IOT BASED SMART PARKING SYSTEM

A PROJECT REPORT FOR EMBEDDED SYSTEM DESIGN LABORATORY

Submitted by

KEERTHIVASAN G **2021505020**

NISHAN R **2021505027**

TARUN S **2021505050**

SATHISH KUMAR **2021505304**

VIKRAM **2021505305**

RAGHUL **2021505315**

MADRAS INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY : CHENNAI 600 044

JANUARY 2023

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The IoT-enabled Smart Parking project aims to revolutionize urban parking management by leveraging the power of Internet of Things (IoT) technology. This system employs sensor networks and real-time data analytics to monitor parking spa JUMPER WIRE ce availability, providing users with accurate and up-to-date information through a mobile application or other platforms. By optimizing parking utilization, reducing congestion, and minimizing environmental impact, this innovative solution contributes to a more efficient and sustainable urban infrastructure. The integration of IoT devices not only enhances the overall user experience but also facilitates smart city development by addressing one of the critical challenges in urban mobility

CHAPTER 1

INTRODUCTION

In response to the escalating challenges of urban parking, exacerbated by burgeoning urbanization, this project introduces a cutting-edge IoT-enabled Smart Parking system. Leveraging the power of five Infrared (IR) modules, a Servo motor, an LCD display, and the ESP8266 microcontroller, this system goes beyond conventional approaches. The IR modules serve as precision sensors, detecting the occupancy status of parking spaces in real time. The Servo motor facilitates automated gate control, allowing seamless entry and exit for vehicles. The LCD display provides on-site information, while the ESP8266 ensures connectivity and data exchange. Furthermore, the integration with Adafruit IO creates a robust platform for centralized data management and user interfaces. This project not only optimizes parking utilization but also exemplifies the transformative potential of IoT in reshaping ur JUMPER WIRE ban infrastructure and enhancing the overall urban mobility experience

CHAPTER 2

METHODOLOGY

2.1 OVERVIEW

ESP8266's built-in Wi-Fi capabilities, the system seamlessly communicates real-time data to and from Adafruit IO, facilitating centralized data management.The servo motor, controlled by the ESP8266, ensures automated gate operations, providing a seamless entry and exit experience for vehicles.Utilizing an LCD display interfaced with the ESP8266, real-tim JUMPER WIRE e parking space occupancy status is visually presented at the entrance for user convenience.The ESP8266 orchestrates the transmission of parking data, encompassing occupancy details, to Adafruit IO, ensuring continuous and secure data exchange.The system provides real-time monitoring of parking spaces, contributing to efficient space utilization and minimizing unnecessary traffic within the parking area.

2.2 HARWARE REQUIRED

1) ESP8266

2) IR module

3)LCD 16X2 display

4)Servo motor

5)Jumper wire

6)i2c module

.

2.3 NODEMCU ESP8266

The NodeMCU (Node MicroController Unit) is an open-source software and hardware development environment built around an inexpensive System-on-a-Chip (SoC) called the ESP8266. The ESP8266, designed and manufactured by Espressif Systems, contains the crucial elements of a computer: CPU, RAM, networking (WiFi), and even a modern operating system and SDK. That makes it an excellent choice for Internet of Things (IoT) projects of all kinds.

