD will display a message as “Welcome to parking” and it will instruct the servo motor to move 90 degrees to allow the vehicle to enter the parking area. After a delay of 5 secs, the servo motor will back to its original position. In case, any of these two conditions won’t be fulfilled, then the LCD will display “Sorry parking is full”.

Now, We will move to the last part of our program which will assist us with the calculation of the distance of the object from the ultrasonic sensor. For this, we need to have a float number to return in the last. Firstly, we need to set the trig with a high signal and then we can make it as low. Secondly, we can calculate the time pulse to bounce back from the object and return to the ultrasonic sensor using the `pulseIn()` function. The time between the sent and received signal will be measured. To calculate the distance, we will multiply the time with “0.034” as the sound speed in air is approx. 34,000 cm/sec. Finally, We need to divide the value by 2 as the pulse has traveled 2 times (triggered and bounced back). Then we can get the exact distance of the object toward the sensor. This return value will be used in the conditions mentioned in the above paragraph for checking the free parking spots in the parking area. (Albertus Ega Dwiputra, 2018)

```

float sensor_1(void)
{
    digitalWrite(trig_1, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig_1, LOW);
    Time_1 = pulseIn(echo_1, HIGH);
    distance_in_CM_01 = Time_1 * 0.034;
    return result_in_CM_01 = distance_in_CM_01 / 2;
}

```

Like the first case, we can follow the same for the other two parking spots.

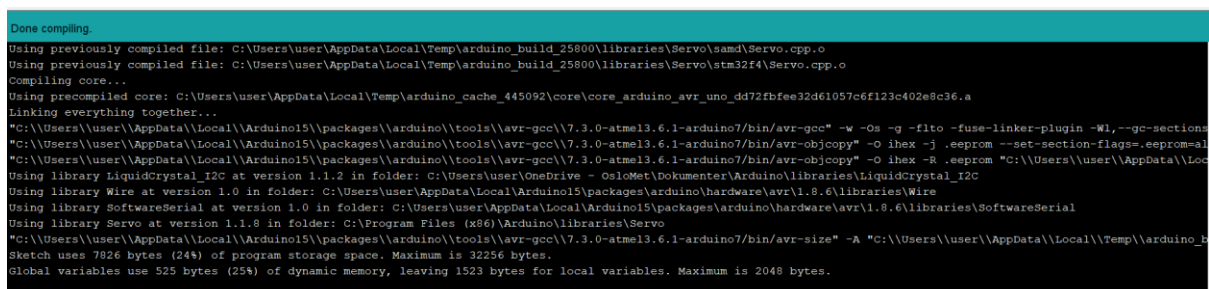
```

float sensor_2(void)
{
    digitalWrite(trig_2, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig_2, LOW);
    Time_2 = pulseIn(echo_2, HIGH);
    distance_in_CM_02 = Time_2 * 0.034;
    return result_in_CM_02 = distance_in_CM_02 / 2;
}

float sensor_3(void)
{
    digitalWrite(trig_3, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig_3, LOW);
    Time_3 = pulseIn(echo_3, HIGH);
    distance_in_CM_03 = Time_3 * 0.034;
    return result_in_CM_03 = distance_in_CM_03 / 2;
}

```

Once all these are done, we can check the codes and compile the same in our Arduino app. The result should look as below.



```

Done compiling.
Using previously compiled file: C:\Users\user\AppData\Local\Temp\arduino_build_25800\libraries\Servo\samd\Servo.cpp.o
Using previously compiled file: C:\Users\user\AppData\Local\Temp\arduino_build_25800\libraries\Servo\stm32f4\Servo.cpp.o
Compiling core...
Using precompiled core: C:\Users\user\AppData\Local\Temp\arduino_cache_445092\core\core_arduino_avr_uno_dd72fbfee32d61057c6f123c402e8c36.a
Linking everything together...
"C:\Users\user\AppData\Local\Arduino15\packages\arduino\tools\avr-gcc\7.3.0-atmel3.6.1-arduino7/bin/avr-gcc" -w -Os -g -flto -fuse-linker-plugin -Wl,--gc-sections
"C:\Users\user\AppData\Local\Arduino15\packages\arduino\tools\avr-gcc\7.3.0-atmel3.6.1-arduino7/bin/avr-objcopy" -O ihex -j .eeprom --set-section-flags=.eeprom=alloc,store_bits,ram_bits,rom_bits --no-sections "C:\Users\user\AppData\Local\Temp\arduino_build_25800\sketch\sketch.ino.o" -O ihex -R .eeprom "C:\Users\user\AppData\Local\Temp\arduino_build_25800\sketch\sketch.ino.hex"
Using library Wire at version 1.0 in folder: C:\Users\user\AppData\Local\Arduino15\packages\arduino\hardware\avr\1.8.6\libraries\Wire
Using library LiquidCrystal_I2C at version 1.1.2 in folder: C:\Users\user\AppData\Local\Arduino15\packages\arduino\hardware\avr\1.8.6\libraries\LiquidCrystal_I2C
Using library SoftwareSerial at version 1.0 in folder: C:\Users\user\AppData\Local\Arduino15\packages\arduino\hardware\avr\1.8.6\libraries\SoftwareSerial
Using library Servo at version 1.1.8 in folder: C:\Program Files (x86)\Arduino\libraries\Servo
"C:\Users\user\AppData\Local\Arduino15\packages\arduino\tools\avr-gcc\7.3.0-atmel3.6.1-arduino7/bin/avr-size" -A "C:\Users\user\AppData\Local\Temp\arduino_build_25800\sketch\sketch.ino.hex"
Sketch uses 7826 bytes (24%) of program storage space. Maximum is 32256 bytes.
Global variables use 525 bytes (25%) of dynamic memory, leaving 1523 bytes for local variables. Maximum is 2048 bytes.

