Smart Parking System

A PROJECT REPORT

Submitted by

Amit Mohan Rajput- University Roll No: 2100290140024
Dhaval Krishna- University Roll No: 2100290140059
Nikhil Agarwal- University Roll No: 2100290140095

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION
Session: 2022-23(4th Semester)

Under the supervision of
Ms. Shweta Singh
Assistant Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET GROUP OF INSTITUTIONS, GHAZIABAD
Uttar Pradesh-201206
(June 2023)

DECLARATION

I hereby declare that the work presented in this report entitled “Smart Parking System", was carried out by me. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute. I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources. I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

Name : Amit Mohan Rajput	Name: Dhaval Krishna
Roll. No. : 2100290140024	Roll.No. 2100290140059
Branch: MCA	Branch: MCA
(Candidate Signature)	(Candidate Signature)

Name : Nikhil Agarwal

Roll. No. : 2100290140095

Branch: MCA

(Candidate Signature)

CERTIFICATE

Certified that **Amit Mohan Rajput- 2100290140024**, **Dhaval Krishna-2100290140059**, **Nikhil Agarwal- 2100290140095** have carried out the project work having **"Smart Parking System"** for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University (AKTU), Lucknow under my supervision. The project report embodies original work and studies are carried out by the students himself and the contents of the project report do not form the basis for the award of any other degree to the candidate of to anybody else from this or any other University/Institution.

Date:

Amit Mohan Rajput (2100290140024)

Dhaval Krishna (2100290140059)

Nikhil Agarwal (2100290140095)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: **Ms. Shweta Singh**

(Assistant Professor)
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Name of Internal Examiner **Signature of External Examiner**

Dr. Arun Kumar Tripathi
Head, Department of Computer Applications
KIET Group of Institutions, Ghaziabad

ABSTRACT

The increasing urbanization and the growing number of vehicles have led to significant challenges in managing parking spaces efficiently. To address this issue, the implementation of a smart parking system using the Internet of Things (IoT) has emerged as a promising solution. This abstract explores the concept, benefits, and key components of such a system.

The smart parking system utilizes IoT technology to connect parking spaces, sensors, and a central control system. The sensors installed in parking lots or on-street parking spaces detect the presence of vehicles and transmit real-time data to the control system. This data includes information on parking availability, occupancy status, and duration of parking.

The central control system processes the received data and provides valuable insights to both parking facility managers and users. Through mobile applications or digital signage, users can access up-to-date information about parking availability, reserve spaces, or make payments. Parking facility managers can monitor parking patterns, optimize space allocation, and implement dynamic pricing strategies.

The benefits of a smart parking system using IoT are manifold. It improves the overall parking experience for users by reducing the time spent searching for parking spaces, thus alleviating traffic congestion. The system enhances operational efficiency by providing accurate data for better space management, leading to optimized resource utilization and increased revenue generation.

Key components of the system include sensors for vehicle detection, communication networks for data transmission, a central control system for data processing and analysis, and user interfaces for accessing information and making reservations or payments.

However, the successful implementation of a smart parking system using IoT requires careful consideration of factors such as infrastructure requirements, connectivity challenges, security, and privacy concerns. Integration with existing parking infrastructure and systems is also a crucial aspect.

ACKNOWLEDGEMENT

Success in life is never attained single handedly. My deepest gratitude goes to my thesis supervisor, **Ms. Shweta Singh** for her guidance, help and encouragement throughout my research work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Arun Kumar Tripathi**, **Professor and Head, Department of Computer Applications**, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Amit Mohan Rajput(2100290140024)

Dhaval Krishna(2100290140059)

Nikhil Agarwal(2100290140095)

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Smart parking system using IoT has sensors added into an interconnected system to determine parking space or level and provide real-time feedback. It is accomplished by using cameras, counters on the doors or gates of parking lots, sensors embedded in the paved area of individual parking lots, among other locations, depending on the deployment.

Each parking space has an IoT gadget, which includes sensors and microcontrollers. The user gets real-time updates on the availability of all parking spaces and, therefore, an option to choose the best one. This solution alone initiates a chain-reaction of benefits, from lesser traffic congestion to reduced fuel efficiency, in urban areas where parking is often painstaking.

A simple and easy task such as parking is thought as a tedious and time-consuming process due to mismanagement of parking system. Current parking systems involve huge manpower for management and requires user to search for parking space floor by floor. Such conventional systems utilize more power, along with user's valuable time. This paper presents a Smart Parking Energy Management solution for a structured environment such as a multi-storied office parking area.

The system proposes implementation of state-of-the-art Internet of Things (IoT) technology to mold with advanced Honeywell sensors and controllers to obtain a systematic parking system for users. Unoccupied vehicle parking spaces are indicated using lamps and users are guided to an empty parking space, thus eliminating need for searching for a parking space.

1.2 AIM

Smart Parking system are designed to provide drivers an ultimate solution on their journey from the beginning to end without searching for parking, cost, travel time etc. This advantage comes by paying marginal fees to the smart parking service providers.
A Smart parking technology that will help optimize parking space usage, improve the efficiency of the parking operations and help smoother traffic flow.

1.3 EXISTING SYSTEM

The smart car parking system works on the simple principle of detecting obstacle and sending a visual feedback. The proximity sensor is mounted on the ceiling of the parking lot which consists of an Infra-Red emitter and a receiver. The IR emitter emits infra-red rays and these rays generally bounce off objects. The IR receiver receives these rays and converts them into an electrical signal creating a potential difference. The resulting potential difference helps complete the circuit. The LEDs are placed along the driveway and switch on based on the input received by the sensor. A threshold distance is calibrated using the potentiometer to fix a particular distance based on the average height of vehicles for sending and receiving the radiations. Resistors are provided to ensure the safe working of LEDs and IR sensors. For this project based on size a 12V battery is used to power all the components.

1.3.1 LITERATURE REVIEW

Magnetometer sensors and Bluetooth beacons. In [1], the proposed system addresses the issues of scalability, interoperability, low energy consumption, and timely prediction of parking spot availability. The system uses magnetometer sensors and concentrators to collect data on parking spot occupancy, and a control dashboard provides data analytics and management of the monitored parking areas. The system also includes a smart payment service that allows users to pay for parking services automatically through Bluetooth beacons. Experiments conducted on a test-bed demonstrate the system's ability to detect the presence of a vehicle and trigger the payment procedure. The use of virtual entities in the SIoT environment enables interoperability among different types of IoT devices used by separate solutions deployed in adjacent areas, thereby addressing the heterogeneity of IoT devices. Overall, the proposed solution provides an efficient and effective smart parking system for on-street parking areas.

This paper discusses the importance of smart parking systems in addressing the issue of parking scarcity in overcrowded cities. In [2], The authors classify smart parking systems based on soft and hard design factors and provide an overview of enabling technologies and sensors commonly used in the literature. They emphasize the importance of data reliability, security, privacy, and other critical design factors in such systems. The paper also discusses emerging parking trends in the ecosystem, focusing on data interoperability and exchange. The authors outline open research issues in the current state of smart parking systems and recommend a conceptual hybrid-parking model. The paper presents software aspects of smart parking, including parking prediction, path optimization, and parking assignment. Tables are generated to compare several key factors of the main elements in a smart parking system, including sensors and communication modules. The paper concludes by suggesting cloud-based hybrid models to solve key issues in smart parking applications and proposing future research on interoperability and drones.

This paper discusses the design and development of a wireless sensor node for a smart parking system using IoT. In [3], The system uses various sensors to detect parking slot availability, and the information is uploaded to the cloud, which can be accessed by users using an Android app. The sensor node is connected with other sensor nodes to create a heterogeneous wireless sensor network, with one node acting as the master node and the rest as slave nodes. The QoS parameters are observed and calculated to determine the efficiency of the network.

Smart cities aim to incorporate technology to resolve daily issues such as house management, healthcare, and traffic. In [4], Parking is a significant issue in cities, especially during peak hours, and despite various traditional approaches and technologies, flaws still exist. Many proposed solutions have limitations and cost constraints. This paper proposes an optimized and cost-effective framework for parking solutions in smart cities. The system uses Arduino microcontroller and sensors to provide accurate data, which is communicated to the central server and processed by the web application. Users can check availability, reserve parking slots in advance, and additional features such as payment and security services can be added. The proposed framework offers a simple, low-cost solution to the parking problem and can be further improved with additional features.

This paper proposes an IoT-based smart parking system designed to regulate the number of vehicles parked in designated areas, with the aim of reducing the human work and making it easier and more secure for users to park their vehicles. In [5], The system uses vehicle number plates to authenticate and reserve parking spaces, and an Android application to locate the nearest parking lots, check real-time space availability, and pay for parking slots. The system employs Raspberry Pi for data transfer and sensor data collection, and uses Python and Java programming languages for codes of Deep learning and Android application, respectively. The system is secure and works quickly as the K Nearest Neighbour algorithm is used with OpenCV to recognize number plates. The proposed system is recommended for commercial purposes, and future extensions could include multilevel and multiple parking areas.

Traffic congestion and unavailability of free parking spaces are major urban transport problems that cause high energy consumption and air pollution. In [6], In this context, the development of IoT-based smart parking information systems is one of the most demanded research problems for sustainable smart cities. This paper proposes a deep long short term memory network-based framework to predict the availability of parking spaces using the Birmingham parking sensors dataset. The proposed model outperforms the state-of-the-art prediction models and predicts both the overall availability of parking space and location-wise availability of parking space. The location wise, day-wise, and hour-wise car parking availability prediction gives better insight to the drivers to choose their route and destination. However, the decision support system developed in this study considers only the parking occupancy information and not the weather condition and social events.

This study proposes a new mobile smart parking application that utilizes deep learning and cloud-based architecture to minimize the problem of searching for parking spaces in cities. In [7], The application uses a Long short-term memory (LSTM) model to predict parking space occupancy rates, and it allows dynamic access to the model through the user's mobile device. The real-time car parking data collected in Istanbul, Turkey, was used to test the accuracy of the model. The proposed model was compared to other methods, such as Support Vector Machine, Random Forest, and ARIMA, and it yielded high accuracy results. The study suggests that deep learning-based models can be used to predict parking space occupancy rates in cities and reduce energy consumption, environmental pollution, and stress caused by searching for parking spaces.

This paper discusses the problems and inefficiencies associated with traditional parking lots, which lead to inconvenience and wastage of resources. In [8], The authors propose a solution using an IoT-based smart parking system that allows users to find vacant parking slots in real-time using a mobile app, IR sensors, RFID, and Arduino. The system also enables users to pre-book and pay for parking slots, reducing uncertainty and ambiguity. Upon arrival, users authenticate themselves using RFID tags, and payments are processed automatically using the in-app wallet. The system improves efficiency, reliability, and convenience, reducing the time and resources spent searching for parking spaces and pollution. The paper highlights the advantages of using RFID over other identification methods and presents the limitations of the proposed system, such as the need for constant internet connectivity and the requirement for first-time users to collect RFID tags. The system's potential applications include commercial parking spaces and multi-story parking lots.

The rapid increase in population and urbanization has led to a significant increase in the number of vehicles on the roads, which in turn has caused parking and traffic problems. In [9], To address this issue, an automatic real-time system for vehicle parking has been proposed in this paper. The system is based on the Internet of Things (IoT) and uses Arduino Uno as a microcontroller to communicate with the physical devices. The system incorporates an infrared sensor in each parking slot to detect the availability of the slot, and users can book their parking slots in advance through a website. The proposed system also includes security measures such as exclusive usernames and passwords for users and alerts for any misuse of the system. The integration of smart parking with IoT cloud technology is expected to improve the standard of living for consumers by providing an effective parking system that ensures smooth vehicle traffic.

This paper presents a solution for a structured environment such as a multi-storied office parking area by proposing a Smart Parking & Energy Management system that utilizes IoT technology along with Honeywell sensors and controllers. In [10], The system is designed to eliminate the need for manpower for parking management and reduce user's valuable time. The unoccupied parking spaces are indicated using lamps, and the central system directs the upcoming cars to empty spaces. The system also features automatically controlled light illuminance to reduce energy usage and improve the aesthetics of the parking area. The proposed system can be implemented in hospitals, shopping malls, and apartments. The system aims to reduce challenges faced in conventional parking systems such as space, time, labour cost, and productivity while also ensuring privacy and security. The paper emphasizes the importance of IoT-enabled smart parking systems in improving parking management and enhancing user convenience.

This study proposes an IoT-based Intelligent Parking System (IPS) to address the issue of finding parking spaces in smart cities. In [11], The IPS collects real-time data through sensors and sends it to the cloud, enabling users to check nearby parking availability and reserve a spot using a mobile application. The system is implemented using Raspberry Pi, NodeMCU, RFID, and IR sensors, and the results demonstrate its usability and efficiency. The study also suggests adding route direction and alarm generation features to enhance the IPS further. The proposed IPS can help solve the problem of finding parking spaces in densely populated areas, making it a valuable addition to the infrastructure of smart cities.

Signal strength of LoRaWAN smart parking devices, which can be exploited by attackers to estimate parking lot occupancy. In [12], The study uses supervised machine learning techniques based on neural networks and random forest approach to estimate parking lot occupancy with high accuracy. The research highlights the privacy implications of LoRaWAN smart parking systems and suggests countermeasures to enhance their security, such as implementing encryption and noise generation mechanisms. The study is significant for smart city development and emphasizes the need to consider privacy and security concerns when implementing IoT technologies.

Smart parking systems are essential for the development of IoT in Smart Cities, as they reduce traffic congestion and gas emissions while improving the quality of life of residents. In [13], This paper presents a comprehensive overview of technologies used for smart parking detection, with a focus on low-power solutions. The paper analyzes the performance of various sensor devices, including photodiodes, LDRs, infrared LEDs, ultrasonic sensors, LiDAR, and magnetic sensors, and concludes that the 3-axis magnetic sensor is the most suitable for vehicle detection. In addition, the paper compares the performance of LPWA radio communication technologies such as LoRa, NB-IoT, and Sigfox, and shows that LoRa is the most efficient in terms of power consumption. Finally, the paper proposes two strategies for optimizing smart parking systems, including using a drop in signal strength to indicate parking space occupancy and a novel device that harvests surrounding energy to reduce battery consumption. These solutions offer great potential for improving the efficiency and sustainability of future smart parking systems.

The paper proposes an IoT-based smart parking system controlled by a smartphone device that provides a reliable parking solution for both commuters and parking lot owners. In [14], It utilizes IR sensors to determine the availability of a parking space and communicates the information to the server via a Wi-Fi module and remote communication technology. An RFID tag attached to a vehicle is used to verify the identity of the user who occupies a regular, daily, weekly, or monthly parking spot. The system also uses a scheduling algorithm to determine the nearest available parking space based on the size of the vehicle. The mobile app offers a user-friendly interface, and the parking lot owner can monitor the number of free and occupied parking spaces, adjust variable parking fees, and analyze daily, weekly, and monthly revenue. The proposed system is expected to improve efficiency, reduce traffic congestion, and enable users to book parking spaces remotely in the future.

This paper proposes a solution to the time-consuming process of parking management by presenting a smart parking system that utilizes IoT technology and Honeywell sensors and controllers. In [15], The system provides users with a systematic parking experience by indicating unoccupied parking slots using LEDs and storing occupied slots in the cloud for central systems to direct incoming cars. The fully automatic system aims to improve time, value, and convenience in a parking area, while reducing the need for manpower and improving parking illuminance. The paper also highlights the growth of IoT in making smart cities a reality and how the proposed system addresses the issue of parking by providing real-time information on parking spot availability and allowing users to book a slot through a web application. Overall, the system aims to enhance the quality of life for people in urban environments.

1.4 PROPOSED SYSTEM

Our proposed smart parking system aims to revolutionize the parking experience by leveraging advanced technology and automation. The system incorporates various components to streamline parking operations, enhance efficiency, and optimize space utilization.

Firstly, we will deploy a network of sensors and cameras throughout the parking area to monitor occupancy in real-time. This data will be transmitted to a central control system, which will analyze and manage parking availability information. Users can access this information through a dedicated mobile application or digital signage, allowing them to locate and reserve parking spaces in advance.

To facilitate seamless entry and exit, we will implement automated barriers and license plate recognition systems. This will eliminate the need for manual ticketing and reduce congestion at entry and exit points.

Furthermore, the system will incorporate payment integration, enabling users to make cashless transactions through the mobile app or automated payment kiosks.