Figure – 2.1 NODEMCU ESP8266

2.4 PIN CONFIGURATION OF NODEMCU ESP8266

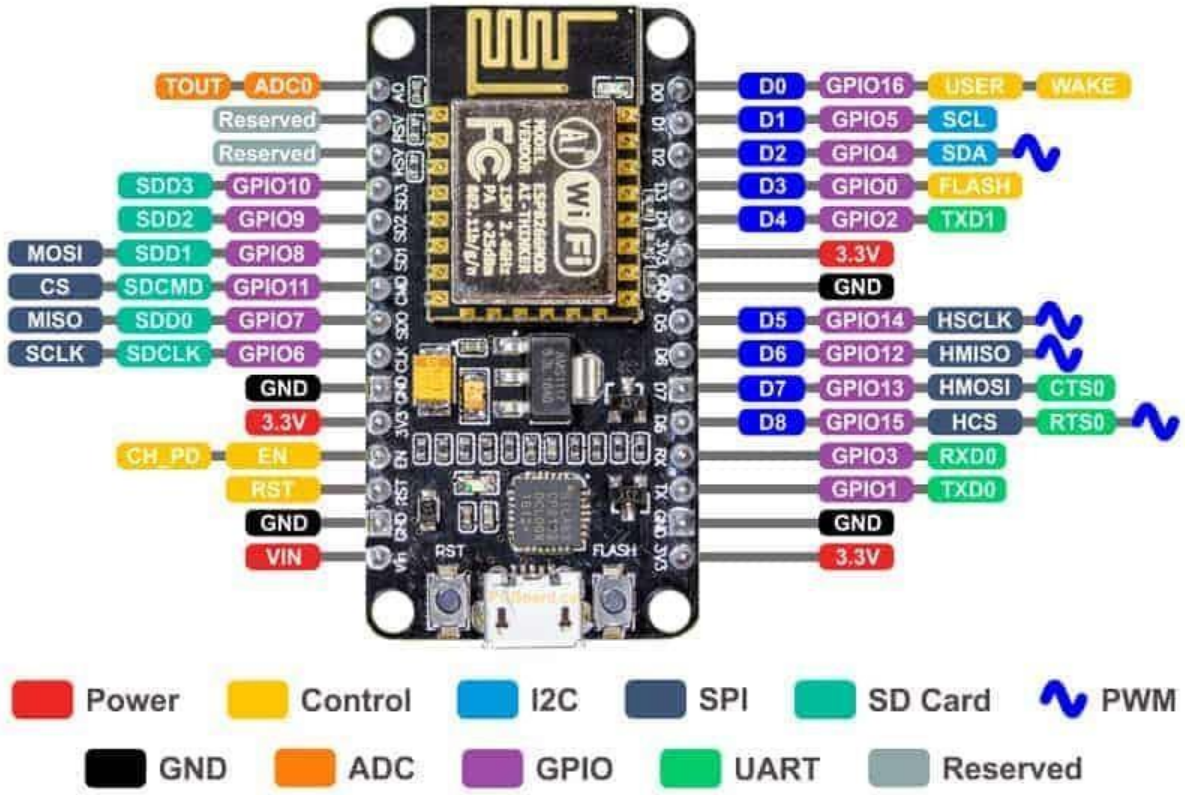

Figure-2.2 pin configuration of esp8266

2.4.1 POWER PINS

There are four power pins. VIN pin and three 3.3V pins.

i)VIN can be used to directly supply the NodeMCU/ESP8266 and its peripherals. Power delivered on VIN is regulated through the onboard regulator on the NodeMCU module – you can also supply 5V regulated to the VIN pin

ii)3.3V pins are the output of the onboard voltage regulator and can be used to supply power to external components.

2.4.2 I2C PIN

I2C PIN are used to connect I2C sensors and peripherals. Both I2C Master and I2C Slave are supported. I2C interface functionality can be realized programmatically, and the clock frequency is 100 kHz at a maximum. It should be noted that I2C clock frequency should be higher than the slowest clock frequency of the slave device.

2.4.3 GND

GND are the ground pins of NodeMCU/ESP8266.

2.4.4 GPIO PINS

NodeMCU/ESP8266 has 17 GPIO pins which can be assigned to functions such as I2C, I2S, UART, PWM, IR Remote Control, LED Light and Button programmatically. Each digital enabled GPIO can be configured to internal pull-up or pull-down, or set to high impedance. When configured as an input, it can also be set to edge-trigger or level-trigger to generate CPU interrupts

2.4.5 ADC CHANNEL

The NodeMCU is embedded with a 10-bit precision SAR ADC. The two functions can be implemented using ADC. Testing power supply voltage of VDD3P3 pin and testing input voltage of TOUT pin. However, they cannot be implemented at the same time**.**

2.4.6 UART

NodeMCU/ESP8266 has 2 UART interfaces (UART0 and UART1) which provide asynchronous communication (RS232 and RS485), and can

communicate at up to 4.5 Mbps. UART0 (TXD0, RXD0, RST0 & CTS0 pins) can be used for communication. However, UART1 (TXD1 pin) features only data transmit signal so, it is usually used for printing log.

2.4.7 SPI PINS

NodeMCU/ESP8266 features two SPIs (SPI and HSPI) in slave and master modes. These SPIs also support the following general-purpose SPI features:

i) 4 timing modes of the SPI format transfer

ii) Up to 80 MHz and the divided clocks of 80 MHz

iii) Up to 64-Byte FIFO

2.4.8 SDIO PIN

NodeMCU/ESP8266 features Secure Digital Input/Output Interface (SDIO) which is used to directly interface SD cards. 4-bit 25 MHz SDIO v1.1 and 4-bit 50 MHz SDIO v2.0 are supported.

2.4.9 PWM

The board has 4 channels of Pulse Width Modulation (PWM). The PWM output can be implemented programmatically and used for driving digital motors and LEDs. PWM frequency range is adjustable from 1000 μs to 10000 μs (100 Hz and 1 kHz).

2.4.10 CONTROL PINS

Control pins are used to control the NodeMCU/ESP8266. These pins include Chip Enable pin (EN), Reset pin (RST) and WAKE pin.