```



```
Done uploading.
avrdude: Writing flash (7826 bytes):

Writing | ##### | 100% 1.30s

avrdude: 7826 bytes of flash written
avrdude: verifying flash memory against C:\Users\user\AppData\Local\Temp\arduino_build_25800/Demo_Version_Smart_Car_Parking_System.ino.hex:
avrdude: load data flash data from input file C:\Users\user\AppData\Local\Temp\arduino_build_25800/Demo_Version_Smart_Car_Parking_System.ino.hex:
avrdude: input file C:\Users\user\AppData\Local\Temp\arduino_build_25800/Demo_Version_Smart_Car_Parking_System.ino.hex contains 7826 bytes
avrdude: reading on-chip flash data:

Reading | ##### | 100% 1.02s

avrdude: verifying ...
avrdude: 7826 bytes of flash verified

avrdude done. Thank you.
```

After this, we can check the hardware and test various test cases which we will discuss in the result segment. Now we will discuss how we can efficiently integrate the ESP8266 Wi-Fi module and portray the collected data by Arduino into the cloud so that it can be accessed on public platforms.

For that, we need one ESP8266 programmer which will merge with the ESP8266 Wi-Fi module and sort the 2 pins of the programmer (IO0 to GND) so that the ESP8266 module will boot into firmware upload mode instead of normal operation mode. Now we can write our codes and upload the same to the ESP8266 Wi-Fi module with the USB (Port - COM4) easily. Once this will connect with the common Wi-Fi then we can view the collected data in the cloud platform.

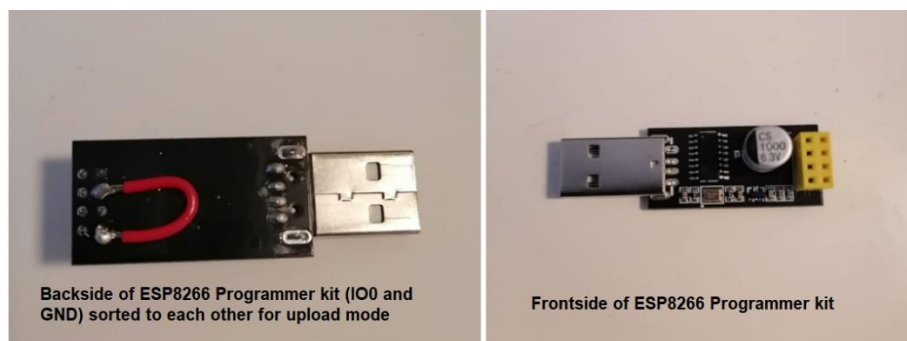


Figure 9 ESP8266 Programmer kit

In the below codes, we have added the “ThingSpeak.h” and “ESP8266Wi-Fi.h” libraries. Then we need to add the ESP8266 Board which will help us to execute the code by merging both ESP8266 programmer kit and ESP8266 Wi-Fi module. After that, in the codes “SSID” and “password” for the router has highlighted. Moreover, we have added the “channel-id” and “API keys” so that the collected data would refer to the specific channel. (ElectronicProjectHubTeam, 2020)



Figure 10 Procedure for uploading codes in ESP8266 Wi-Fi Module

```

#include "ThingSpeak.h"
#include <ESP8266WiFi.h>

//***** WI-FI Credentials *****/
char ssid[] = "TP-Link_";
char pass[] = "17";

//***** Channel details with API Keys *****/
unsigned long Channel_ID = 20;
const char * myWriteAPIKey = "XXXXXXXXXXXX";

```

Notes: For security reasons, sensitive pieces of information are not disclosed here

After this, we have declared some constants (field number) with an empty string which values will add as per the time, initiating the "client" from the Wi-Fi Client class from the ESP8266 library.

```

const int Field_Number_1 = 1;
String value = "";
int value_1 = 0;
WiFiClient client;