To ensure security, CCTV cameras and alarm systems will be installed to monitor the parking area.

Overall, our smart parking system promises to optimize parking operations, reduce traffic congestion, enhance user convenience, and improve overall efficiency in urban parking scenarios.

1.5 HARDWARE & SOFTWARE REQUIREMENTS SPECIFICATION

1.5.1 Hardware Requirements

Table 1.1 Hardware Requirements

Number	**Description**
1	Node MCU (1)
2	IR Sensors (5)
3	Servo Motors (2)

1.5.1.1 NodeMCU MicroController

NodeMCU is an open source firmware for which open source prototyping board designs are available. The name "NodeMCU" combines "node" and "MCU" (micro-controller unit). Strictly speaking, the term "NodeMCU" refers to the firmware rather than the associated development kits.

Both the firmware and prototyping board designs are open source.

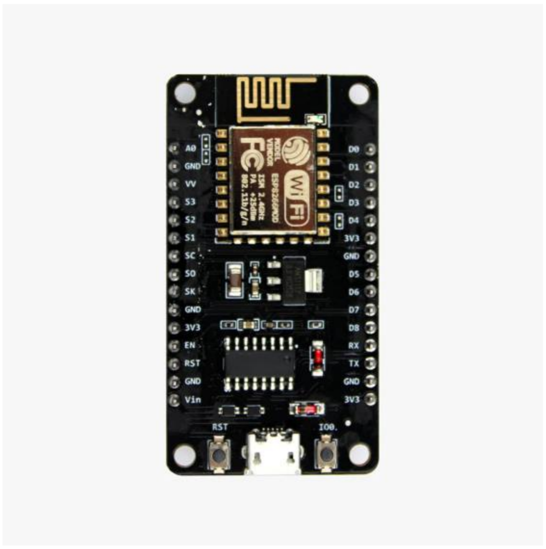


Figure 1.1 NodeMCU MicroController

The firmware uses the Lua scripting language. The firmware is based on the eLua project, and built on the Espressif Non-OS SDK for ESP8266. It uses many open source projects, such as lua-cjson and SPIFFS. Due to resource constraints, users need to select the modules relevant for their project and build a firmware tailored to their needs. Support for the 32-bit ESP32 has also been implemented.

The prototyping hardware typically used is a circuit board functioning as a dual in-line package (DIP) which integrates a USB controller with a smaller surface-mounted board containing the MCU and antenna. The choice of the DIP format allows for easy prototyping on breadboards. The design was initially based on the ESP-12 module of the ESP8266, which is a Wi-Fi SoC integrated with a Tensilica Xtensa LX106 core, widely used in IoT applications.

There are two available versions of NodeMCU as version 0.9 & 1.0 where the version 0.9 contains ESP-12 and version 1.0 contains ESP-12E where E stands for "Enhanced". We have used version 1.0 ESP-12E in this project.

NodeMCU was created shortly after the ESP8266 came out. On December 30, 2013, Espressif Systems began production of the ESP8266. NodeMCU started on 13 Oct 2014, when Hong committed the first file of nodemcu-firmware to GitHub. Two months later, the project expanded to include an open-hardware platform when developer Huang R committed the gerber file of an ESP8266 board, named devkit v0.9. Later that month, Tuan PM ported MQTT client library from Contiki to the ESP8266 SoC platform, and committed to NodeMCU project, then NodeMCU was able to support the MQTT IoT protocol, using Lua to access the MQTT broker. Another important update was made on 30 Jan 2015, when Devsaurus ported the u8glib to the NodeMCU project, enabling NodeMCU to easily drive LCD, Screen, OLED, even VGA displays.

In the summer of 2015 the original creators abandoned the firmware project and a group of independent contributors took over. By the summer of 2016 the NodeMCU included more than 40 different modules.

1.5.1.1.1 ESP8266 Arduino Core

As Arduino.cc began developing new MCU boards based on non-AVR processors like the ARM/SAM MCU used in the Arduino Due, they needed to modify the Arduino IDE so it would be relatively easy to change the IDE to support alternate toolchains to allow Arduino C/C++ to be compiled for these new processors. They did this with the introduction of the Board Manager and the SAM Core. A "core" is the collection of software components required by the Board Manager and the Arduino IDE to compile an Arduino C/C++ source file for the target MCU's machine language. Some ESP8266 enthusiasts developed an Arduino core for the ESP8266 WiFi SoC, popularly called the "ESP8266 Core for the Arduino IDE". This has become a leading software development platform for the various ESP8266-based modules and development boards, including NodeMCUs.

1.5.1.2 IR Sensor

The Infrared Obstacle Avoidance Sensor has a pair of infrared transmitting and receiving sensors. The infrared LED emits Infrared signals at certain frequency and when an obstacle appears on the line of infrared light, it is reflected back by the obstacle which is sensed by the receiver.

When the sensor detects an obstacle, the LED indicator lights up, giving a low-level output signal in the OUT pin. The sensor detects distance of 2 - 30cm. The sensor has a potentiometer which can be adjusted to change the detection distance.

Features:

- Voltage: DC 3-5V
- Detection distance: 2 ~ 30cm
- Detection angle: 35 °

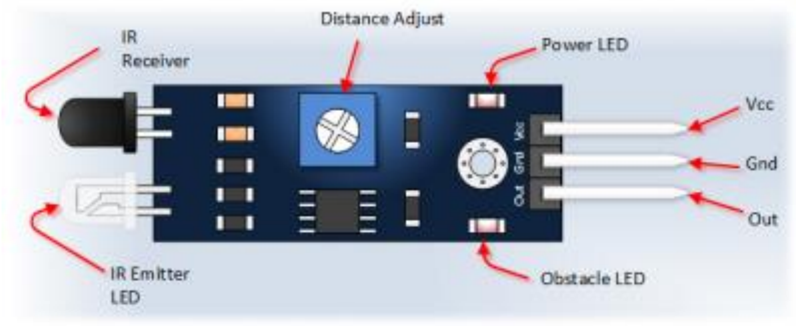


Figure 1.2 IR Obstacle Sensor

An IR sensor consists of an IR LED and an IR Photodiode; together they are called as Photo–Coupler or Opto–Coupler. As said before,the Infrared Obstacle Sensor has builtin IR transmitter and IR receiver. Infrared Transmitter is a light emitting diode (LED) which emits infrared radiations. Hence, they are called IR LED’s. Even though an IR LED looks like a normal LED, the radiation emitted by it is invisible to the human eye. Infrared receivers are also called as infrared sensors as they detect the radiation from an IR transmitter. IR receivers come in the form of photodiodes and phototransistors. Infrared Photodiodes are different from normal photo diodes as they detect only infrared radiation. When the IR transmitter emits radiation, it reaches the object and some of the radiation reflects back to the IR receiver. Based on the intensity of the reception by the IR receiver, the output of the sensor is defined.

1.5.1.3 Servo Motor

A servomotor (or servo motor) is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity, and acceleration. It consists of a suitable motor coupled to a sensor for position feedback. It also requires a relatively sophisticated controller, often a dedicated module designed specifically for use with servomotors.

Servomotors are not a specific class of motor, although the term servomotor is often used to refer to a motor suitable for use in a closed-loop control system.

Servomotors are used in applications such as robotics, CNC machinery, and automated manufacturing.

Mechanism: A servomotor is a closed-loop servomechanism that uses position feedback to control its motion and final position. The input to its control is a signal (either analog or digital) representing the position commanded for the output shaft.

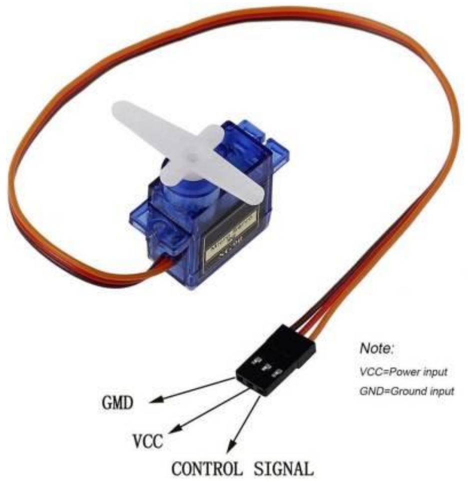


Figure 1.3 Servo Motor

The motor is paired with some type of position encoder to provide position and speed feedback. In the simplest case, only the position is measured. The measured position of the output is compared to the command position, the external input to the controller. If the output position differs from that required, an error signal is generated which then causes the motor to rotate in either direction, as needed to bring the output shaft to the appropriate position. As the positions approach, the error signal reduces to zero, and the motor stops.

The very simplest servomotors use position-only sensing via a potentiometer and bang-bang control of their motor; the motor always rotates at full speed (or is stopped). This type of servomotor is not widely used in industrial motion control, but it forms the basis of the simple and cheap servos used for radio-controlled models.

More sophisticated servomotors make use of an Absolute Encoder (a type of rotary encoder) to calculate the shafts position and infer the speed of the output shaft. A variable-speed drive is used to control the motor speed. Both of these enhancements, usually in combination with a PID control algorithm, allow the servomotor to be brought to its commanded position more quickly and more precisely, with less overshooting.

Servomotors vs. stepper motors: This Section needs additional citations for verification. Please help improve this article by adding citations to reliable sources in this Section. Unsourced material may be challenged and removed.

Servomotors are generally used as a high-performance alternative to the stepper motor. Stepper motors have some inherent ability to control position, as they have built-in output steps. This often allows them to be used as an open-loop position control, without any feedback encoder, as their drive signal specifies the number of steps of movement to rotate, but for this, the controller needs to 'know' the position of the stepper motor on power up. Therefore, on the first power-up, the controller will have to activate the stepper motor and turn it to a known position, e.g. until it activates an end limit switch. This can be observed when switching on an inkjet printer; the controller will move the ink jet carrier to the extreme left and right to establish the end positions. A servomotor can immediately turn to whatever angle the controller instructs it to, regardless of the initial position at power up if an absolute encoder is used.

The lack of feedback of a stepper motor limits its performance, as the stepper motor can only drive a load that is well within its capacity, otherwise missed steps under load may lead to positioning errors and the system may have to be restarted or recalibrated. The encoder and controller of a servomotor are an additional cost, but they optimize the performance of the overall system (for all of speed, power, and accuracy) relative to the capacity of the basic motor. With larger systems, where a powerful motor represents an increasing proportion of the system cost, servomotors have the advantage.

There has been increasing popularity in closed-loop stepper motors in recent years.[citation needed] They act like servomotors but have some differences in their software control to get smooth motion. The main benefit of a closed-loop stepper motor is its relatively low cost. There is also no need to tune the PID controller on a closed loop stepper system.

Many applications, such as laser cutting machines, may be offered in two ranges, the low-priced range using stepper motors and the high-performance range using servomotors.

1.5.2 Software Requirements

Table 1.2 Software Requirements

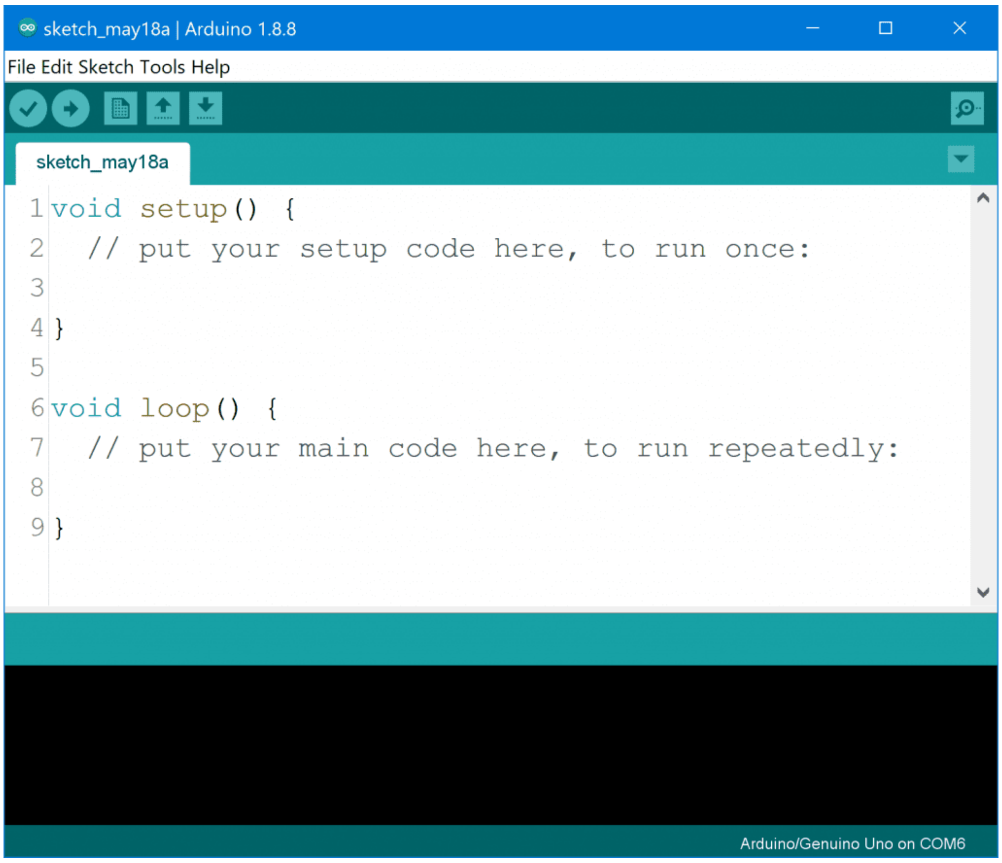
Number	**Description**	**Type**
1	Arduino IDE	Windows
2	Adafruit	Cloud Platform

1.5.2.1 Arduino IDE

Arduino is an open-source hardware and software company, project, and user community that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices. Its hardware products are licensed under a CC BY-SA license, while the software is licensed under the GNU Lesser General Public License (LGPL) or the GNU General Public License (GPL), permitting the manufacture of Arduino boards and software distribution by anyone. Arduino boards are available commercially from the official website or through authorized distributors.

Arduino board designs use a variety of microprocessors and controllers. The boards are equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards ('shields') or breadboards (for prototyping) and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs. The microcontrollers can be programmed using the C and C++ programming languages, using a standard API which is also known as the Arduino Programming Language, inspired by the Processing language and used with a modified version of the Processing IDE. In addition to using traditional compiler toolchains, the Arduino project provides an integrated development environment (IDE) and a command line tool developed in Go.

The Arduino project began in 2005 as a tool for students at the Interaction Design Institute Ivrea, Italy, aiming to provide a low-cost and easy way for novices and professionals to create devices that interact with their environment using sensors and actuators. Common examples of such devices intended for beginner hobbyists include simple robots, thermostats, and motion detectors.


sketch_may18a | Arduino 1.8.8
File Edit Sketch Tools Help
sketch_may18a

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Arduino/Genuino Uno on COM6

Figure: 1.4 ArduinoIDE Environment for building codes

The name Arduino comes from a bar in Ivrea, Italy, where some of the project's founders used to meet. The bar was named after Arduin of Ivrea, who was the margrave of the March of Ivrea and King of Italy from 1002 to 1014.

The Arduino project was started at the Interaction Design Institute Ivrea (IDII) in Ivrea, Italy.[3] At that time, the students used a BASIC Stamp microcontroller at a cost of $50. In 2003 Hernando Barragán created the development platform Wiring as a Master's thesis project at IDII, under the supervision of Massimo Banzi and Casey Reas. Casey Reas is known for co-creating, with Ben Fry, the Processing development platform. The project goal was to create simple, low cost tools for creating digital projects by non-engineers. The Wiring platform consisted of a printed circuit board (PCB) with an ATmega128 microcontroller, an IDE based on Processing and library functions to easily program the microcontroller. In 2005, Massimo Banzi, with David Mellis, another IDII student, and David Cuartielles, extended Wiring by adding support for the cheaper ATmega8 microcontroller. The new project, forked from Wiring, was called Arduino.

In the realm of electronics prototyping and programming, the Arduino IDE (Integrated Development Environment) has emerged as a powerful and versatile platform. Developed with a focus on simplicity and accessibility, the Arduino IDE has become the go-to choice for beginners and experienced developers alike. With its intuitive interface, extensive libraries, and a large supportive community, the Arduino IDE has revolutionized the way people create interactive electronic projects.