EN: The ESP8266 chip is enabled when EN pin is pulled HIGH. When pulled LOW the chip works at minimum power.

RST: RST pin is used to reset the ESP8266 chip.

WAKE: Wake pin is used to wake the chip from deep-sleep

2.5 LCD DISPLAY 16X2

FIGURE 2.3 LCD 16X2 DISPLAY

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.

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.

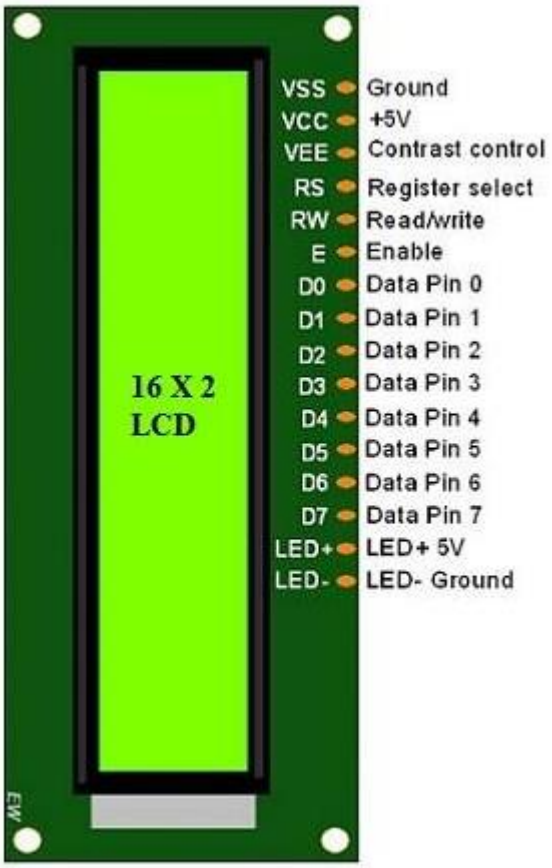

FIGURE 2.4 LCD PINOUTS

2.6 I2C MODULE

FIGURE 2.5 I2C MODULE

Instead of using multiple GPIO pins for data and control signals, the I2C module simplifies the connection to just two wires – SDA (data line) and SCL (clock line). Confirm the voltage level compatibility between the I2C module and your microcontroller (ESP8266). Most I2C modules for LCDs operate at 5V, so if your microcontroller operates at 3.3V (like the ESP8266), you may need level shifters to ensure proper communication. Utilize I2C libraries in your programming environment (Arduino IDE, for example) that support the I2C module for LCD. Popular libraries, such as the "LiquidCrystal_I2C" library for Arduino, simplify the code required for controlling the LCD

2.7 MG90S SERVO

FIGURE 2.6 MG90S SERVO

Metal Gears:

The inclusion of metal gears increases the durability and longevity of the MG90S. Metal gears are better equipped to handle stress and wear, making the servo suitable for applications where reliability is crucial.

Torque and Speed:

The MG90S offers a reasonable amount of torque given its small size. The exact specifications may vary slightly, but it typically provides torque in the range of 1.8 to 2.2 kg-cm. The speed of the servo is also noteworthy, responding quickly to input signals.

Voltage Range:

The operating voltage of the MG90S is commonly in the range of 4.8 to 6 volts. This makes it compatible with common power sources such as four-cell AA batteries or a dedicated servo controller.

2.7 IR MODULE

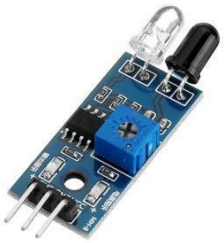

FIGURE 2.7 IR MODULE

The module features a 3 wire interface with Vcc, GND, and an OUTPUT pin on its tail. It works fine with 3.3 to 5V levels. Upon hindrance/reflectance, the output pin gives out a digital signal (a low-level signal). The onboard preset helps to fine-tune the range of operation, the effective distance range is 2cm to 80cm**.**

2.8 JUMPER WIRE

FIGURE 2.8 JUMPER WIRE

Jumper wires typically come in three versions: male-to-male, male-to-female and female-to-female. The difference between each is in the end point of the wire. Male ends have a pin protruding and can plug into things, while female ends do not and are used to plug things into. Male-to-male jumper wires are the most common and what you likely will use most often. When connecting two ports on a breadboard, a male-to-male wire is what you'll need.

2.9 IOT PLATFORM FOR CLOUD STORAGE

Adafruit IO is a system that makes data useful. Our focus is on ease of use, and allowing simple data connections with little programming required.

IO includes client libraries that wrap our REST and MQTT APIs. IO is built on Ruby on Rails, and Node.js.

io.adafruit.com to sign up.