```

In the section of void setup() function, we can put communication with a certain baud rate (115200), WiFi mode will help to connect the ESP8266 with the above-mentioned WiFi credentials, initiating the ThingSpeak library with the variable as a client and finally, the internet connectivity for ESP8266 to connect to the WiFi network.

```

void setup()
{
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    ThingSpeak.begin(client);
    internet();
}

```

In the void loop() function, we are checking if the ESP8266 is connected to the Wi-Fi network, and it will repeat until the power is removed or the codes are reset. In the Void internet() function the program ensures that every time it will connect with the internet or the provided Wi-Fi network and finally the void upload() function helps to upload the data in a proper time sequence to the mentioned channel and respective API keys.

ESP8266 Wi-Fi Module and Arduino UNO final codes are present in my git account please have a look for better clarifications: https://s371102.github.io/loT_Project/

5. EVALUATION OF RESULTS

Once the circuit connection is done properly and the written codes are compiled and uploaded to the ESP8266 Wi-Fi Module and the Arduino UNO R3 then we can proceed with the test results and check whether the codes are working fine as per our expectation or not. We will analyze various test cases which will provide enough evidence for the success of this project.

First, We will see the project setup as mentioned in the below figures.



Figure 6 Entrance LCD Display of Car Parking



Figure 7 Sensor Positions at car parking

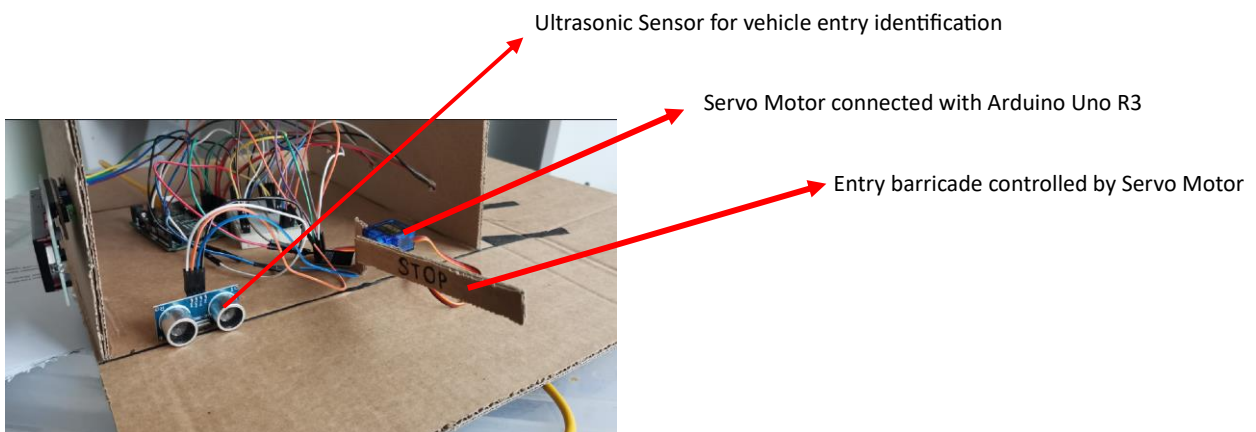


Figure 11 Different Items of Circuit

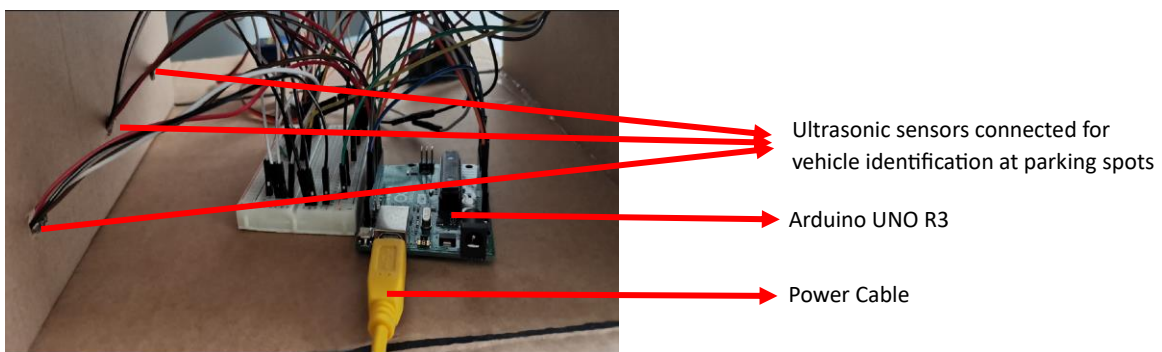
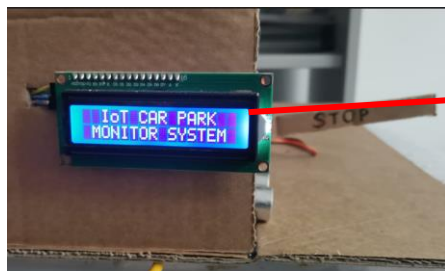