At its core, the Arduino IDE is a software application that provides a streamlined development environment for programming Arduino boards. Arduino boards are microcontroller-based platforms that allow users to build interactive electronic projects, ranging from simple LED blinkers to complex robotic systems. The Arduino IDE acts as a bridge between the hardware and software, providing an easy-to-use interface for writing, compiling, and uploading code to Arduino boards.

One of the key strengths of the Arduino IDE is its user-friendly interface. The IDE offers a simplified coding environment that utilizes a variant of the C++ programming language. The interface includes a text editor with syntax highlighting, making it easier for users to write and edit their code. It also provides a built-in serial monitor, which allows developers to communicate with their Arduino boards and debug their programs in real-time. The simplicity of the interface lowers the barrier to entry for beginners, enabling them to quickly grasp the basics of programming and electronics.

The Arduino IDE comes with a rich library ecosystem, offering a wide range of pre-written code modules called "sketches." These sketches cover various functionalities, such as controlling LEDs, reading sensor data, and communicating with other devices. Users can easily import and modify these sketches to suit their specific project requirements. Additionally, the Arduino community actively contributes to the library collection, continually expanding the available resources and providing support for a vast array of hardware components and sensors.

Another notable feature of the Arduino IDE is its compatibility with multiple operating systems, including Windows, macOS, and Linux. This cross-platform support allows developers to use their preferred operating system without limitations. The IDE also supports a broad range of Arduino boards, accommodating different microcontroller architectures and configurations. This flexibility makes it convenient for users to choose the most suitable Arduino board for their specific project needs.

The Arduino IDE promotes a collaborative and supportive community. Arduino enthusiasts worldwide share their projects, code snippets, and troubleshooting tips through various online platforms, such as forums and social media groups. This community-driven approach encourages knowledge sharing and helps users overcome challenges they may encounter during their development journey. The wealth of online resources, tutorials, and examples further enhances the learning experience, making the Arduino IDE an excellent tool for both beginners and experienced developers.

Furthermore, the Arduino IDE integrates seamlessly with Arduino's open-source ecosystem. The hardware designs, software libraries, and documentation associated with Arduino boards are freely available to the public, fostering innovation and enabling users to extend the capabilities of their projects. This open-source philosophy empowers developers to explore new possibilities, contribute back to the community, and build upon the work of others.

In conclusion, the Arduino IDE has revolutionized the world of electronics prototyping and programming. Its user-friendly interface, extensive library collection, cross-platform compatibility, and supportive community make it an invaluable tool for both beginners and advanced users. By providing a simplified development environment and a rich set of resources, the Arduino IDE empowers individuals to turn their creative ideas into interactive electronic projects. The integration with Arduino's open-source ecosystem further enhances its appeal, fostering collaboration and innovation within the community. Whether it's a hobbyist tinkering with their first project or a professional developer building a complex system, the Arduino IDE serves as a versatile and accessible platform for unleashing creativity and turning imagination into reality.

1.5.2.2 Adafruit Cloud

Adafruit Industries is an open-source hardware company based in New York City. It was founded by Limor Fried in 2005. The company designs, manufactures and sells electronics products, electronics components, tools, and accessories. It also produces learning resources, including live and recorded videos about electronics, technology, and programming.

In addition to distributing third-party components and boards such as the Raspberry Pi, Adafruit develops and sells its own development boards for educational and hobbyist purposes. In 2016, the company released the Circuit Playground, a board with an Atmel ATmega32u4 microcontroller and a variety of sensors, followed in 2017 by the more powerful Atmel SAMD21 based Circuit Playground Express. They, like many Adafruit products, are circular in shape for ease of use in education and wearable electronics projects, along with the FLORA and Gemma, the company's wearable electronics development platforms. In 2017, Adafruit Industries' best-selling product was the Circuit Playground Express.

Adafruit.io is a cloud service - that just means we run it for you and you don't have to manage it. You can connect to it over the Internet. It's meant primarily for storing and then retrieving data but it can do a lot more than just that!

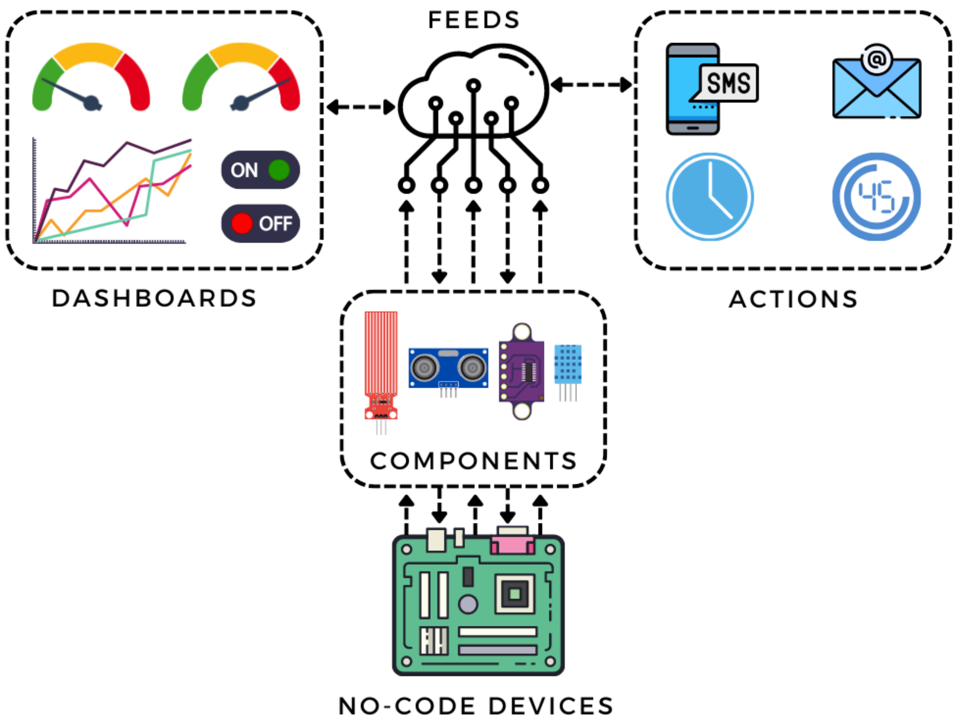


Figure 1.5 Adafruit Cloud

Features:

- Display your data in real-time, online
- Make your project internet-connected: Control motors, read sensor data, and more!
- Connect projects to web services like Twitter, RSS feeds, weather services, etc.
- Connect your project to other internet-enabled devices
- The best part? All of the above is do-able for **free** with Adafruit IO.

In the ever-expanding realm of the Internet of Things (IoT), developers and enthusiasts are constantly seeking robust platforms to connect and manage their devices. Among the many platforms available, Adafruit Cloud Platform has emerged as a prominent player, offering a comprehensive ecosystem for IoT development, data management, and visualization. With its user-friendly interface, extensive features, and commitment to open-source principles, Adafruit Cloud Platform has become a go-to solution for individuals and businesses alike.

At its core, Adafruit Cloud Platform provides a cloud-based infrastructure that enables seamless connectivity and communication between IoT devices. Developers can connect a wide range of devices, sensors, and actuators using popular protocols such as MQTT, HTTP, and WebSocket. Whether it's a tiny microcontroller or a sophisticated IoT gateway, the platform offers the flexibility to integrate and manage diverse hardware.

One of the standout features of Adafruit Cloud Platform is its intuitive web interface. Users can easily navigate through the platform, set up their devices, and define data feeds for sensor readings. The platform supports real-time data streaming, allowing developers to monitor their devices and receive instant updates. The dashboard provides customizable visualizations, enabling users to create meaningful charts, graphs, and gauges that reflect the behavior of their IoT systems.

Data management is a critical aspect of any IoT project, and Adafruit Cloud Platform excels in this domain. It offers a powerful data storage system that allows users to store and retrieve sensor data effortlessly. The platform leverages cloud storage and databases to ensure scalability and reliability. Furthermore, it provides APIs and libraries for popular programming languages, making it easier for developers to interact with the data and build custom applications and integrations.

Adafruit Cloud Platform also emphasizes collaboration and community engagement. Through its open-source nature, it encourages developers to share their projects and contribute to the platform's ecosystem. The Adafruit community is a vibrant hub where users can exchange ideas, seek assistance, and showcase their creations. This collaborative environment fosters innovation and enables users to learn from one another, making Adafruit Cloud Platform an ideal choice for both beginners and experienced IoT enthusiasts.

Security is of paramount importance when dealing with IoT devices, and Adafruit Cloud Platform prioritizes the protection of user data. The platform implements robust authentication and authorization mechanisms, ensuring that only authorized individuals can access and control devices. It also supports secure communication protocols, such as TLS/SSL, to encrypt data transmission between devices and the cloud. With these measures in place, users can trust that their IoT deployments are well-guarded against potential threats.

In conclusion, Adafruit Cloud Platform stands out as a comprehensive and user-friendly solution for IoT development, data management, and visualization. Its intuitive interface, extensive features, and commitment to open-source principles make it an appealing choice for individuals and businesses venturing into the IoT realm. By offering seamless device connectivity, real-time data streaming, and customizable visualizations, Adafruit Cloud Platform empowers users to build and monitor their IoT systems with ease. With its focus on collaboration and community engagement, the platform cultivates a vibrant ecosystem that fosters innovation and knowledge sharing. Above all, its robust security measures ensure the protection of user data, instilling confidence in the reliability and integrity of IoT deployments.

CHAPTER 2

FEASIBILITY STUDY

2.1 FEASIBILITY STUDY

A feasibility study is a high-level capsule version of the entire System analysis and Design Process. The study begins by classifying the problem definition. Feasibility is to determine if it’s worth doing. Once an acceptance problem definition has been generated, the analyst develops a logical model of the system. A search for alternatives is analyzed carefully. There are 3 parts in feasibility study.

1. Economical Feasibility
2. Technical Feasibility
3. Operational Feasibility

2.2 Economical Feasibility

An economic feasibility study for a smart parking system examines the financial viability of implementing such a system and assesses its potential benefits and costs. The study evaluates whether the project can generate a positive return on investment and contribute to the overall economic well-being of the stakeholders involved.

A smart parking system utilizes advanced technologies like sensors, data analytics, and mobile applications to optimize parking management. It aims to improve the efficiency of parking operations, reduce traffic congestion, and enhance the overall user experience.

The economic feasibility study would analyze various factors, including the initial investment required to deploy the smart parking system, such as infrastructure setup, hardware costs, software development, and installation. Additionally, ongoing operational costs like maintenance, staff, and system upgrades would be considered.

The study would also assess the potential revenue streams for the smart parking system, such as parking fees, advertising, or data monetization. It would analyze the market demand, pricing strategies, and competition to estimate the potential income generated.

Furthermore, the study would consider the potential benefits of the smart parking system, such as reduced fuel consumption, decreased emissions, and improved traffic flow, which can have positive economic impacts on the community.

By analyzing the costs, revenues, and potential benefits, the economic feasibility study would provide stakeholders with valuable insights into the financial viability of implementing a smart parking system, enabling them to make informed decisions about the project's implementation.

2.3 Technical Feasibility

A technical feasibility study for a smart parking system assesses the viability and practicality of implementing such a system from a technological standpoint. It evaluates the technical requirements, infrastructure, and capabilities needed to deploy and operate the system effectively.

The study begins by identifying the key components of the smart parking system, such as sensors, communication networks, data management systems, and user interfaces. It examines the availability and compatibility of these technologies in the intended implementation area.

The study also analyzes the scalability and reliability of the system to handle the expected volume of parking spaces and user interactions. It considers factors like sensor accuracy, connectivity, and data processing capabilities to ensure accurate and real-time information for parking availability and reservation.

Furthermore, the technical feasibility study evaluates the integration of the smart parking system with existing infrastructure, such as parking meters, payment systems, and enforcement mechanisms. It examines the compatibility and interoperability requirements to ensure seamless integration and functionality.

Moreover, the study examines the security and privacy aspects of the smart parking system. It considers measures to protect user data, prevent unauthorized access, and ensure the system's overall integrity and trustworthiness.

Additionally, the study assesses the technical expertise and resources required for the implementation, operation, and maintenance of the smart parking system. It examines factors like skill sets, training, and technical support to ensure the system can be effectively managed.

By conducting a technical feasibility study, stakeholders can gain insights into the technological requirements, challenges, and opportunities associated with implementing a smart parking system. It helps in making informed decisions regarding system design, technology selection, and resource allocation to ensure a successful deployment and operation of the smart parking system.

2.4 Operational Feasibility

An operational feasibility study for a smart parking system examines the practicality and effectiveness of implementing such a system from an operational perspective. It evaluates whether the system can be integrated smoothly into existing operations, processes, and workflows.

The study begins by analyzing the current parking management practices and identifying the pain points and inefficiencies that the smart parking system aims to address. It assesses the potential impact of the system on streamlining operations, reducing congestion, improving user experience, and enhancing overall efficiency.

Furthermore, the study evaluates the readiness of the organization or stakeholders to adopt and embrace the changes associated with the smart parking system. It considers factors such as organizational culture, employee training needs, and potential resistance to change. It also examines the capacity and willingness to allocate resources for implementation and ongoing operation and maintenance.

Moreover, the operational feasibility study assesses the compatibility of the smart parking system with existing processes and infrastructure. It examines how the system can be seamlessly integrated with parking enforcement, payment systems, and customer support channels. It also considers the adaptability of the system to different parking facility types, such as on-street parking, parking lots, or garages.

Additionally, the study evaluates the potential risks and challenges associated with the operation of the smart parking system. It considers factors like system reliability, data accuracy, user adoption, and the ability to handle peak parking demand periods effectively.

By conducting an operational feasibility study, stakeholders can gain insights into the practicality and readiness of implementing a smart parking system. It helps in identifying potential operational hurdles, designing effective implementation plans, and ensuring a smooth transition to the new system while maximizing the benefits for both the organization and end users.

CHAPTER 3

DESIGN AND PLANNING

3.1 SOFTWARE DEVELOPMENT LIFE CYCLE MODEL PROTOTYPE MODEL

The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software. In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.

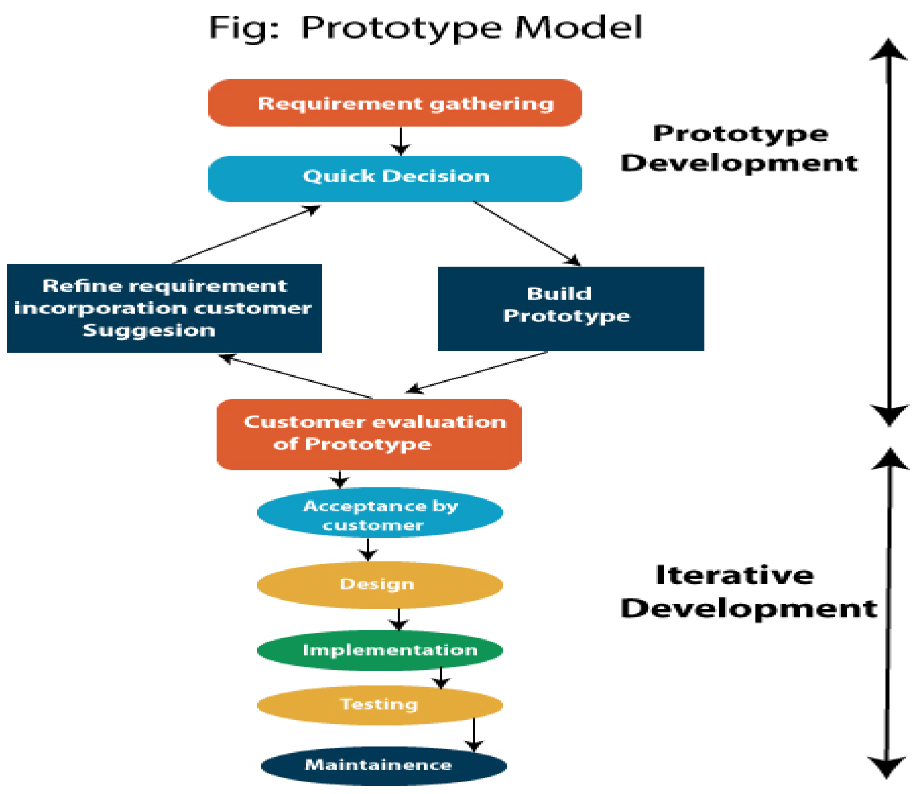


Figure 3.1 Prototype Model

The Software Development Life Cycle (SDLC) for a smart parking system typically consists of the following phases:

3.1.1 Requirement Gathering:

In this phase, the requirements of the smart parking system are collected by engaging with stakeholders, including parking facility managers, users, and other relevant parties. This involves understanding the desired features, functionalities, and goals of the system.