2.10 ARDUINO IDE COMPILER

Arduino designs, manufactures, and supports electronic devices and software, allowing people around the world to easily access advanced

technologies that interact with the physical world. Our products are straightforward, simple, and powerful, ready to satisfy users' needs from students to makers and all the way to professional developers

Install ESP8266 Board Support Package:

1) Open Arduino IDE:

2) Open the Arduino IDE on your computer.

Go to Preferences:

Navigate to "File" > "Preferences."

Add Additional Boards Manager URLs:

In the "Additional Boards Manager URLs" field, add the following URL for the ESP8266 board support package:

bash

http://arduino.esp8266.com/stable/package_esp8266com_index.json

If you already have URLs in the field, separate them with a comma.

Click "OK":

Click "OK" to save the preferences.

Open Boards Manager:

Navigate to "Tools" > "Board" > "Boards Manager."

Search for ESP8266:

In the Boards Manager, type "ESP8266" into the search bar.

Install ESP8266 Board Support Package:

Find "esp8266 by ESP8266 Community" and click the "Install" button. Select ESP8266 Board:

After installation, go to "Tools" > "Board" and select the specific ESP8266 board you are using (e.g., NodeMCU, Wemos D1 Mini).

2.10.1 LIBRARIES USED

1) #include <ESP8266WiFi.h>:

This line tells the compiler to include the library for Wi-Fi functionality on the ESP8266 board. This library provides functions for connecting to Wi-Fi networks and managing Wi-Fi communication.

2) #include "Adafruit_MQTT.h":

This line includes the Adafruit MQTT library, which allows the device to communicate with Adafruit IO's MQTT server. MQTT is a lightweight messaging protocol used for sending and receiving data between devices.

3) #include "Adafruit_MQTT_Client.h":

This line includes a helper library for interacting with the MQTT client within the Adafruit MQTT library. It simplifies connecting, subscribing, and publishing messages.

4) #include <Servo.h>:

This line includes the Servo library, which allows you to control servo motors connected to your ESP8266. It provides functions for setting their position, rotating them, and reading their current position.

5) #include <Wire.h>:

This line includes the Wire library, which enables I2C communication. I2C is a two-wire interface used for connecting various devices over short distances. In this case, it's used to communicate with the Liquid Crystal display.

6) #include <LiquidCrystal_I2C.h>:

This line includes the LiquidCrystal_I2C library, which allows you to control Liquid Crystal displays connected via I2C. It provides functions for writing text, clearing the display, and setting the backlight.

2.11 ARDUINO NANO

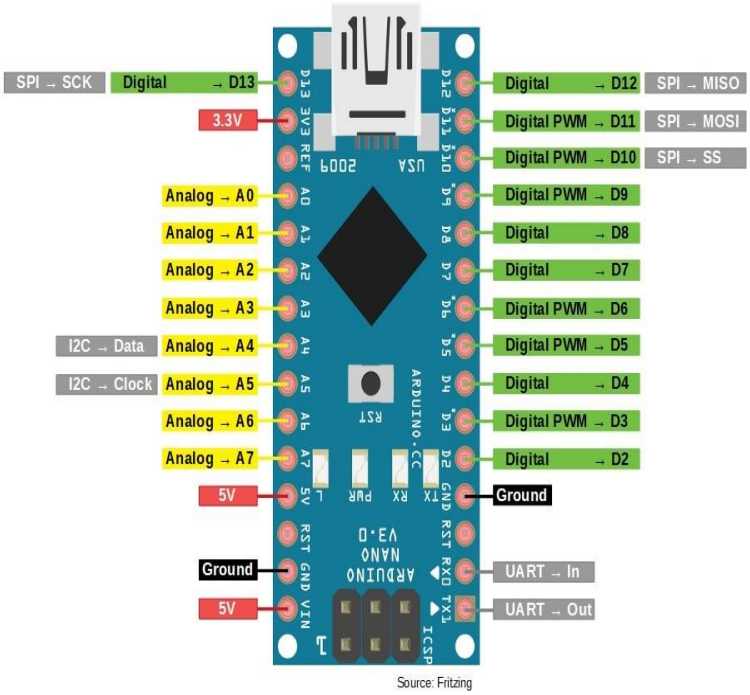

Fig 2.9 ARDUINO NANO

1) Power Pin (Vin, 3.3V, 5V, GND):

These pins are power pins

i)Vin is the input voltage of the board, and it is used when an external power sourceis used from 7V to 12V.

ii)5V is the regulated power supplyvoltage of the nano board and it is used to give the supply to the board as well as components.

iii)3V is the minimum voltage which is generated from thevoltage regulator on the board.