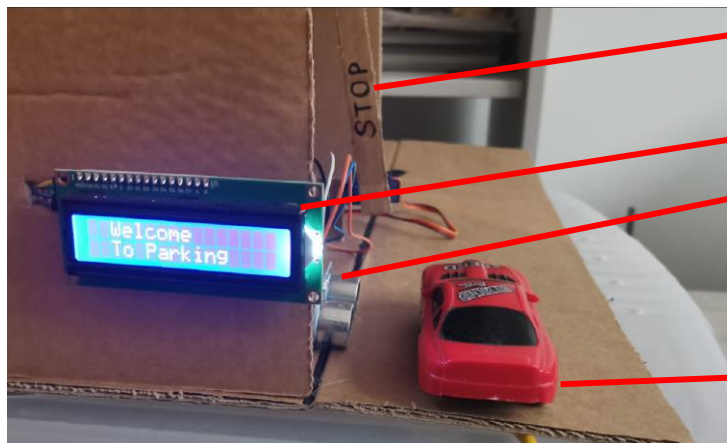
Figure 12 Power Cable and Arduino Connection



Starting display

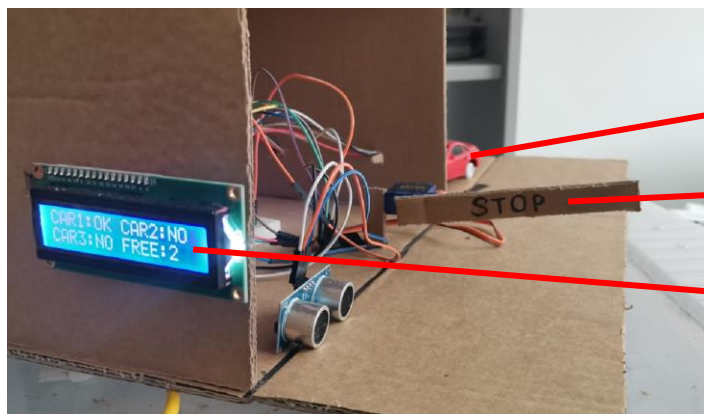


Status of car parking
NO : Spot available
OK : Spot Occupied
FREE :No of available parking spots



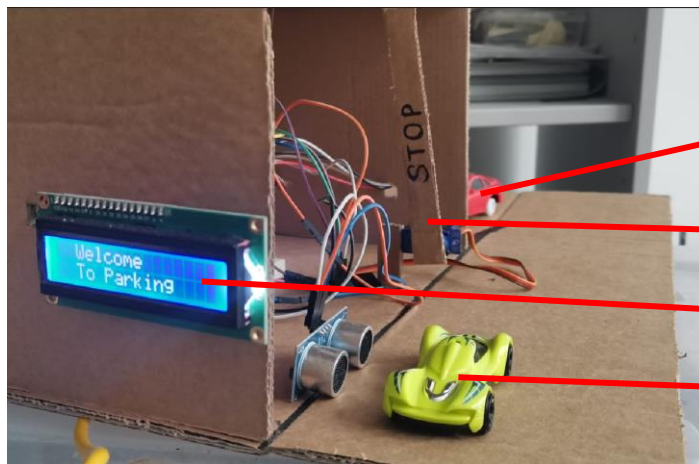
4. Servo motor moves 90 degree and allow the vehicle to enter the parking area as per availability of parking spots
3. Welcome message displays on the LCD screen
2. Ultrasonic Sensor Identifies that vehicle's presence
1. Car arrives at the parking entrance

Figure 13 Car enter phase inside parking area



1. Car got parked in the vacant parking area
2. Servo motor move back to its original position
3. In the LCD display, CAR1 got "OK" which means parking occupied and the FREE value is reduced to 3 to 2