3.1.2 System Design:

The system design phase involves creating a blueprint for the smart parking system. This includes designing the architecture, database structure, user interfaces, and integration points with other systems. It also involves defining the hardware and software components required for the system.

3.1.3 Development:

In this phase, the actual development of the smart parking system takes place. Programmers and developers write code to implement the functionalities and features outlined in the system design phase. The software development process may follow methodologies like Agile or Waterfall, depending on the project requirements.

3.1.4 Testing:

The testing phase is crucial to ensure the quality and functionality of the smart parking system. It involves various types of testing, such as unit testing, integration testing, system testing, and user acceptance testing. This phase helps identify and fix any bugs, errors, or issues before the system is deployed.

3.1.5 Deployment:

Once the smart parking system has been thoroughly tested and deemed ready for use, it is deployed in the target environment. This involves installing and configuring the necessary hardware and software components, ensuring compatibility with existing infrastructure, and training staff on how to use and manage the system.

3.1.6 Maintenance and Support:

After deployment, the smart parking system requires ongoing maintenance and support. This includes addressing any issues or bugs that arise, providing regular software updates and enhancements, and offering technical support to users and administrators. The maintenance phase ensures the system continues to function effectively and meets evolving user needs.

Throughout the SDLC, it is essential to follow best practices for project management, documentation, and communication to ensure the successful development and implementation of the smart parking system. It is also important to consider security and data privacy aspects throughout the development process to protect user information and prevent unauthorized access.

3.2 Protocols Used for Cloud Connectivity

3.2.1 MQTT Protocol

MQTT (originally an initialism of MQ Telemetry Transport[a]) is a lightweight, publish-subscribe, machine to machine network protocol for message queue/message queuing service. It is designed for connections with remote locations that have devices with resource constraints or limited network bandwidth, such as in the Internet of Things (IoT). It must run over a transport protocol that provides ordered, lossless, bi-directional connections—typically, TCP/IP. It is an open OASIS standard and an ISO recommendation (ISO/IEC 20922).

Andy Stanford-Clark (IBM) and Arlen Nipper (then working for Eurotech, Inc.) authored the first version of the protocol in 1999. It was used to monitor oil pipelines within the SCADA industrial control system. The goal was to have a protocol that is bandwidth-efficient, lightweight and uses little battery power, because the devices were connected via satellite link which, at that time, was extremely expensive.

Historically, the "MQ" in "MQTT" came from the IBM MQ (then 'MQSeries') product line, where it stands for "Message Queue". However, the protocol provides publish-and-subscribe messaging (no queues, in spite of the name). In the specification opened by IBM as version 3.1 the protocol was referred to as "MQ Telemetry Transport". Subsequent versions released by OASIS strictly refers to the protocol as just "MQTT", although the technical committee itself is named "OASIS Message Queuing Telemetry Transport Technical Committee". Since 2013, "MQTT" does not stand for anything.

In 2013, IBM submitted MQTT v3.1 to the OASIS specification body with a charter that ensured only minor changes to the specification could be accepted. After taking over maintenance of the standard from IBM, OASIS released version 3.1.1 on October 29, 2014. A more substantial upgrade to MQTT version 5, adding several new features, was released on March 7, 2019.

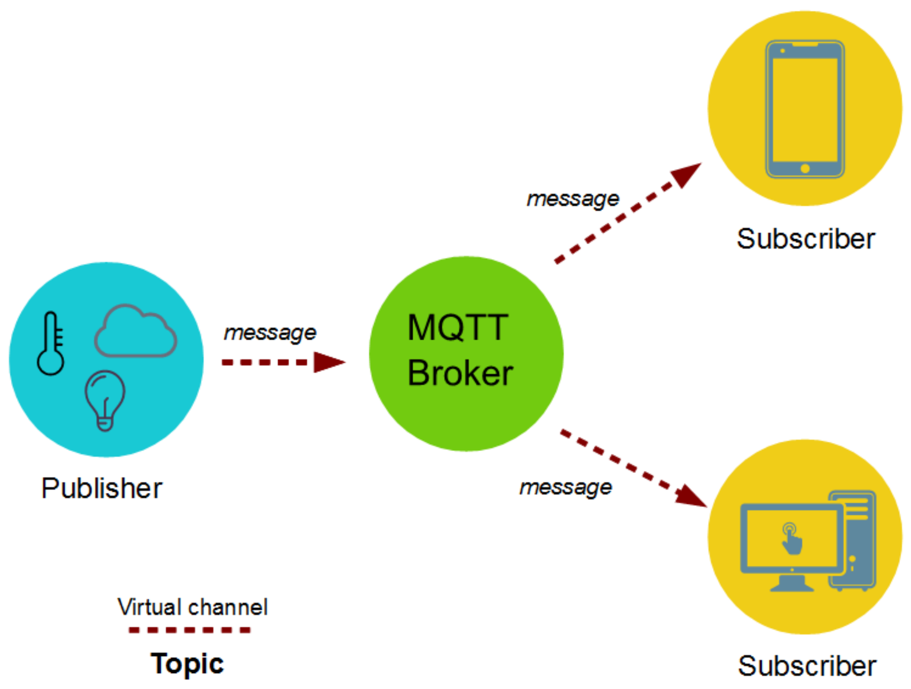


Figure 3.2 Basic Architecture of Protocol

MQTT-SN (MQTT for Sensor Networks) is a variation of the main protocol aimed at battery-powered embedded devices on non-TCP/IP networks, such as Zigbee. The MQTT protocol defines two types of network entities: a message broker and a number of clients. An MQTT broker is a server that receives all messages from the clients and then routes the messages to the appropriate destination clients. An MQTT client is any device (from a micro controller up to a fully-fledged server) that runs an MQTT library and connects to an MQTT broker over a network.

Information is organized in a hierarchy of topics. When a publisher has a new item of data to distribute, it sends a control message with the data to the connected broker. The broker then distributes the information to any clients that have subscribed to that topic. The publisher does not need to have any data on the number or locations of subscribers, and subscribers, in turn, do not have to be configured with any data about the publishers.

If a broker receives a message on a topic for which there are no current subscribers, the broker discards the message unless the publisher of the message designated the message as a retained message. A retained message is a normal MQTT message with the retained flag set to true. The broker stores the last retained message and the corresponding QoS for the selected topic. Each client that subscribes to a topic pattern that matches the topic of the retained message receives the retained message immediately after they subscribe. The broker stores only one retained message per topic. This allows new subscribers to a topic to receive the most current value rather than waiting for the next update from a publisher.

When a publishing client first connects to the broker, it can set up a default message to be sent to subscribers if the broker detects that the publishing client has unexpectedly disconnected from the broker.

Clients only interact with a broker, but a system may contain several broker servers that exchange data based on their current subscribers' topics.

A minimal MQTT control message can be as little as two bytes of data. A control message can carry nearly 256 megabytes of data if needed. There are fourteen defined message types used to connect and disconnect a client from a broker, to publish data, to acknowledge receipt of data, and to supervise the connection between client and server.

MQTT relies on the TCP protocol for data transmission. A variant, MQTT-SN, is used over other transports such as UDP or Bluetooth.

MQTT sends connection credentials in plain text format and does not include any measures for security or authentication. This can be provided by using TLS to encrypt and protect the transferred information against interception, modification or forgery.

The default unencrypted MQTT port is 1883. The encrypted port is 8883.

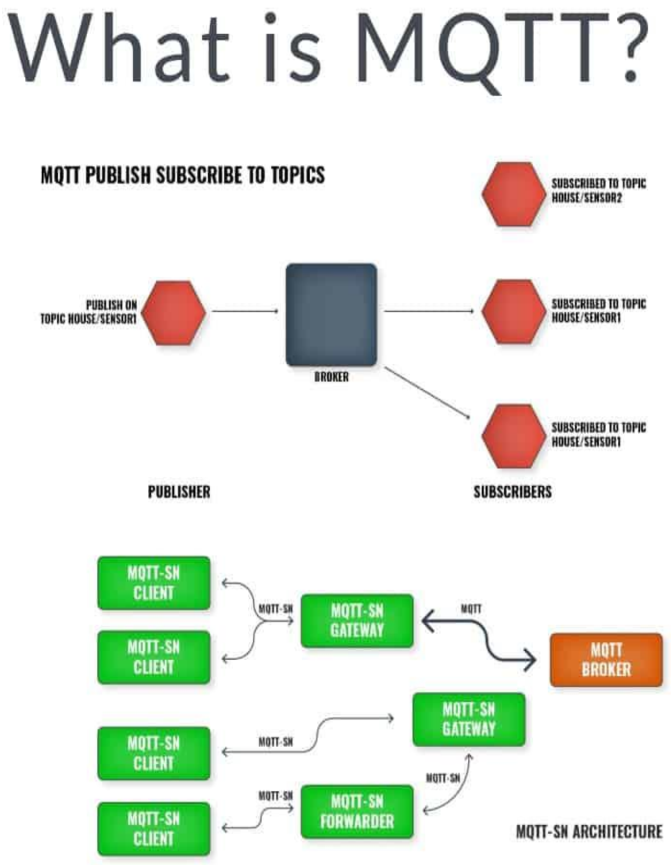


Figure 3.3 MQTT Components

To understand the MQTT architecture, we first look at the components of the MQTT.

- **Message**
- **Client**
- **Server or Broker**
- **TOPIC**

3.2.1.1 Message

The message is the data that is carried out by the protocol across the network for the application. When the message is transmitted over the network, then the message contains the following parameters:

- Payload data
- Quality of Service (QoS)
- Collection of Properties
- Topic Name

3.2.1.2 Client

In MQTT, the subscriber and publisher are the two roles of a client. The clients subscribe to the topics to publish and receive messages. In simple words, we can say that if any program or device uses an MQTT, then that device is referred to as a client. A device is a client if it opens the network connection to the server, publishes messages that other clients want to see, subscribes to the messages that it is interested in receiving, unsubscribes to the messages that it is not interested in receiving, and closes the network connection to the server.

In MQTT, the client performs two operations:

Publish: When the client sends the data to the server, then we call this operation as a publish.

Subscribe: When the client receives the data from the server, then we call this operation a subscription.

3.2.1.3 Server

The device or a program that allows the client to publish the messages and subscribe to the messages. A server accepts the network connection from the client, accepts the messages from the client, processes the subscribe and unsubscribe requests, forwards the application messages to the client, and closes the network connection from the client.

3.2.1.4 TOPIC

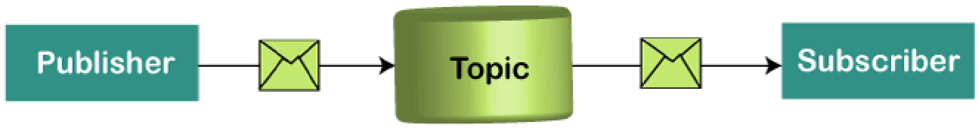


Figure 3.4 MQTT Protocol

The label provided to the message is checked against the subscription known by the server is known as TOPIC.