2) GND is the ground pin of the board

3) RST Pin (Reset): This pin is used to reset the microcontroller

4) Analog Pins (A0-A7): These pins are used to calculate the analog voltage of the board within the range of 0V to 5V

5) I/O Pins (Digital Pins from D0 – D13): These pins are used as an i/p otherwise o/p pins. 0V & 5V

6) Serial Pins (Tx, Rx): These pins are used to transmit & receive TTL serial data.

7) External Interrupts (2, 3): These pins are used to activate an interrupt.

8) PWM (3, 5, 6, 9, 11): These pins are used to provide 8-bit of PWM output.

9) SPI (10, 11, 12, & 13): These pins are used for supporting SPI communication.

10)Inbuilt LED (13): This pin is used to activate the LED.
11)IIC (A4, A5): These pins are used for supporting TWI communication.
12)AREF: This pin is used to give reference voltage to the input voltage

2.12 SERVO SG90

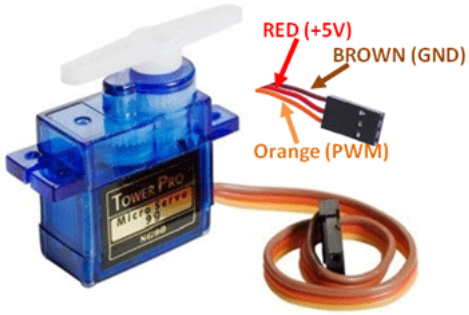

Fig 2.10 servo sg90

1)Brown wire-Ground wire connected to the ground of system

2)Red wire-Powers the motor typically +5V is used

3)Orange wire-PWM signal is given in through this wire to drive the motor

2.13 SOURCE CODE

2.13.1 ESP8266

carparking.ino BlynkSimpleEsp8266.h

```
#include <ESP8266WiFi.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
#include <Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,20,4);
Servo myservo;
int pos;
/* WiFi Access Point ***/

#define WLAN_SSID       "TRIANGLES3.0"
#define WLAN_PASS       "123456789"

/* Adafruit.io Setup ***/

#define AIO_SERVER      "io.adafruit.com"
#define AIO_SERVERPORT  1883                   // use 8883 for SSL
#define AIO_USERNAME    "raghuln5315"
#define AIO_KEY         "aio_nubG10oZdeDwR8mfzfr2nwGByrxP"

/* Global State (you don't need to change this!) */

// Create an ESP8266 WiFiClient class to connect to the MQTT server.
WiFiClient client;
// or... use WiFiClientSecure for SSL
//WiFiClientSecure client;

// Setup the MQTT client class by passing in the WiFi client and MQTT server and login details.
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);

/** Feeds ***/

// Setup a feed called 'photocell' for publishing.
// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>
Adafruit_MQTT_Publish photocell1 = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/b");
Adafruit_MQTT_Publish photocell2 = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/bu");
Adafruit_MQTT_Publish photocell3 = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/c");
Adafruit_MQTT_Publish photocell4 = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/c1");
Adafruit_MQTT_Publish photocell5 = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/a");

// Setup a feed called 'onoff' for subscribing to changes.
Adafruit_MQTT_Subscribe onoffbutton = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/onoff");

/* Sketch Code **/

// Bug workaround for Arduino 1.6.6, it seems to need a function declaration
// for some reason (only affects ESP8266, likely an arduino-builder bug).
void MQTT_connect();