Figure 14 First car parked and showed status in LCD Display



First Car parked on the Spot-1

3. Barricade opened and allow the vehicle to enter as available parking spot is more than 0

2. Welcome display on the LCD screen

1. Second Car arrives at entry point of the car parking

Figure 15 Second Car entered into parking area



Parking spot-3 is vacant

Second Car at parking spot-2

First Car at parking spot-1

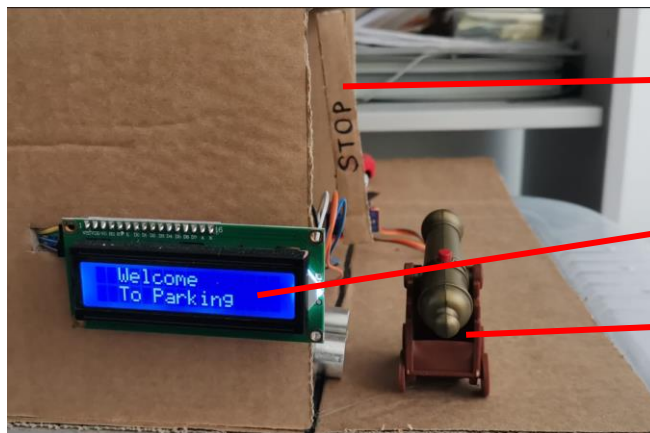
Figure 16 Two cars parked successfully



Servo motor backs to its original position

After second car parked, LCD displayed the status of parking

Figure 17 LCD displays the parking spot status

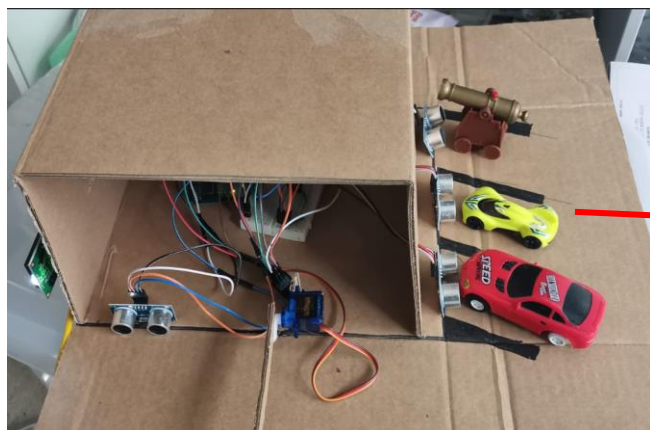


The barricade opened and servo motor moves 90 degrees to allow the vehicle to enter into parking area

Welcome message at LCD parking

3rd Vehicle arrived at entry point of the car parking

Figure 18 Third Car enters into parking area



All parking spot occupied

Figure 19 Three cars parked in the parking area



Barricade didn't allow the vehicle as no free parking space

Display shows the parking status

4th Car arrives at parking entrance

Figure 20 4th car at the entry point

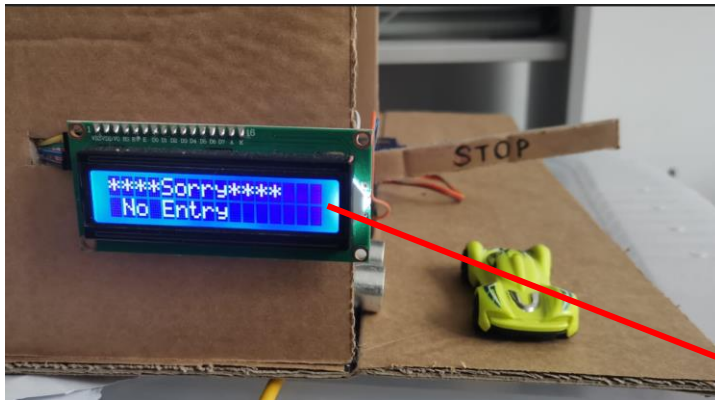


Figure 21 No Entry display on LCD screen



Figure 22 No parking display on LCD screen

LCD display show the message of no parking spot available

SL No	Ultrasonic Sensors for Parking Spots			Free Parking Spots	Ultrasonic Sensor for Servo Motor	LCD Display	Parking status
	Sensor - 1	Sensor - 2	Sensor - 3		Sensor - 4		
1	Free	Free	Free	3	Object Found	Welcome to parking	Allow vehicles to enter the parking area
2	Occupied	Free	Free	2	Object Found	Welcome to parking	Allow vehicles to enter the parking area
3	Occupied	Occupied	Free	1	Object Found	Welcome to parking	Allow vehicles to enter the parking area
4	Occupied	Occupied	Occupied	0	Object Found	Sorry Parking is full	Restrict the vehicle to enter the parking area
5	Free/Occupied	Free/Occupied	Free/Occupied	Any Value between (0-3)	Object not found	No Action	No Action

Table 2 Test result observation

5.1 THINGSPEAK CLOUD PLATFORM RESULT ANALYSIS

We can configure the ThingSpeak cloud platform by sharing the channel Id and API keys with the ESP8266 Wi-Fi module and adding various widgets in public view.

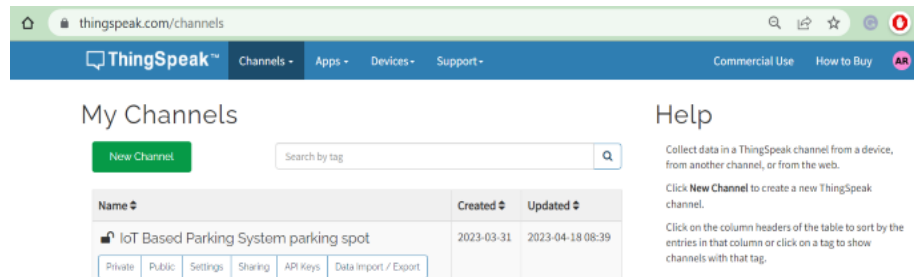


Figure 23 ThingSpeak cloud platform channel details

As we can see in the below images, we can configure the comparison graph between the two quantities (Parking spot available Vs Time). This will help to monitor the time series volume in a day when the car park is mostly used.