3.2.1.5 MQTT Interaction Code

Adafruit_MQTT.cpp

```
// The MIT License (MIT)
//
// Copyright (c) 2015 Adafruit Industries
```

```
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE
// SOFTWARE.
#include "Adafruit_MQTT.h"

#if defined(ARDUINO_SAMD_ZERO) || defined(ARDUINO_SAMD_MKR1000) ||
\
  defined(ARDUINO_SAMD_MKR1010) || defined(ARDUINO_ARCH_SAMD)
static char *dtostrf(double val, signed char width, unsigned char prec,
                    char *sout) {
 char fmt[20];
 sprintf(fmt, "%%%d.%df", width, prec);
 sprintf(sout, fmt, val);
 return sout;
}
#endif

#if defined(ESP8266)
int strncasecmp(const char *str1, const char *str2, int len) {
 int d = 0;
 while (len--) {
   int c1 = tolower(*str1++);
   int c2 = tolower(*str2++);
```

```
    if (((d = c1 - c2) != 0) || (c2 == '\0')) {
      return d;
    }
  }
  return 0;
}
#endif

void printBuffer(uint8_t *buffer, uint16_t len) {
  DEBUG_PRINTER.print('\t');
  for (uint16_t i = 0; i < len; i++) {
    if (isprint(buffer[i]))
      DEBUG_PRINTER.write(buffer[i]);
    else
      DEBUG_PRINTER.print(" ");
    DEBUG_PRINTER.print(F(" [0x"));
    if (buffer[i] < 0x10)
      DEBUG_PRINTER.print("0");
    DEBUG_PRINTER.print(buffer[i], HEX);
    DEBUG_PRINTER.print("], ");
    if (i % 8 == 7) {
      DEBUG_PRINTER.print("\n\t");
    }
  }
  DEBUG_PRINTER.println();
}

/* Not used now, but might be useful in the future
static uint8_t *stringprint(uint8_t *p, char *s) {
  uint16_t len = strlen(s);
  p[0] = len >> 8; p++;
  p[0] = len & 0xFF; p++;
  memmove(p, s, len);
  return p+len;
}
*/

static uint8_t *stringprint(uint8_t *p, const char *s, uint16_t maxlen = 0) {
  // If maxlen is specified (has a non-zero value) then use it as the maximum
  // length of the source string to write to the buffer.  Otherwise write
  // the entire source string.
  uint16_t len = strlen(s);
  if (maxlen > 0 && len > maxlen) {
    len = maxlen;
  }
  /*
  for (uint8_t i=0; i<len; i++) {
    Serial.write(pgm_read_byte(s+i));
  }
  */
  p[0] = len >> 8;
  p++;
  p[0] = len & 0xFF;
  p++;
  strncpy((char *)p, s, len);
  return p + len;
}

// packetAdditionalLen is a helper function used to figure out
// how bigger the payload needs to be in order to account for
// its variable length field. As per
// http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Table_2.4_Size
// See also readFullPacket
static uint16_t packetAdditionalLen(uint32_t currLen) {
  /* Increase length field based on current length */
  if (currLen < 128) // 7-bits
    return 0;
  if (currLen < 16384) // 14-bits
    return 1;
  if (currLen < 2097152) // 21-bits
    return 2;
  return 3;
}

// Adafruit_MQTT Definition ////////////////////////////////////////////////////

Adafruit_MQTT::Adafruit_MQTT(const char *server, uint16_t port, const char *cid,
                             const char *user, const char *pass) {
  servername = server;
  portnum = port;
  clientid = cid;
  username = user;
  password = pass;

  // reset subscriptions
  for (uint8_t i = 0; i < MAXSUBSCRIPTIONS; i++) {
    subscriptions[i] = 0;
  }
  will_topic = 0;
  will_payload = 0;
  will_qos = 0;
  will_retain = 0;

  keepAliveInterval = MQTT_CONN_KEEPALIVE;

  packet_id_counter = 0;
}

Adafruit_MQTT::Adafruit_MQTT(const char *server, uint16_t port,
                             const char *user, const char *pass) {
  servername = server;
  portnum = port;
  clientid = "";
  username = user;
  password = pass;

  // reset subscriptions
  for (uint8_t i = 0; i < MAXSUBSCRIPTIONS; i++) {
    subscriptions[i] = 0;
  }

  will_topic = 0;
  will_payload = 0;
  will_qos = 0;
  will_retain = 0;

  keepAliveInterval = MQTT_CONN_KEEPALIVE;

  packet_id_counter = 0;
}

int8_t Adafruit_MQTT::connect() {
  // Connect to the server.
  if (!connectServer())
    return -1;

  // Construct and send connect packet.
  uint8_t len = connectPacket(buffer);
  if (!sendPacket(buffer, len))
    return -1;
  // Read connect response packet and verify it
  len = readFullPacket(buffer, MAXBUFFERSIZE, CONNECT_TIMEOUT_MS);
  if (len != 4) {
    return -1;
  }
  if ((buffer[0] != (MQTT_CTRL_CONNECTACK << 4)) || (buffer[1] != 2)) {
    return -1;
  }
  if (buffer[3] != 0)
    return buffer[3];

  // Setup subscriptions once connected.
  for (uint8_t i = 0; i < MAXSUBSCRIPTIONS; i++) {
    // Ignore subscriptions that aren't defined.
    if (subscriptions[i] == 0)
      continue;

    boolean success = false;
    for (uint8_t retry = 0; (retry < 3) && !success;
         retry++) { // retry until we get a suback
      // Construct and send subscription packet.
      uint8_t len = subscribePacket(buffer, subscriptions[i]->topic,
                                    subscriptions[i]->qos);
      if (!sendPacket(buffer, len))
        return -1;

      if (MQTT_PROTOCOL_LEVEL < 3) // older versions didn't suback
        break;

      // Check for SUBACK if using MQTT 3.1.1 or higher
      // TODO: The Server is permitted to start sending PUBLISH packets matching
      // the Subscription before the Server sends the SUBACK Packet. (will
      // really need to use callbacks - ada)

      if (processPacketsUntil(buffer, MQTT_CTRL_SUBACK,
SUBACK_TIMEOUT_MS)) {
        success = true;
        break;
      }
    }
    if (!success)
      return -2; // failed to sub for some reason
  }
  return 0;
}

int8_t Adafruit_MQTT::connect(const char *user, const char *pass) {
  username = user;
  password = pass;
  return connect();
}

void Adafruit_MQTT::processSubscriptionPacket(Adafruit_MQTT_Subscribe *sub) {
  if (sub->callback_uint32t != NULL) {
    // execute callback in integer mode
    uint32_t data = 0;
    data = atoi((char *)sub->lastread);
    sub->callback_uint32t(data);
  } else if (sub->callback_double != NULL) {
    // execute callback in doublefloat mode
    double data = 0;
    data = atof((char *)sub->lastread);
    sub->callback_double(data);
  } else if (sub->callback_buffer != NULL) {
    // execute callback in buffer mode
    sub->callback_buffer((char *)sub->lastread, sub->datalen);
  } else if (sub->callback_io != NULL) {
    // execute callback in io mode
    ((sub->io_mqtt)->*(sub->callback_io))((char *)sub->lastread, sub->datalen);
  } else {
    DEBUG_PRINTLN(
        "ERROR: Subscription packet did not have an associated callback");
    return;
  }
  // mark subscription message as "read""
  sub->new_message = false;
}

uint16_t Adafruit_MQTT::processPacketsUntil(uint8_t *buffer,
                                            uint8_t waitforpackettype,
                                            uint16_t timeout) {
  uint16_t len;

  while (true) {
    len = readFullPacket(buffer, MAXBUFFERSIZE, timeout);

    if (len == 0) {
      break;
    }

    uint8_t packetType = (buffer[0] >> 4);
    if (packetType == waitforpackettype) {
      return len;
    } else {
      if (packetType == MQTT_CTRL_PUBLISH) {
        Adafruit_MQTT_Subscribe *sub = handleSubscriptionPacket(len);
        if (sub)
          processSubscriptionPacket(sub);
      } else {
        ERROR_PRINTLN(F("Dropped a packet"));
      }
    }
  }
  return 0;
}

uint16_t Adafruit_MQTT::readFullPacket(uint8_t *buffer, uint16_t maxsize,
                                       uint16_t timeout) {
  // will read a packet and Do The Right Thing with length
  uint8_t *pbuff = buffer;

  uint16_t rlen;

  // read the packet type:
  rlen = readPacket(pbuff, 1, timeout);
  if (rlen != 1)
    return 0;

  DEBUG_PRINT(F("Packet Type:\t"));
  DEBUG_PRINTBUFFER(pbuff, rlen);
  pbuff++;

  uint32_t value = 0;
  uint32_t multiplier = 1;
  uint8_t encodedByte;

  do {
    rlen = readPacket(pbuff, 1, timeout);
    if (rlen != 1)
      return 0;
    encodedByte = pbuff[0]; // save the last read val
    pbuff++;               // get ready for reading the next byte
    uint32_t intermediate = encodedByte & 0x7F;
    intermediate *= multiplier;
    value += intermediate;
    multiplier *= 128;
    if (multiplier > (128UL * 128UL * 128UL)) {
      DEBUG_PRINT(F("Malformed packet len\n"));
      return 0;
    }
  } while (encodedByte & 0x80);

  DEBUG_PRINT(F("Packet Length:\t"));
  DEBUG_PRINTLN(value);

  // maxsize is limited to 65536 by 16-bit unsigned
  if (value > uint32_t(maxsize - (pbuff - buffer) - 1)) {
    DEBUG_PRINTLN(F("Packet too big for buffer"));
    rlen = readPacket(pbuff, (maxsize - (pbuff - buffer) - 1), timeout);
  } else {
    rlen = readPacket(pbuff, value, timeout);
  }
  // DEBUG_PRINT(F("Remaining packet:\t")); DEBUG_PRINTBUFFER(pbuff, rlen);

  return ((pbuff - buffer) + rlen);
}

const __FlashStringHelper *Adafruit_MQTT::connectErrorString(int8_t code) {
  switch (code) {
  case 1:
    return F(
        "The Server does not support the level of the MQTT protocol requested");
  case 2:
    return F(
        "The Client identifier is correct UTF-8 but not allowed by the Server");
  case 3:
    return F("The MQTT service is unavailable");
  case 4:
    return F("The data in the user name or password is malformed");
  case 5:
    return F("Not authorized to connect");
  case 6:
    return F("Exceeded reconnect rate limit. Please try again later.");
  case 7:
    return F("You have been banned from connecting. Please contact the MQTT "
            "server administrator for more details.");
  case -1:
    return F("Connection failed");
  case -2:
    return F("Failed to subscribe");
  default:
    return F("Unknown error");
  }
}

bool Adafruit_MQTT::disconnect() {

  // Construct and send disconnect packet.
  uint8_t len = disconnectPacket(buffer);
  if (!sendPacket(buffer, len))
    DEBUG_PRINTLN(F("Unable to send disconnect packet"));

  return disconnectServer();
}

bool Adafruit_MQTT::publish(const char *topic, const char *data, uint8_t qos,
                            bool retain) {
  return publish(topic, (uint8_t *)(data), strlen(data), qos, retain);
}

bool Adafruit_MQTT::publish(const char *topic, uint8_t *data, uint16_t bLen,
                            uint8_t qos, bool retain) {
  // Construct and send publish packet.
  uint16_t len = publishPacket(buffer, topic, data, bLen, qos,
                               (uint16_t)sizeof(buffer), retain);

  if (!sendPacket(buffer, len))
    return false;

  // If QOS level is high enough verify the response packet.
  if (qos > 0) {
    len          =          processPacketsUntil(buffer,          MQTT_CTRL_PUBACK,
PUBLISH_TIMEOUT_MS);

    DEBUG_PRINT(F("Publish QOS1+ reply:\t"));
    DEBUG_PRINTBUFFER(buffer, len);
    if (len != 4)
      return false;
    uint16_t packnum = buffer[2];
    packnum <<= 8;
    packnum |= buffer[3];

    // we increment the packet_id_counter right after publishing so inc here too
    // to match
    packnum++;
    if (packnum != packet_id_counter)
      return false;
  }

  return true;
}

bool Adafruit_MQTT::will(const char *topic, const char *payload, uint8_t qos,
                         uint8_t retain) {

  if (connected()) {
    DEBUG_PRINT(F("Will defined after connect"));
    return false;
  }

  will_topic = topic;
  will_payload = payload;
  will_qos = qos;
  will_retain = retain;

  return true;
}

/**************************************************************************
*****/
/*!
    @brief  Sets the connect packet's KeepAlive Interval, in seconds. This
            function MUST be called prior to connect().
    @param    keepAlive
              Maximum amount of time without communication between the
              client and the MQTT broker, in seconds.
    @returns  True if called prior to connect(), False otherwise.
*/
/**************************************************************************
*****/
bool Adafruit_MQTT::setKeepAliveInterval(uint16_t keepAlive) {
  if (connected()) {
    DEBUG_PRINT(F("keepAlive defined after connection established."));
    return false;
  }
  keepAliveInterval = keepAlive;
  return true;
}

bool Adafruit_MQTT::subscribe(Adafruit_MQTT_Subscribe *sub) {
  uint8_t i;
  // see if we are already subscribed
  for (i = 0; i < MAXSUBSCRIPTIONS; i++) {
    if (subscriptions[i] == sub) {
      DEBUG_PRINTLN(F("Already subscribed"));
      return true;
    }
  }
  if (i == MAXSUBSCRIPTIONS) { // add to subscriptionlist
    for (i = 0; i < MAXSUBSCRIPTIONS; i++) {
      if (subscriptions[i] == 0) {
        DEBUG_PRINT(F("Added sub "));
        DEBUG_PRINTLN(i);
        subscriptions[i] = sub;
        return true;
      }
    }
  }

  DEBUG_PRINTLN(F("no more subscription space :("));
  return false;
}

bool Adafruit_MQTT::unsubscribe(Adafruit_MQTT_Subscribe *sub) {
  uint8_t i;

  // see if we are already subscribed
  for (i = 0; i < MAXSUBSCRIPTIONS; i++) {

    if (subscriptions[i] == sub) {

      DEBUG_PRINTLN(
          F("Found matching subscription and attempting to unsubscribe."));

      // Construct and send unsubscribe packet.
      uint8_t len = unsubscribePacket(buffer, subscriptions[i]->topic);
      // sending unsubscribe failed
      if (!sendPacket(buffer, len))
        return false;

      // if QoS for this subscription is 1 or 2, we need
      // to wait for the unsuback to confirm unsubscription
      if (subscriptions[i]->qos > 0 && MQTT_PROTOCOL_LEVEL > 3) {

        // wait for UNSUBACK
        len = readFullPacket(buffer, MAXBUFFERSIZE, CONNECT_TIMEOUT_MS);
        DEBUG_PRINT(F("UNSUBACK:\t"));
        DEBUG_PRINTBUFFER(buffer, len);

        if ((len != 5) || (buffer[0] != (MQTT_CTRL_UNSUBACK << 4))) {
          return false; // failure to unsubscribe
        }
      }

      subscriptions[i] = 0;
      return true;
    }
  }

  // subscription not found, so we are unsubscribed
  return true;
}

void Adafruit_MQTT::processPackets(int16_t timeout) {

  uint32_t elapsed = 0, endtime, starttime = millis();

  while (elapsed < (uint32_t)timeout) {
    Adafruit_MQTT_Subscribe *sub = readSubscription(timeout - elapsed);
    if (sub)
      processSubscriptionPacket(sub);
    // keep track over elapsed time
    endtime = millis();
    if (endtime < starttime) {
      starttime = endtime; // looped around!")
    }
    elapsed += (endtime - starttime);
  }
}
```

```
Adafruit_MQTT_Subscribe *Adafruit_MQTT::readSubscription(int16_t timeout) {

  // Sync or Async subscriber with message
  Adafruit_MQTT_Subscribe *s = 0;

  // Check if are unread messages
  for (uint8_t i = 0; i < MAXSUBSCRIPTIONS; i++) {
    if (subscriptions[i] && subscriptions[i]->new_message) {
      s = subscriptions[i];
      break;
    }
  }

  // not unread message
  if (!s) {
    // Check if data is available to read.
    uint16_t len = readFullPacket(buffer, MAXBUFFERSIZE,
                                  timeout); // return one full packet
    s = handleSubscriptionPacket(len);
  }

  // it there is a message, mark it as not pending
  if (s) {
    s->new_message = false;
  }

  return s;
}

Adafruit_MQTT_Subscribe  *Adafruit_MQTT::handleSubscriptionPacket(uint16_t  len)
{
  uint16_t i, topiclen, datalen;

  if (!len) {
    return NULL; // No data available, just quit.
  }
  DEBUG_PRINT("Packet len: ");
  DEBUG_PRINTLN(len);
  DEBUG_PRINTBUFFER(buffer, len);

  if (len < 3) {
    return NULL;
  }
  if ((buffer[0] & 0xF0) != (MQTT_CTRL_PUBLISH) << 4) {
```

```
  return NULL;
}

// Parse out length of packet.
uint16_t const topicoffset = packetAdditionalLen(len);
uint16_t const topicstart = topicoffset + 4;
topiclen = buffer[3 + topicoffset];
DEBUG_PRINT(F("Looking for subscription len "));
DEBUG_PRINTLN(topiclen);

// Find subscription associated with this packet.
for (i = 0; i < MAXSUBSCRIPTIONS; i++) {
  if (subscriptions[i]) {
    // Skip this subscription if its name length isn't the same as the
    // received topic name.
    if (strlen(subscriptions[i]->topic) != topiclen)
      continue;
    // Stop if the subscription topic matches the received topic. Be careful
    // to make comparison case insensitive.
    if (strncasecmp((char *)buffer + topicstart, subscriptions[i]->topic,
                    topiclen) == 0) {
      DEBUG_PRINT(F("Found sub #"));
      DEBUG_PRINTLN(i);
      if (subscriptions[i]->new_message) {
        DEBUG_PRINTLN(F("Lost previous message"));
      } else {
        subscriptions[i]->new_message = true;
      }

      break;
    }
  }
}
if (i == MAXSUBSCRIPTIONS)
  return NULL; // matching sub not found ???

uint8_t packet_id_len = 0;
uint16_t packetid = 0;
// Check if it is QoS 1, TODO: we dont support QoS 2
if ((buffer[0] & 0x6) == 0x2) {
  packet_id_len = 2;
  packetid = buffer[topiclen + topicstart];
  packetid <<= 8;
  packetid |= buffer[topiclen + topicstart + 1];
  }

  // zero out the old data
  memset(subscriptions[i]->lastread, 0, SUBSCRIPTIONDATALEN);

  datalen = len - topiclen - packet_id_len - topicstart;
  if (datalen > SUBSCRIPTIONDATALEN) {
    datalen = SUBSCRIPTIONDATALEN - 1; // cut it off
  }
  // extract out just the data, into the subscription object itself
  memmove(subscriptions[i]->lastread,
          buffer + topicstart + topiclen + packet_id_len, datalen);
  subscriptions[i]->datalen = datalen;
  DEBUG_PRINT(F("Data len: "));
  DEBUG_PRINTLN(datalen);
  DEBUG_PRINT(F("Data: "));
  DEBUG_PRINTLN((char *)subscriptions[i]->lastread);

  if ((MQTT_PROTOCOL_LEVEL > 3) && (buffer[0] & 0x6) == 0x2) {
    uint8_t ackpacket[4];

    // Construct and send puback packet.
    uint8_t len = pubackPacket(ackpacket, packetid);
    if (!sendPacket(ackpacket, len))
      DEBUG_PRINT(F("Failed"));
  }

  // return the valid matching subscription
  return subscriptions[i];
}

void Adafruit_MQTT::flushIncoming(uint16_t timeout) {
  // flush input!
  DEBUG_PRINTLN(F("Flushing input buffer"));
  while (readPacket(buffer, MAXBUFFERSIZE, timeout))
    ;
}

bool Adafruit_MQTT::ping(uint8_t num) {
  // flushIncoming(100);

  while (num--) {
    // Construct and send ping packet.
    uint8_t len = pingPacket(buffer);
    if (!sendPacket(buffer, len))
      continue;

    // Process ping reply.
    len          =          processPacketsUntil(buffer,          MQTT_CTRL_PINGRESP,
PING_TIMEOUT_MS);
    if (buffer[0] == (MQTT_CTRL_PINGRESP << 4))
      return true;
  }

  return false;
}

// Packet Generation Functions /////////////////////////////////////////////////

// The current MQTT spec is 3.1.1 and available here:
//   http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718028
// However this connect packet and code follows the MQTT 3.1 spec here (some
// small differences in the protocol):
//   http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html#connect
uint8_t Adafruit_MQTT::connectPacket(uint8_t *packet) {
  uint8_t *p = packet;
  uint16_t len;

  // fixed header, connection messsage no flags
  p[0] = (MQTT_CTRL_CONNECT << 4) | 0x0;
  p += 2;
  // fill in packet[1] last

#if MQTT_PROTOCOL_LEVEL == 3
  p = stringprint(p, "MQIsdp");
#elif MQTT_PROTOCOL_LEVEL == 4
  p = stringprint(p, "MQTT");
#else
#error "MQTT level not supported"
#endif

  p[0] = MQTT_PROTOCOL_LEVEL;
  p++;

  // always clean the session
  p[0] = MQTT_CONN_CLEANSESSION;

  // set the will flags if needed
```

```
if (will_topic && pgm_read_byte(will_topic) != 0) {

  p[0] |= MQTT_CONN_WILLFLAG;

  if (will_qos == 1)
    p[0] |= MQTT_CONN_WILLQOS_1;
  else if (will_qos == 2)
    p[0] |= MQTT_CONN_WILLQOS_2;

  if (will_retain == 1)
    p[0] |= MQTT_CONN_WILLRETAIN;
}

if (pgm_read_byte(username) != 0)
  p[0] |= MQTT_CONN_USERNAMEFLAG;
if (pgm_read_byte(password) != 0)
  p[0] |= MQTT_CONN_PASSWORDFLAG;
p++;

p[0] = keepAliveInterval >> 8;
p++;
p[0] = keepAliveInterval & 0xFF;
p++;

if (MQTT_PROTOCOL_LEVEL == 3) {
  p = stringprint(p, clientid, 23); // Limit client ID to first 23 characters.
} else {
  if (pgm_read_byte(clientid) != 0) {
    p = stringprint(p, clientid);
  } else {
    p[0] = 0x0;
    p++;
    p[0] = 0x0;
    p++;
    DEBUG_PRINTLN(F("SERVER GENERATING CLIENT ID"));
  }
}

if (will_topic && pgm_read_byte(will_topic) != 0) {
  p = stringprint(p, will_topic);
  p = stringprint(p, will_payload);
}

if (pgm_read_byte(username) != 0) {
    p = stringprint(p, username);
  }
  if (pgm_read_byte(password) != 0) {
    p = stringprint(p, password);
  }

  len = p - packet;

  packet[1] = len - 2; // don't include the 2 bytes of fixed header data
  DEBUG_PRINTLN(F("MQTT connect packet:"));
  DEBUG_PRINTBUFFER(buffer, len);
  return len;
}

// as per
// http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718040
uint16_t Adafruit_MQTT::publishPacket(uint8_t *packet, const char *topic,
                                     uint8_t *data, uint16_t bLen, uint8_t qos,
                                     uint16_t maxPacketLen, bool retain) {
  uint8_t *p = packet;
  uint16_t len = 0;

  // calc length of non-header data
  len += 2;             // two bytes to set the topic size
  len += strlen(topic); // topic length
  if (qos > 0) {
    len += 2; // qos packet id
  }
  // Calculate additional bytes for length field (if any)
  uint16_t additionalLen = packetAdditionalLen(len + bLen);

  // Payload remaining length. When maxPacketLen provided is 0, let's
  // assume buffer is big enough. Fingers crossed.
  // 2 + additionalLen: header byte + remaining length field (from 1 to 4 bytes)
  // len = topic size field + value (string)
  // bLen = buffer size
  if (!(maxPacketLen == 0 ||
        (len + bLen + 2 + additionalLen <= maxPacketLen))) {
    // If we make it here, we got a pickle: the payload is not going
    // to fit in the packet buffer. Instead of corrupting memory, let's
    // do something less damaging by reducing the bLen to what we are
    // able to accomodate. Alternatively, consider using a bigger
    // maxPacketLen.
    bLen = maxPacketLen - (len + 2 + packetAdditionalLen(maxPacketLen));
  }
  len += bLen; // remaining len excludes header byte & length field

  // Now you can start generating the packet!
  p[0] = MQTT_CTRL_PUBLISH << 4 | qos << 1 | (retain ? 1 : 0);
  p++;

  // fill in packet[1] last
  do {
    uint8_t encodedByte = len % 128;
    len /= 128;
    // if there are more data to encode, set the top bit of this byte
    if (len > 0) {
      encodedByte |= 0x80;
    }
    p[0] = encodedByte;
    p++;
  } while (len > 0);

  // topic comes before packet identifier
  p = stringprint(p, topic);

  // add packet identifier. used for checking PUBACK in QOS > 0
  if (qos > 0) {
    p[0] = (packet_id_counter >> 8) & 0xFF;
    p[1] = packet_id_counter & 0xFF;
    p += 2;

    // increment the packet id
    packet_id_counter++;
  }

  memmove(p, data, bLen);
  p += bLen;
  len = p - packet;
  DEBUG_PRINTLN(F("MQTT publish packet:"));
  DEBUG_PRINTBUFFER(buffer, len);
  return len;
}

uint8_t Adafruit_MQTT::subscribePacket(uint8_t *packet, const char *topic,
                                        uint8_t qos) {
  uint8_t *p = packet;
  uint16_t len;
  p[0] = MQTT_CTRL_SUBSCRIBE << 4 | MQTT_QOS_1 << 1;
  // fill in packet[1] last
  p += 2;

  // packet identifier. used for checking SUBACK
  p[0] = (packet_id_counter >> 8) & 0xFF;
  p[1] = packet_id_counter & 0xFF;
  p += 2;

  // increment the packet id
  packet_id_counter++;

  p = stringprint(p, topic);

  p[0] = qos;
  p++;

  len = p - packet;
  packet[1] = len - 2; // don't include the 2 bytes of fixed header data
  DEBUG_PRINTLN(F("MQTT subscription packet:"));
  DEBUG_PRINTBUFFER(buffer, len);
  return len;
}

uint8_t Adafruit_MQTT::unsubscribePacket(uint8_t *packet, const char *topic) {

  uint8_t *p = packet;
  uint16_t len;

  p[0] = MQTT_CTRL_UNSUBSCRIBE << 4 | 0x1;
  // fill in packet[1] last
  p += 2;

  // packet identifier. used for checking UNSUBACK
  p[0] = (packet_id_counter >> 8) & 0xFF;
  p[1] = packet_id_counter & 0xFF;
  p += 2;

  // increment the packet id
  packet_id_counter++;

  p = stringprint(p, topic);
  len = p - packet;
  packet[1] = len - 2; // don't include the 2 bytes of fixed header data
  DEBUG_PRINTLN(F("MQTT unsubscription packet:"));
  DEBUG_PRINTBUFFER(buffer, len);
  return len;
}

uint8_t Adafruit_MQTT::pingPacket(uint8_t *packet) {
  packet[0] = MQTT_CTRL_PINGREQ << 4;
  packet[1] = 0;
  DEBUG_PRINTLN(F("MQTT ping packet:"));
  DEBUG_PRINTBUFFER(buffer, 2);
  return 2;
}

uint8_t Adafruit_MQTT::pubackPacket(uint8_t *packet, uint16_t packetid) {
  packet[0] = MQTT_CTRL_PUBACK << 4;
  packet[1] = 2;
  packet[2] = packetid >> 8;
  packet[3] = packetid;
  DEBUG_PRINTLN(F("MQTT puback packet:"));
  DEBUG_PRINTBUFFER(buffer, 4);
  return 4;
}

uint8_t Adafruit_MQTT::disconnectPacket(uint8_t *packet) {
  packet[0] = MQTT_CTRL_DISCONNECT << 4;
  packet[1] = 0;
  DEBUG_PRINTLN(F("MQTT disconnect packet:"));
  DEBUG_PRINTBUFFER(buffer, 2);
  return 2;
}

// Adafruit_MQTT_Publish Definition ////////////////////////////////////////////

Adafruit_MQTT_Publish::Adafruit_MQTT_Publish(Adafruit_MQTT *mqttserver,
                                             const char *feed, uint8_t q) {
  mqtt = mqttserver;
  topic = feed;
  qos = q;
}

bool Adafruit_MQTT_Publish::publish(int32_t i, bool retain) {
  char payload[12];
  ltoa(i, payload, 10);
  return mqtt->publish(topic, payload, qos, retain);
}

bool Adafruit_MQTT_Publish::publish(uint32_t i, bool retain) {
  char payload[11];
  ultoa(i, payload, 10);
  return mqtt->publish(topic, payload, qos, retain);
}

bool Adafruit_MQTT_Publish::publish(double f, uint8_t precision, bool retain) {
  char payload[41]; // Need to technically hold float max, 39 digits and minus
                    // sign.
  dtostrf(f, 0, precision, payload);
  return mqtt->publish(topic, payload, qos, retain);
}

bool Adafruit_MQTT_Publish::publish(const char *payload, bool retain) {
  return mqtt->publish(topic, payload, qos, retain);
}

// publish buffer of arbitrary length
bool Adafruit_MQTT_Publish::publish(uint8_t *payload, uint16_t bLen,
                                    bool retain) {
  return mqtt->publish(topic, payload, bLen, qos, retain);
}

// Adafruit_MQTT_Subscribe Definition ////////////////////////////////////////

Adafruit_MQTT_Subscribe::Adafruit_MQTT_Subscribe(Adafruit_MQTT *mqttserver,
                                                 const char *feed, uint8_t q) {
  mqtt = mqttserver;
  topic = feed;
  qos = q;
  datalen = 0;
  callback_uint32t = 0;
  callback_buffer = 0;
  callback_double = 0;
  callback_io = 0;
  io_mqtt = 0;
  new_message = false;
}

void Adafruit_MQTT_Subscribe::setCallback(SubscribeCallbackUInt32Type cb) {
  callback_uint32t = cb;
}

void Adafruit_MQTT_Subscribe::setCallback(SubscribeCallbackDoubleType cb) {
  callback_double = cb;
}

void Adafruit_MQTT_Subscribe::setCallback(SubscribeCallbackBufferType cb) {
  callback_buffer = cb;
}

void Adafruit_MQTT_Subscribe::setCallback(AdafruitIO_MQTT *io,
                                          SubscribeCallbackIOType cb) {
  callback_io = cb;
  io_mqtt = io;
}

void Adafruit_MQTT_Subscribe::removeCallback(void) {
  callback_uint32t = 0;
  callback_buffer = 0;
  callback_double = 0;
  callback_io = 0;
  io_mqtt = 0;
}
```