```

```
51  void setup() {
52    Serial.begin(115200);
53    lcd.init();
54    lcd.backlight();
55    delay(10);
56    myservo.attach(D0);
57    Serial.println(F("Adafruit MQTT demo"));
58
59    // Connect to WiFi access point.
60    Serial.println(); Serial.println();
61    Serial.print("Connecting to ");
62    Serial.println(WLAN_SSID);
63
64    WiFi.begin(WLAN_SSID, WLAN_PASS);
65    while (WiFi.status() != WL_CONNECTED) {
66      delay(500);
67      Serial.print(".");
68    }
69    Serial.println();
70
71    Serial.println("WiFi connected");
72    Serial.println("IP address: "); Serial.println(WiFi.localIP());
73
74    // Setup MQTT subscription for onoff feed.
75    mqtt.subscribe(&onoffbutton);
76  }
77
78  uint32_t x=0;
79
80  void loop() {
81    MQTT_connect();
82    lcd.clear();
83    lcd.setCursor(0,0);
84    lcd.print("S1-");
85
86    lcd.setCursor(6,0);
87    lcd.print("S2-");
88
89    lcd.setCursor(11,0);
90    lcd.print("S3-");
91
92    lcd.setCursor(0,1);
93    lcd.print("S4-");
94
95    lcd.setCursor(6,1);
96    lcd.print("S5-");
97    if(analogRead(A0)<250)
98    {
99      lcd.setCursor(3,0);
100     lcd.print("1");
```

```
101         if (! photocell1.publish(0)) {
102           Serial.println(F("Failed"));
103         } else {
104           Serial.println(F("OK!"));
105         }
106         delay(2000);
107       }
108
109       if(digitalRead(D5)==LOW)
110       {
111         lcd.setCursor(9,0);
112         lcd.print("1");
113         if (! photocell2.publish(0)) {
114           Serial.println(F("Failed"));
115         } else {
116           Serial.println(F("OK!"));
117         }
118         delay(2000);
119       }
120
121       if(digitalRead(D6)==LOW)
122       {
123         lcd.setCursor(14,0);
124         lcd.print("1");
125         if (! photocell3.publish(0)) {
126           Serial.println(F("Failed"));
127         } else {
128           Serial.println(F("OK!"));
129         }
130         delay(2000);
131       }
132
133       if(digitalRead(D7)==LOW)
134       {
135         lcd.setCursor(3,1);
136         lcd.print("1");
137         if (! photocell4.publish(0)) {
138           Serial.println(F("Failed"));
139         } else {
140           Serial.println(F("OK!"));
141         }
142         delay(2000);
143       }
144
145       if(digitalRead(D8)==LOW)
146       {
147         lcd.setCursor(9,1);
148         lcd.print("1");
149         if (! photocell5.publish(0)) {
150           Serial.println(F("Failed"));
```

```
    } else {
      Serial.println(F("OK!"));
    }
    delay(2000);
  }

  if(digitalRead(D6)==HIGH)
  {
    lcd.setCursor(14,0);
    lcd.print("0");
    if (! photocell3.publish(1)) {
      Serial.println(F("Failed"));
    } else {
      Serial.println(F("OK!"));
    }
    delay(2000);
  }

  if(digitalRead(D8)==HIGH)
  {
    lcd.setCursor(3,1);
    lcd.print("0");
    if (! photocell4.publish(1)) {
      Serial.println(F("Failed"));
    } else {
      Serial.println(F("OK!"));
    }
    delay(2000);
  }

  if(digitalRead(D8)==HIGH)
  {
    lcd.setCursor(9,1);
    lcd.print("0");
    if (! photocell5.publish(1)) {
      Serial.println(F("Failed"));
    } else {
      Serial.println(F("OK!"));
    }
    delay(2000);
  }

//  if(digitalRead(D3)==LOW)
//  {
//    for (pos = 0; pos <= 180; pos += 1) {  // goes from 0 degrees to 180 degrees
//      // in steps of 1 degree
//      myservo.write(pos);  // tell servo to go to position in variable 'pos'
//      delay(15);           // waits 15ms for the servo to reach the position
//    }
//    delay(4000);
```

```
//    for (pos = 180; pos >= 0; pos -= 1) {  // goes from 180 degrees to 0 degrees
//      myservo.write(pos);                  // tell servo to go to position in variable 'pos'
//      delay(15);                           // waits 15ms for the servo to reach the position
//    }
//  }
  while(digitalRead(D8)==LOW&&digitalRead(D8)==LOW&&digitalRead(D6)==LOW&&digitalRead(D5)==LOW&&analogRead(A0)<250)
  {
    myservo.write(0);
    delay(1000);
    photocell1.publish(0);
    delay(2000);
    photocell2.publish(0);
    delay(2000);
    photocell3.publish(0);
    delay(2000);
    photocell4.publish(0);
    delay(2000);
    photocell5.publish(0);
    delay(2000);
  }
  if(digitalRead(D4)==LOW)
  {
    for (pos = 0; pos <= 180; pos += 1) {  // goes from 0 degrees to 180 degrees
      // in steps of 1 degree
      myservo.write(pos);  // tell servo to go to position in variable 'pos'
      delay(15);           // waits 15ms for the servo to reach the position
    }
    delay(4000);
    for (pos = 180; pos >= 0; pos -= 1) {  // goes from 180 degrees to 0 degrees
      myservo.write(pos);                  // tell servo to go to position in variable 'pos'
      delay(15);                           // waits 15ms for the servo to reach the position
    }
  }
}

// Function to connect and reconnect as necessary to the MQTT server.
// Should be called in the loop function and it will take care if connecting.
void MQTT_connect() {
  int8_t ret;

  // Stop if already connected.
  if (mqtt.connected()) {
    return;
  }

  Serial.print("Connecting to MQTT... ");

  uint8_t retries = 3;
  while ((ret = mqtt.connect()) != 0) { // connect will return 0 for connected
       Serial.println(mqtt.connectErrorString(ret));
```

```
276         Serial.println("Retrying MQTT connection in 5 seconds...");
277         mqtt.disconnect();
278         delay(5000);  // wait 5 seconds
279         retries--;
280         if (retries == 0) {
281           // basically die and wait for WDT to reset me
282           while (1);
283         }
284       }
285       Serial.println("MQTT Connected!");
286     }
287
```