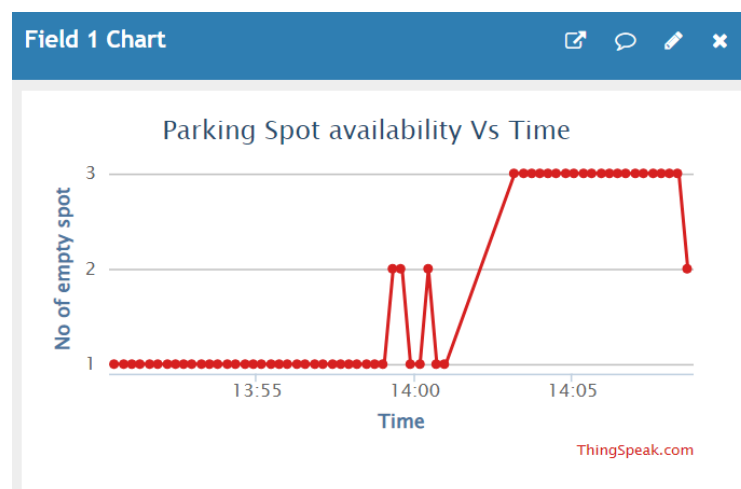


Figure 24 Number of parking spot vs Time

Generating the available parking spot in a single window makes it easy to identify the availability. This attribute will help the drivers to get exact information regarding the free parking spots.

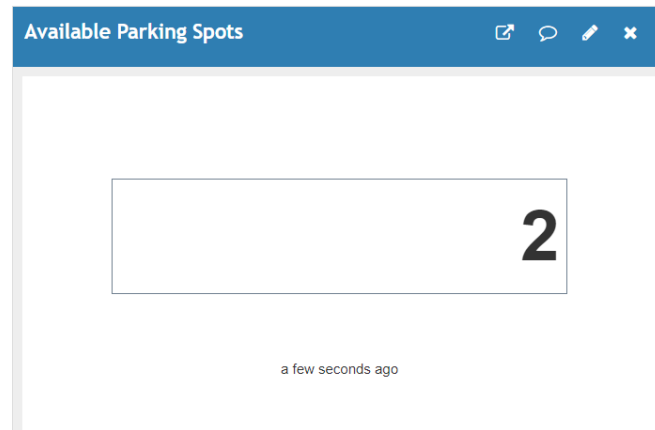


Figure 25 Available parking spots

An alert meter can be set up where we can highlight the meter with red when the parking area is available at less than 30% of its maximum capacity.

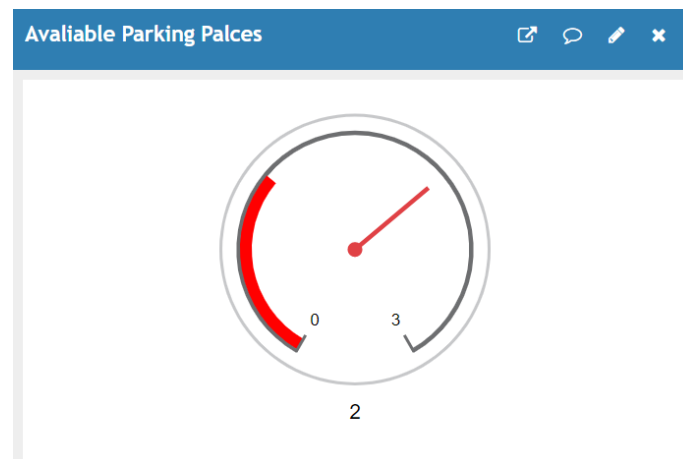


Figure 26 Alert indicator for available parking places

Once we generate the QR code of this public view page, then the user could easily scan the QR and check the parking slots available in different parking areas.