Adafruit_MQTT_Client.cpp

```
// The MIT License (MIT)
//
// Copyright (c) 2015 Adafruit Industries
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE
// SOFTWARE.
#include "Adafruit_MQTT_Client.h"

bool Adafruit_MQTT_Client::connectServer() {
  // Grab server name from flash and copy to buffer for name resolution.
  memset(buffer, 0, sizeof(buffer));
  strcpy((char *)buffer, servername);
  DEBUG_PRINT(F("Connecting to: "));
  DEBUG_PRINTLN((char *)buffer);
  // Connect and check for success (0 result).
  int r = client->connect((char *)buffer, portnum);
  DEBUG_PRINT(F("Connect result: "));
  DEBUG_PRINTLN(r);
  return r != 0;
}

bool Adafruit_MQTT_Client::disconnectServer() {
  // Stop connection if connected and return success (stop has no indication of
  // failure).
  if (client->connected()) {
    client->stop();
  }
  return true;
}

bool Adafruit_MQTT_Client::connected() {
  // Return true if connected, false if not connected.
  return client->connected();
}

uint16_t Adafruit_MQTT_Client::readPacket(uint8_t *buffer, uint16_t maxlen,
                                          int16_t timeout) {
  /* Read data until either the connection is closed, or the idle timeout is
   * reached. */
  uint16_t len = 0;
  int16_t t = timeout;
  if (maxlen == 0) { // handle zero-length packets
    return 0;
  }

  while (client->connected() && (timeout >= 0)) {
    // DEBUG_PRINT('.');
    while (client->available()) {
      // DEBUG_PRINT('!');
      char c = client->read();
      timeout = t; // reset the timeout
      buffer[len] = c;
      // DEBUG_PRINTLN((uint8_t)c, HEX);
      len++;

      if (len == maxlen) { // we read all we want, bail
        DEBUG_PRINT(F("Read data:\t"));
        DEBUG_PRINTBUFFER(buffer, len);
        return len;
      }
    }
    timeout -= MQTT_CLIENT_READINTERVAL_MS;
    delay(MQTT_CLIENT_READINTERVAL_MS);
  }
  return len;
}

bool Adafruit_MQTT_Client::sendPacket(uint8_t *buffer, uint16_t len) {
  uint16_t ret = 0;
  uint16_t offset = 0;
  while (len > 0) {
    if (client->connected()) {
      // send 250 bytes at most at a time, can adjust this later based on Client
      uint16_t sendlen = len > 250 ? 250 : len;
      // Serial.print("Sending: "); Serial.println(sendlen);
      ret = client->write(buffer + offset, sendlen);
      DEBUG_PRINT(F("Client sendPacket returned: "));
      DEBUG_PRINTLN(ret);
      len -= ret;
      offset += ret;

      if (ret != sendlen) {
        DEBUG_PRINTLN("Failed to send packet.");
        return false;
      }
    } else {
      DEBUG_PRINTLN(F("Connection failed!"));
      return false;
    }
  }
  return true;
}
```

3.2.2 NTP Client-Server Protocol

NTP stands for Network Time Protocol and it is a networking protocol for clock synchronization between computer systems. In other words, it is used to synchronize computer clock times in a network.

There are NTP servers like pool.ntp.org that anyone can use to request time as a client. In this case, the ESP8266 is an NTP Client that requests time from an NTP Server.

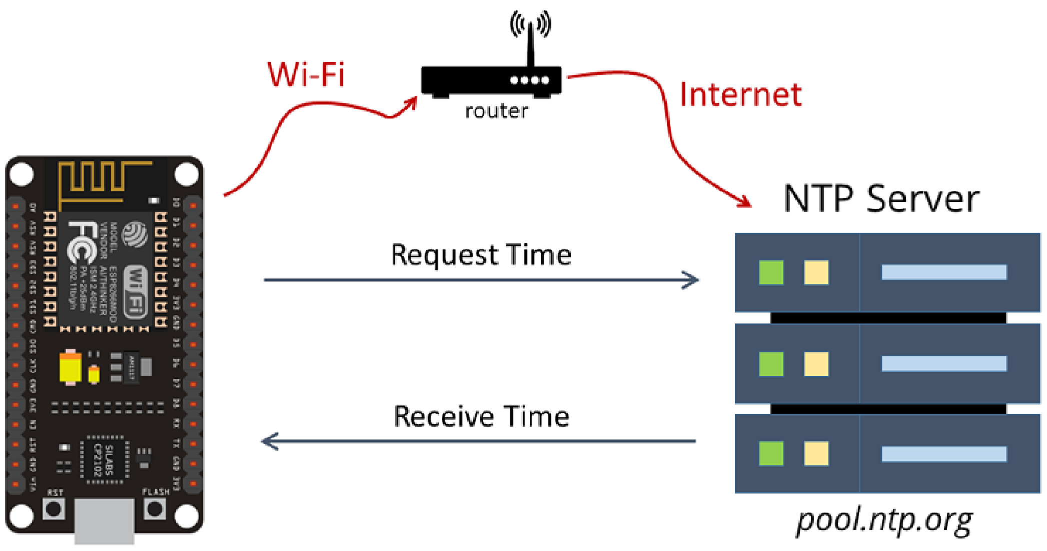


Figure 3.5 NTP Client-Server

NTPClient Library Time Functions
The NTPClient Library comes with the following functions to return time:

getDay() – returns an int number that corresponds to the the week day (0 to 6) starting on Sunday;

getHours() – returns an int number with the current hour (0 to 23) in 24 hour format;

getMinutes() – returns an int number with the current minutes (0 to 59);

getSeconds() – returns an int number with the current second;

getEpochTime() – returns an unsigned long with the epoch time (number of seconds that have elapsed since January 1, 1970 (midnight GMT);

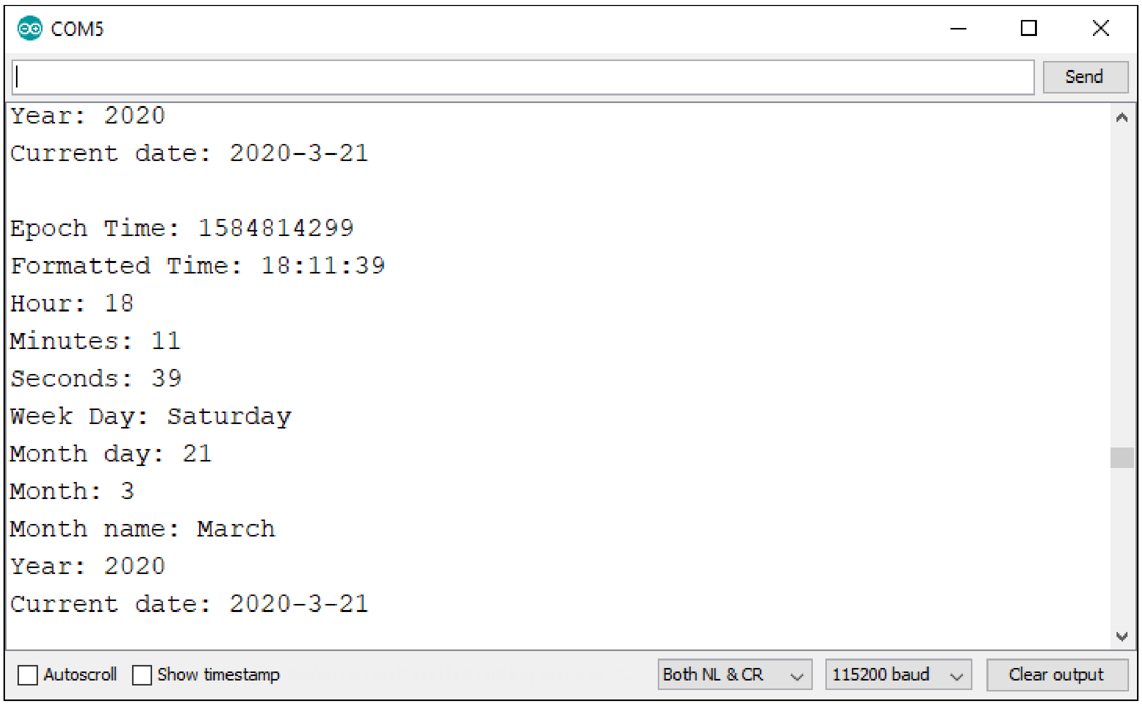
getFormattedTime() – returns a String with the time formatted like HH:MM:SS;

This library doesn't come with functions to return the date, but we'll show you in the code how to get the date (day, month and year).