2.13.2 ARDUINO NANO CODE

```
if(digitalRead(D3)==LOW)
{
  for (pos = 0; pos <= 180; pos += 1) {  // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
   myservo.write(pos);  // tell servo to go to position in variable 'pos'
    delay(15);          // waits 15ms for the servo to reach the position
  }
  delay(4000);
  for (pos = 180; pos >= 0; pos -= 1) {  // goes from 180 degrees to 0 degrees
    myservo.write(pos);                  // tell servo to go to position in variable 'pos'
    delay(15);                           // waits 15ms for the servo to reach the position
  }
}
```

CHAPTER 3

REFERNCE LINKS AND DIFFICULTIES

3.1 REFERNCE LINKS

- https://iotdesignpro.com/projects/iot-based-smart-parking-using-esp8266
- https://www.instructables.com/IoT-Based-Smart-Parking-System-Using-NodeMCU-ESP82/

3.2 DIFFICULTIES

The ESP8266 itself operates at 3.3V, and it doesn't have a 5V output. However, there are a few common methods to obtain 5V for powering external components, such as a servo motor:

USB Power:

If you are powering your ESP8266 through a USB connection, you can tap into the 5V available from the USB source. Many USB chargers and power banks provide a 5V output so can be used for lcd and servo

https://copyprogramming.com/howto/get-5v-output-from-nodemcu-duplicate

CHAPTER 4

RESULT

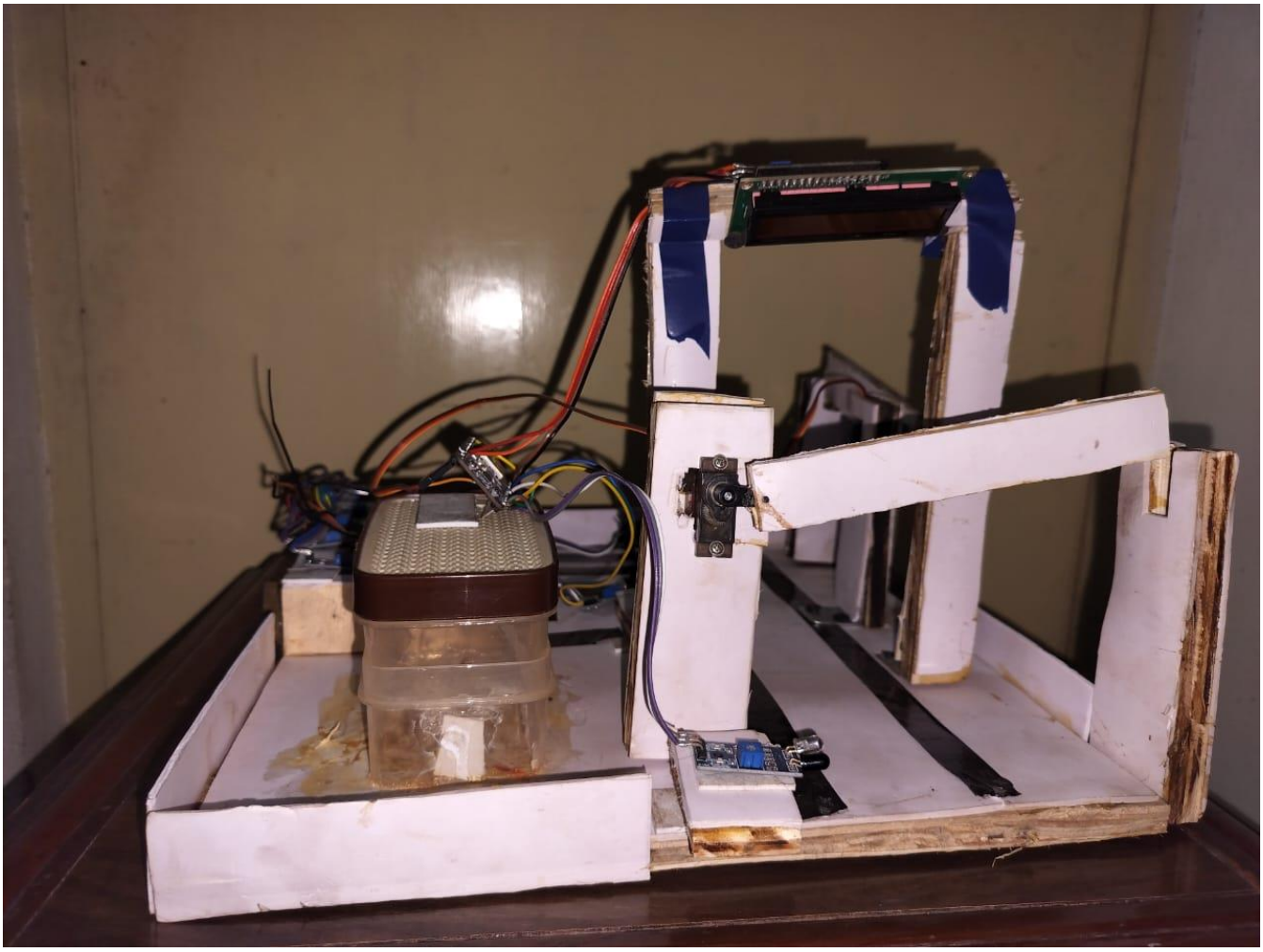

Fig 4.1 : Project Setup

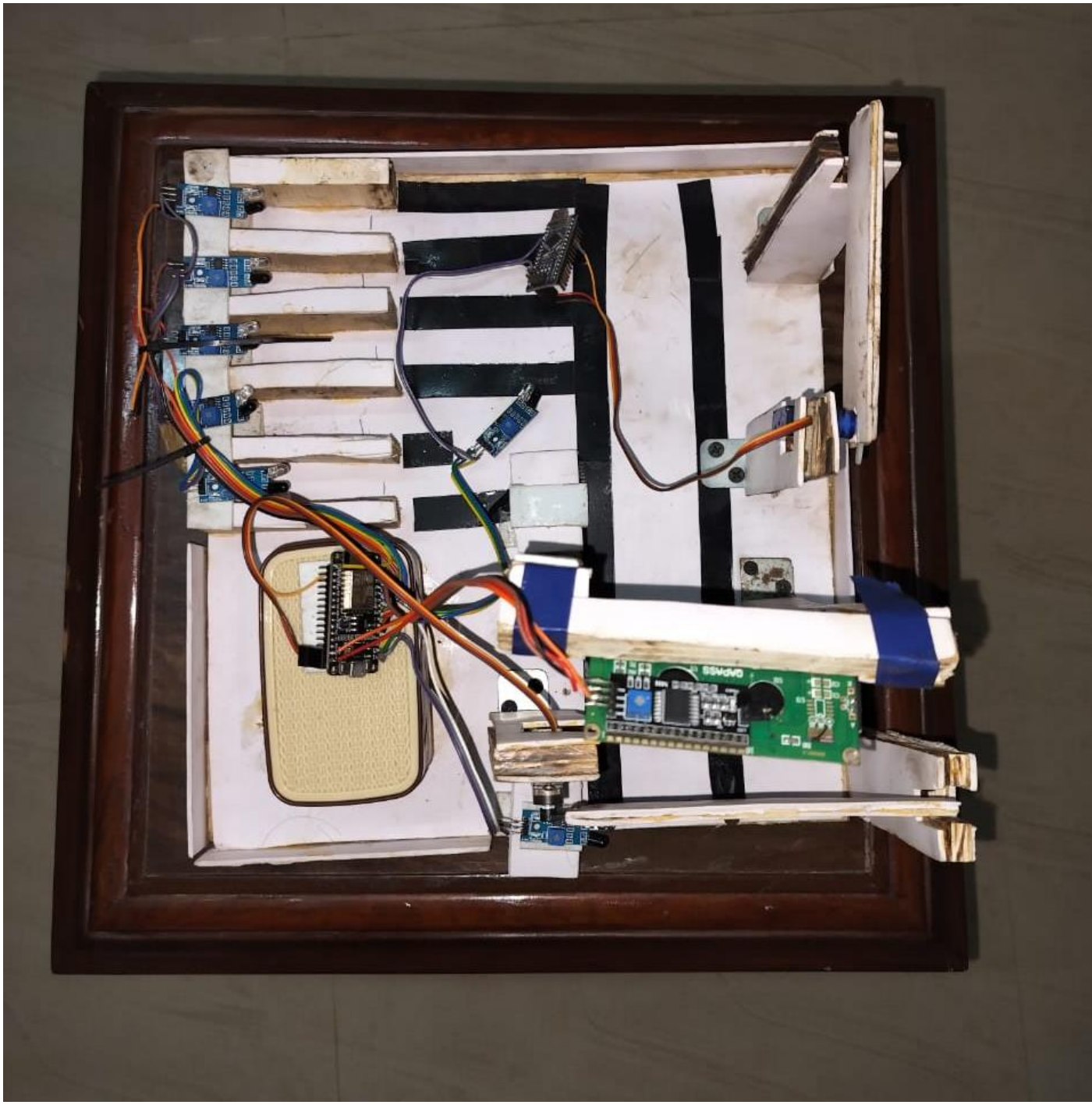

Fig 4.2 : Project Setup 2

Fig 4.3: Adafruit IO data

.