We can use a lamp indicator when the parking spot range is less than equal to 1 then the lamp indicator will blow which indicates the drivers who are looking for the parking spot they can easily notice that this parking area has limited numbers of parking spots so better to try in other parking zones

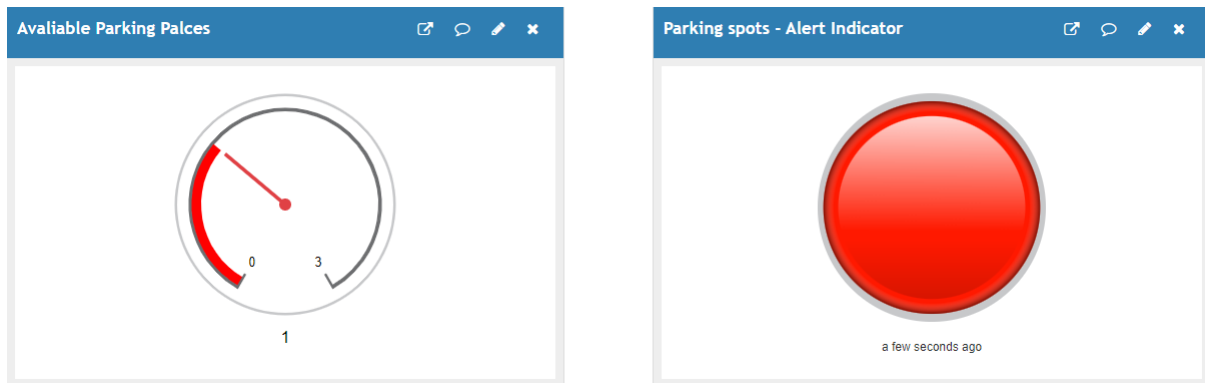


Figure 27 Lamp indicator for less available parking spots

When the available parking spots increased to 2, then the Alert indicator lights turned off. So As per our convenience we can setup the same for bigger number of parking areas.

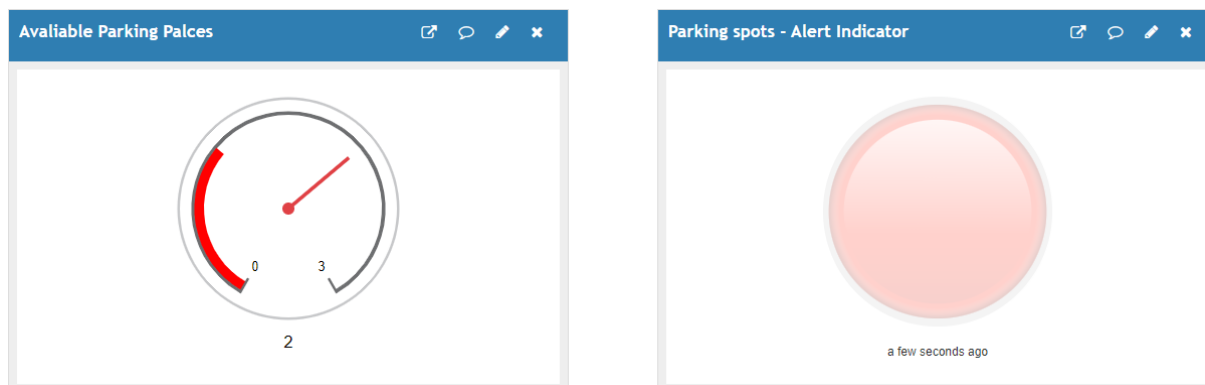


Figure 28 Lamp indicator is off when more parking spots are there

When there are more than equal to 2 car parking spots available in the parking area, the Green indicator will light up which will show the drivers online that they can park their cars in the parking area.