After inserting your network credentials and modifying the variables to adjust the time to your timezone, test the example.

Upload the code your ESP8266 board. Make sure you have the right board and COM port selected.

Open the Serial Monitor at a baud rate of 115200. The date and time should be displayed in several formats as shown below.



```
Year: 2020
Current date: 2020-3-21

Epoch Time: 1584814299
Formatted Time: 18:11:39
Hour: 18
Minutes: 11
Seconds: 39
Week Day: Saturday
Month day: 21
Month: 3
Month name: March
Year: 2020
Current date: 2020-3-21
```

Figure 3.6 Demonstration

NTP Library Files

NTPClient.cpp

```
/**
 * The MIT License (MIT)
 * Copyright (c) 2015 by Fabrice Weinberg
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
```

```
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE
 * SOFTWARE.
 */

#include "NTPClient.h"

NTPClient::NTPClient(UDP& udp) {
  this->_udp            = &udp;
}

NTPClient::NTPClient(UDP& udp, long timeOffset) {
  this->_udp            = &udp;
  this->_timeOffset     = timeOffset;
}

NTPClient::NTPClient(UDP& udp, const char* poolServerName) {
  this->_udp            = &udp;
  this->_poolServerName = poolServerName;
}

NTPClient::NTPClient(UDP& udp, IPAddress poolServerIP) {
  this->_udp            = &udp;
  this->_poolServerIP   = poolServerIP;
  this->_poolServerName = NULL;
}

NTPClient::NTPClient(UDP& udp, const char* poolServerName, long timeOffset) {
  this->_udp            = &udp;
  this->_timeOffset     = timeOffset;
  this->_poolServerName = poolServerName;
}
```

```
NTPClient::NTPClient(UDP& udp, IPAddress poolServerIP, long timeOffset){
  this->_udp            = &udp;
  this->_timeOffset     = timeOffset;
  this->_poolServerIP   = poolServerIP;
  this->_poolServerName = NULL;
}

NTPClient::NTPClient(UDP& udp, const char* poolServerName, long timeOffset,
unsigned long updateInterval) {
  this->_udp            = &udp;
  this->_timeOffset     = timeOffset;
  this->_poolServerName = poolServerName;
  this->_updateInterval = updateInterval;
}

NTPClient::NTPClient(UDP& udp, IPAddress poolServerIP, long timeOffset, unsigned
long updateInterval) {
  this->_udp            = &udp;
  this->_timeOffset     = timeOffset;
  this->_poolServerIP   = poolServerIP;
  this->_poolServerName = NULL;
  this->_updateInterval = updateInterval;
}

void NTPClient::begin() {
  this->begin(NTP_DEFAULT_LOCAL_PORT);
}

void NTPClient::begin(unsigned int port) {
  this->_port = port;

  this->_udp->begin(this->_port);

  this->_udpSetup = true;
}

bool NTPClient::forceUpdate() {
  #ifdef DEBUG_NTPClient
    Serial.println("Update from NTP Server");
  #endif

  // flush any existing packets
  while(this->_udp->parsePacket() != 0)
```

```
    this->_udp->flush();

  this->sendNTPPacket();

  // Wait till data is there or timeout...
  byte timeout = 0;
  int cb = 0;
  do {
    delay ( 10 );
    cb = this->_udp->parsePacket();
    if (timeout > 100) return false; // timeout after 1000 ms
    timeout++;
  } while (cb == 0);

  this->_lastUpdate = millis() - (10 * (timeout + 1)); // Account for delay in reading the
time

  this->_udp->read(this->_packetBuffer, NTP_PACKET_SIZE);

  unsigned long highWord = word(this->_packetBuffer[40], this->_packetBuffer[41]);
  unsigned long lowWord = word(this->_packetBuffer[42], this->_packetBuffer[43]);
  // combine the four bytes (two words) into a long integer
  // this is NTP time (seconds since Jan 1 1900):
  unsigned long secsSince1900 = highWord << 16 | lowWord;

  this->_currentEpoc = secsSince1900 - SEVENZYYEARS;

  return true;  // return true after successful update
}

bool NTPClient::update() {
  if ((millis() - this->_lastUpdate >= this->_updateInterval)     // Update after
_updateInterval
    || this->_lastUpdate == 0) {                                // Update if there was no update yet.
    if (!this->_udpSetup || this->_port != NTP_DEFAULT_LOCAL_PORT) this-
>begin(this->_port); // setup the UDP client if needed
    return this->forceUpdate();
  }
  return false;   // return false if update does not occur
}

bool NTPClient::isTimeSet() const {
  return (this->_lastUpdate != 0); // returns true if the time has been set, else false
}
```

```
unsigned long NTPClient::getEpochTime() const {
  return this->_timeOffset + // User offset
         this->_currentEpoc + // Epoch returned by the NTP server
         ((millis() - this->_lastUpdate) / 1000); // Time since last update
}

int NTPClient::getDay() const {
  return (((this->getEpochTime()  / 86400L) + 4 ) % 7); //0 is Sunday
}
int NTPClient::getHours() const {
  return ((this->getEpochTime()  % 86400L) / 3600);
}
int NTPClient::getMinutes() const {
  return ((this->getEpochTime() % 3600) / 60);
}
int NTPClient::getSeconds() const {
  return (this->getEpochTime() % 60);
}

String NTPClient::getFormattedTime() const {
  unsigned long rawTime = this->getEpochTime();
  unsigned long hours = (rawTime % 86400L) / 3600;
  String hoursStr = hours < 10 ? "0" + String(hours) : String(hours);

  unsigned long minutes = (rawTime % 3600) / 60;
  String minuteStr = minutes < 10 ? "0" + String(minutes) : String(minutes);

  unsigned long seconds = rawTime % 60;
  String secondStr = seconds < 10 ? "0" + String(seconds) : String(seconds);

  return hoursStr + ":" + minuteStr + ":" + secondStr;
}

void NTPClient::end() {
  this->_udp->stop();

  this->_udpSetup = false;
}

void NTPClient::setTimeOffset(int timeOffset) {
  this->_timeOffset     = timeOffset;
}
```

```
void NTPClient::setUpdateInterval(unsigned long updateInterval) {
  this->_updateInterval = updateInterval;
}

void NTPClient::setPoolServerName(const char* poolServerName) {
    this->_poolServerName = poolServerName;
}

void NTPClient::sendNTPPacket() {
  // set all bytes in the buffer to 0
  memset(this->_packetBuffer, 0, NTP_PACKET_SIZE);
  // Initialize values needed to form NTP request
  this->_packetBuffer[0] = 0b11100011;   // LI, Version, Mode
  this->_packetBuffer[1] = 0;     // Stratum, or type of clock
  this->_packetBuffer[2] = 6;     // Polling Interval
  this->_packetBuffer[3] = 0xEC;  // Peer Clock Precision
  // 8 bytes of zero for Root Delay & Root Dispersion
  this->_packetBuffer[12]  = 49;
  this->_packetBuffer[13]  = 0x4E;
  this->_packetBuffer[14]  = 49;
  this->_packetBuffer[15]  = 52;

  // all NTP fields have been given values, now
  // you can send a packet requesting a timestamp:
  if  (this->_poolServerName) {
    this->_udp->beginPacket(this->_poolServerName, 123);
  } else {
    this->_udp->beginPacket(this->_poolServerIP, 123);
  }
  this->_udp->write(this->_packetBuffer, NTP_PACKET_SIZE);
  this->_udp->endPacket();
}

void NTPClient::setRandomPort(unsigned int minValue, unsigned int maxValue) {
  randomSeed(analogRead(0));
  this->_port = random(minValue, maxValue);
}
```

3.3 GENERAL OVERVIEW

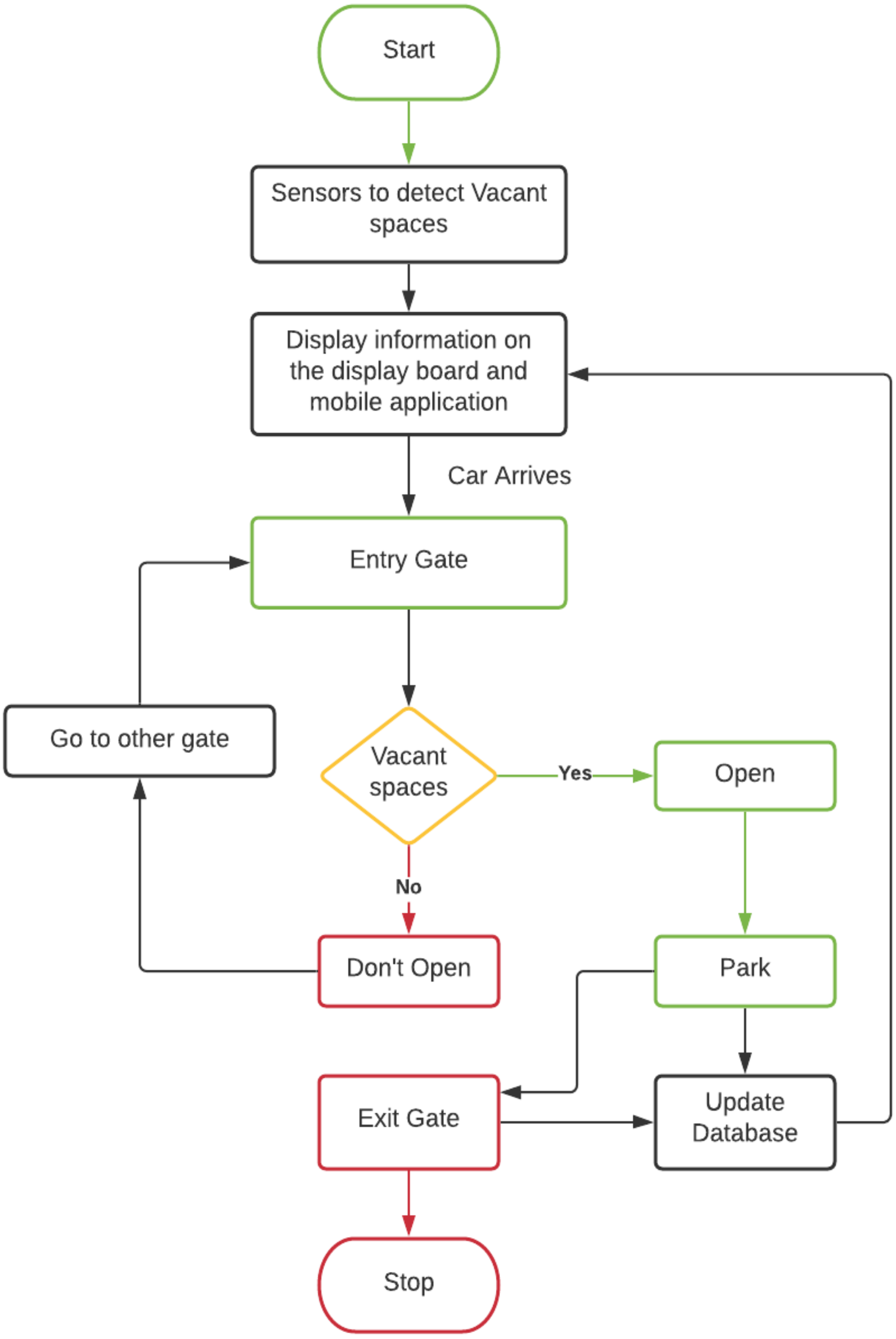


Figure 3.7 General View

3.4 USE CASE DIAGRAM

Use-case diagrams model the behavior of a system and help to capture the requirements of the system. Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors.

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high level functionality of a system and also tells how the user handles a system.

Purposes of a use case diagram given below:
1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

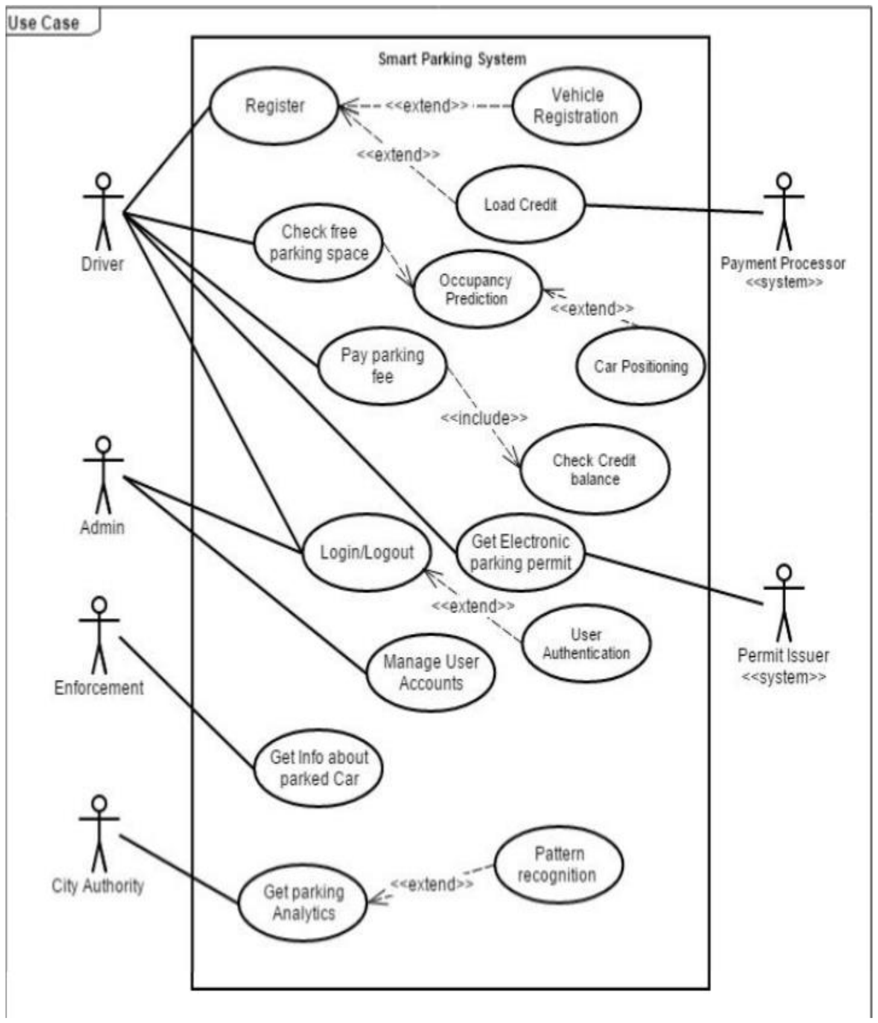


Figure 3.8 Use Case Diagram

3.5 Architecture

3.5.1 Application Layer

The application layer is the top layer of the architecture stack that allows the participants to interact with the system that they use, the mobile application (i.e., Android and iOS), or the Web application. Similarly, the parking services provider can send parking-related information, e.g., parking space availability, to the providers and the offers to the integrated systems. Since the users interact with the integrated system directly, the layer delivers the end-users' final service.