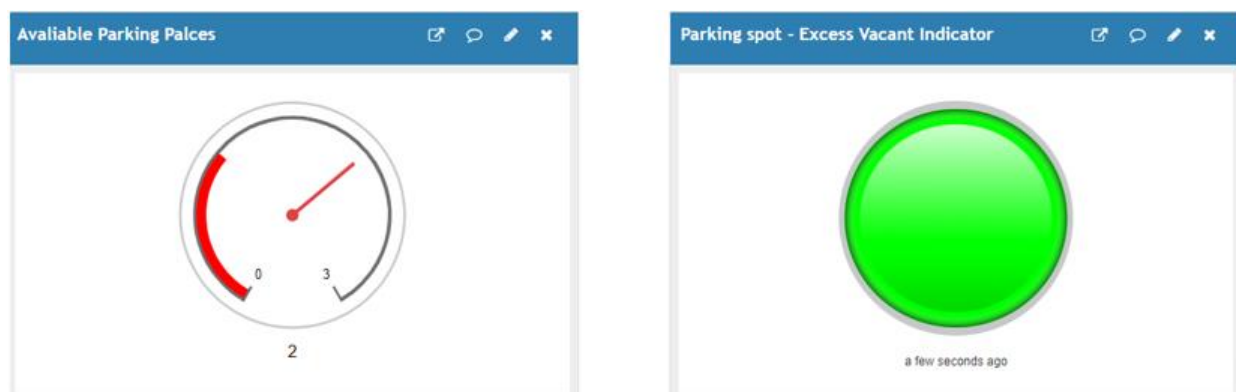


Figure 29 Lamp with Green lights when more parking spots are available

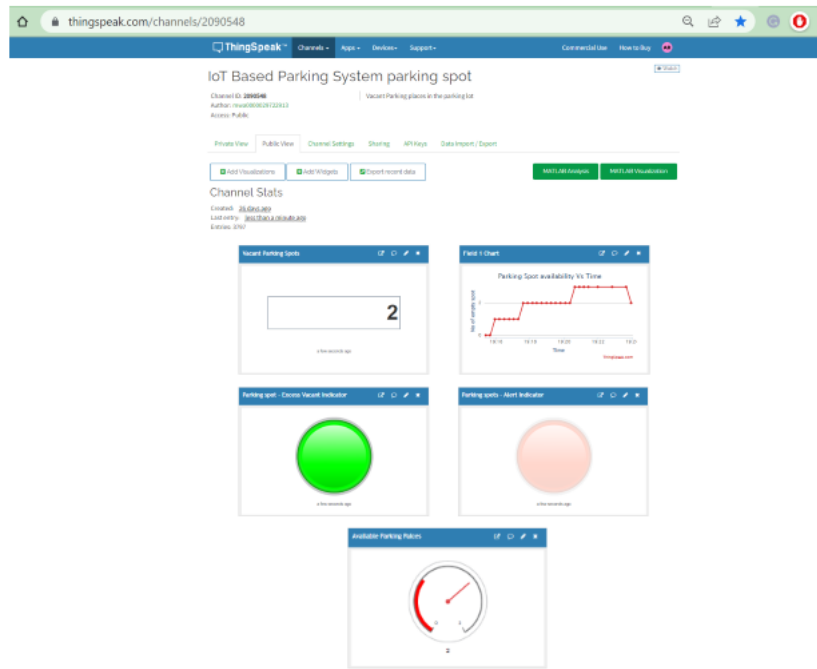


Figure 30 ThingSpeak Public View Final Dashboard

We can generate the QR codes by using which the drivers can easily check the available parking spots in different parking areas. This will land the user on the public view side.

Scan QR Code



Figure 31 ThingSpeak QR code scanners for car parking

6. CONCLUSION

In this project, I have tried to optimize the use of available parking space, reduce traffic congestion, and improve the overall parking experience for drivers. A major part of this project which I have explained in the implementation and result analysis have achieved successfully. There are certain areas that can be improvised like the drivers can book the parking spots in advance, paying the parking fees with online transactions, notifications before 15 mins of the parking time ends which couldn't achieve due to time limitation.

Additionally, the integration of smart technology can enhance the efficiency of parking enforcement, making it easier to manage and regulate parking violations. While the initial cost of implementing such a system may be high, the long-term benefits for both the city and its citizens make it a worthwhile investment.

Overall, this project has enlightened my knowledge towards many new technologies and concepts. Using Sensors, Arduino kit, Servo motors and all other components along with sending data to cloud via ESP8266 using internet was a major learning experience. This experience would definitely assist me for the future work in the field of IoT.

7. BIBLIOGRAPHY

- al, p. e. (2019). *iopscience.iop.org*. Retrieved from iopscience.iop.org:
<https://iopscience.iop.org/article/10.1088/1742-6596/1228/1/012066>
- Albertus Ega Dwiputra, H. K. (2018). IoT-Based Car's Parking Monitoring System.
- Ansar, M. (2019). Retrieved from https://www.youtube.com/watch?v=kGFyBp_PN-s
- Design, T. (2017). *blog.techdesign.com*. Retrieved from blog.techdesign.com:
<https://blog.techdesign.com/use-i2c-control-lcd-module-arduino/#:~:text=I2C%20Bus%20enables%20%20devices,also%20called%20%E2%80%9Cbidiirectional%E2%80%9D%20protocol.>
- DevicePlusEditorialTeam. (2022). *www.deviceplus.com*. Retrieved from deviceplus.com:
<https://www.deviceplus.com/arduino/esp8266-setup-tutorial-using-arduino/>
- ElectronicProjectHubTeam. (2020). *electronics-project-hub.com*. Retrieved from electronics-project-hub.com: <https://electronics-project-hub.com/iot-based-car-parking-system-using-arduino/>
- Mechatronics. (2019). *howtomechatronics.com*. Retrieved from howtomechatronics.com:
https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/?utm_content=cmp-true

SpotHero, A. o. (2017). <https://blog.spothero.com/>. Retrieved from spothero:
<https://blog.spothero.com/park-smarter-parking-search-time#:~:text=The%20average%20American%20driver%20wastes,parking%20across%20all%20drivers%20nationally.>

spothero, b. (2017). Stop Wasting Time Searching for Parking. Retrieved from blog.spothero.com:
<https://blog.spothero.com/park-smarter-parking-search-time#:~:text=The%20average%20American%20driver%20wastes,parking%20across%20all%20drivers%20nationally.>