3.5.2 Network Layer

The network layer ensures seamless communication among the various parking centers, integrated systems, and users. The user and parking center data are transmitted to the integrated system through a layer. The layer contains the different types of communication technologies that may include LAN and WAN, which are used by the users, parking service providers, and the IoT devices related to the parking systems (e.g., the parking sensors and the security camera). They may contain different wireless technologies that include Bluetooth, WI-FI, etc., which, along with the existing GSM technologies, exist as 4G and 5G.

3.5.3 Transaction Layer

This is the layer that is mandated to transact the nodes in the network. The users and the various parking centers exchange the data more securely through the smart contract and the consensus mechanisms. The parking center also updates the public ledger through the layer. The transaction layer preserves the transparent quality of the transaction and the security of the data transmission without trusted third parties.

3.5.4 Physical Layers

The physical layers deal specifically with the mechanisms and the electronic anchorage of the system. The physical layer is based on the set of the physical sensors and the data received from the entities collected that are analyzed and used to manage the entities. The different types of sensors are the significant elements of the layer. The use of IoT device sensors can be recognized through the availability.

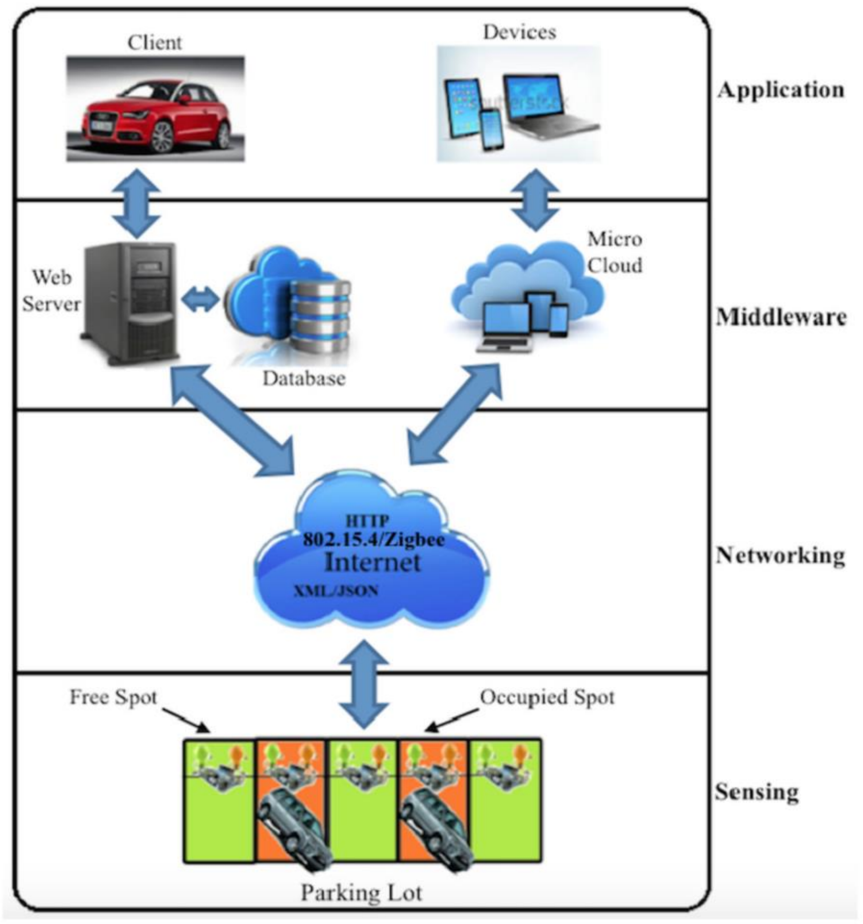


Figure 3.9 Layered Architecture

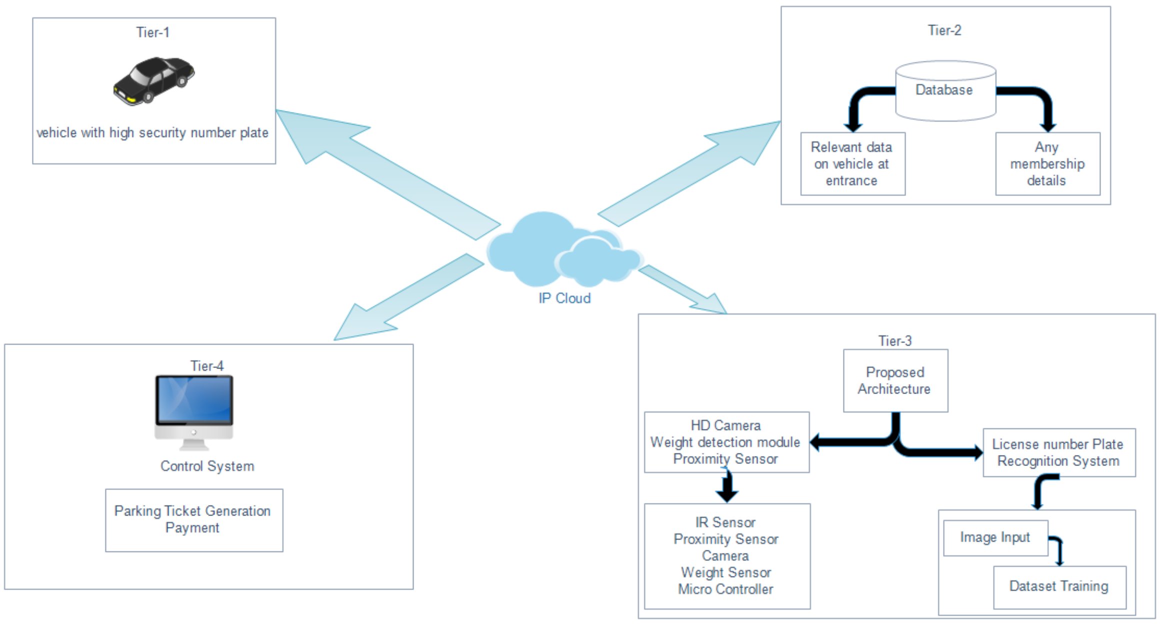

Tier-1
vehicle with high security number plate
Tier-2
Database
Relevant data on vehicle at entrance
Any membership details
IP Cloud
Tier-4
Control System
Parking Ticket Generation Payment
Tier-3
Proposed Architecture
HD Camera Weight detection module Proximity Sensor
License number Plate Recognition System
IR Sensor Proximity Sensor Camera Weight Sensor Micro Controller
Image Input
Dataset Training

Figure 3.10 4-Tier Architecture for cloud based Architecture

3.6 CIRCUIT DIAGRAM

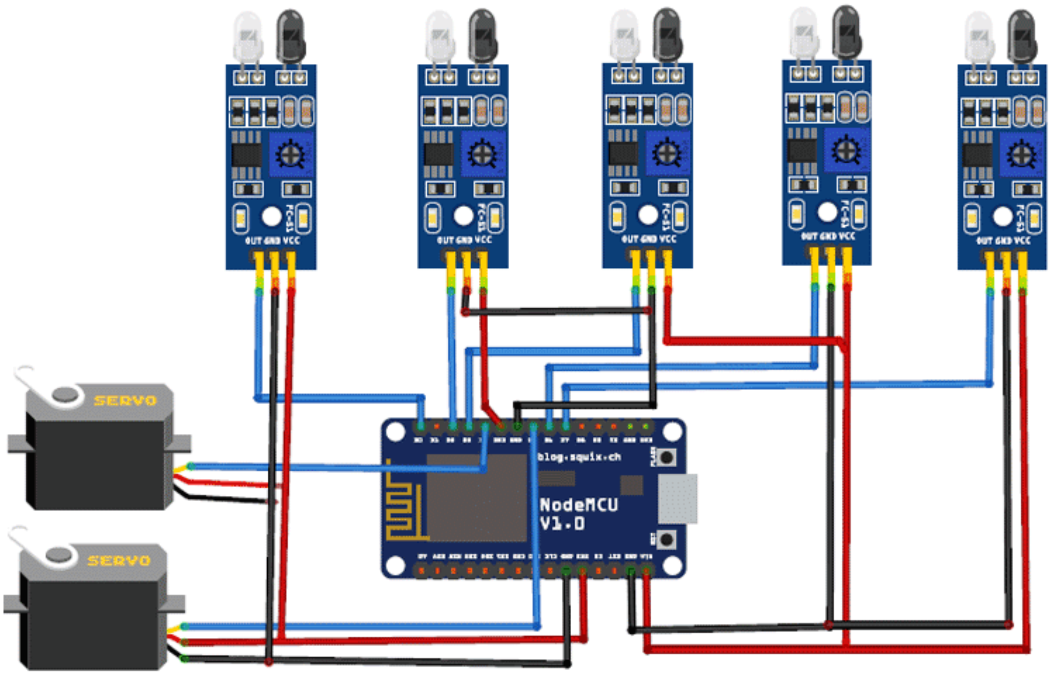


Figure 3.11 Circuit Diagram

3.7 BLOCK DIAGRAM

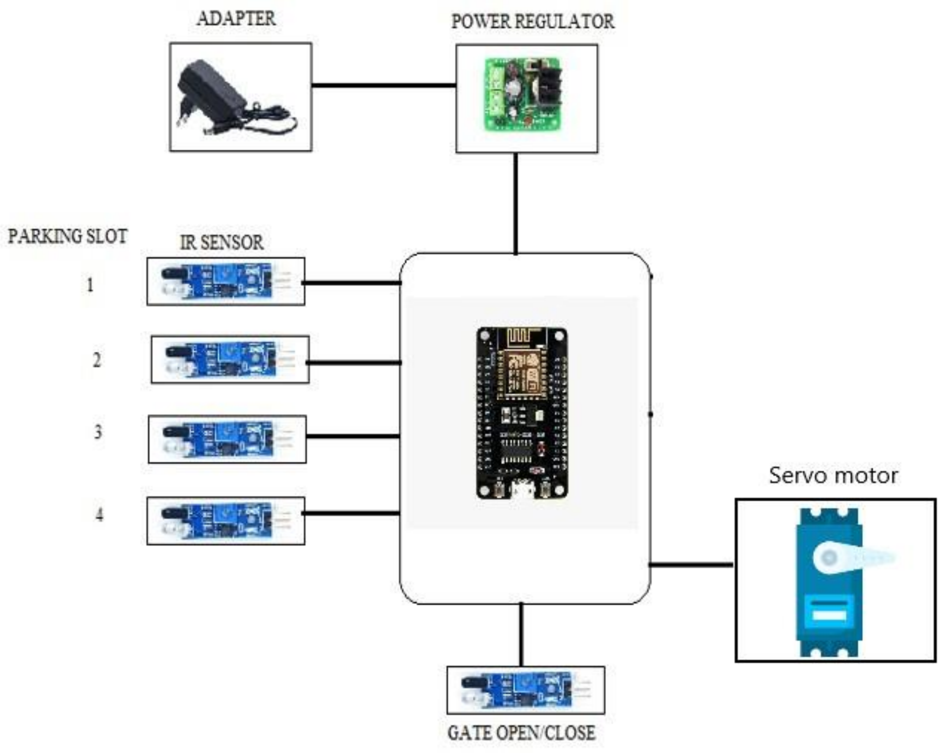


Figure 3.12 Block Diagram

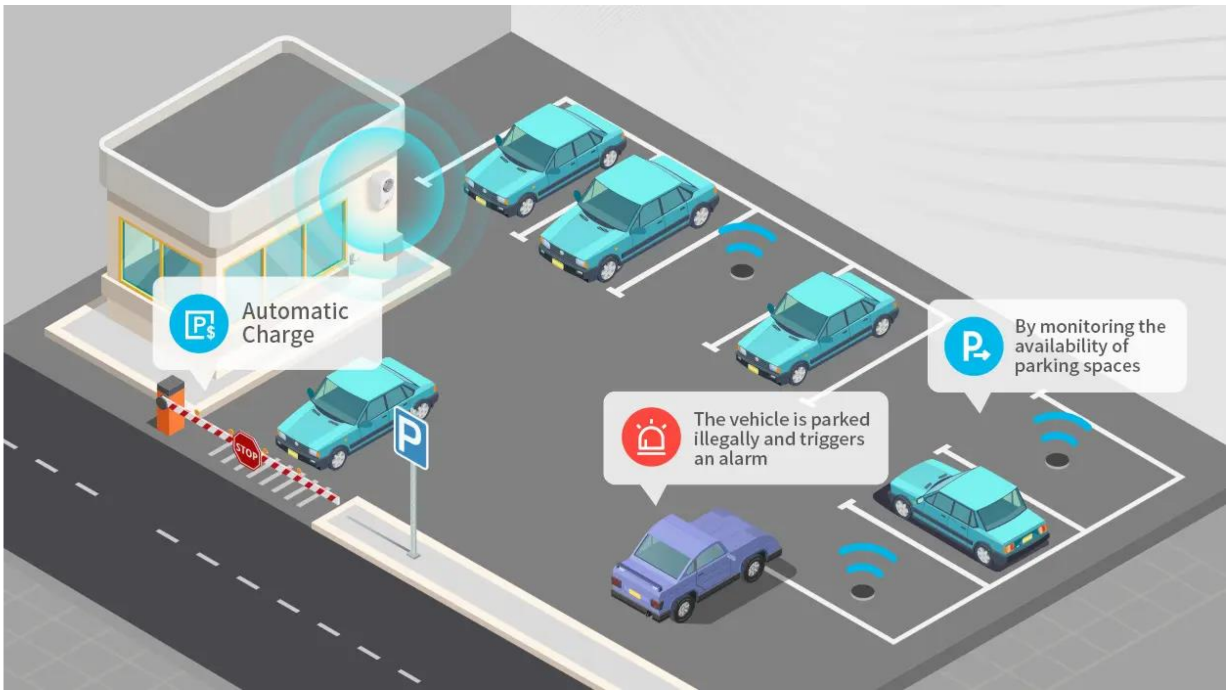

Automatic Charge
The vehicle is parked illegally and triggers an alarm
By monitoring the availability of parking spaces
STOP

Figure 3.13 Working View

CHAPTER 4

CODING

Main.io
(this is operation code)

```
#include <ESP8266WiFi.h>
#include <Servo.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
const char *ssid =  "Amit1";     // Enter your WiFi Name
const char *pass =  "123456789"; // Enter your WiFi Password

#define MQTT_SERV "io.adafruit.com"
#define MQTT_PORT 1883
#define MQTT_NAME "amitrajput1104"
#define MQTT_PASS "aio_qiLh04qWBAWxlsqfMuXs5q5pI2Uy"
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", 19800,60000);
Servo myservo;                    //servo as gate
Servo myservos;                          //servo as gate

int carEnter = D0;                // entry sensor
int carExited = D2;              //exi sensor
int slot3 = D7;
int slot2 = D6;
int slot1  = D3;
int count =0;
int CLOSE_ANGLE = 80;  // The closing angle of the servo motor arm
int OPEN_ANGLE = 0;  // The opening angle of the servo motor arm
int  hh, mm, ss;
int pos;
int pos1;

String h, m,EntryTimeSlot1,ExitTimeSlot1, EntryTimeSlot2,ExitTimeSlot2,
EntryTimeSlot3,ExitTimeSlot3;
boolean entrysensor, exitsensor,s1,s2,s3;
boolean s1_occupied = false;
boolean s2_occupied = false;
boolean s3_occupied = false;

WiFiClient client;
Adafruit_MQTT_Client mqtt(&client, MQTT_SERV, MQTT_PORT, MQTT_NAME,
MQTT_PASS);

//Set up the feed you're subscribing to
Adafruit_MQTT_Subscribe EntryGate = Adafruit_MQTT_Subscribe(&mqtt,
MQTT_NAME "/f/EntryGate");
Adafruit_MQTT_Subscribe ExitGate = Adafruit_MQTT_Subscribe(&mqtt,
MQTT_NAME "/f/ExitGate");

//Set up the feed you're publishing to
Adafruit_MQTT_Publish CarsParked =
Adafruit_MQTT_Publish(&mqtt,MQTT_NAME "/f/CarsParked");
Adafruit_MQTT_Publish EntrySlot1 = Adafruit_MQTT_Publish(&mqtt,MQTT_NAME
"/f/EntrySlot1");
Adafruit_MQTT_Publish ExitSlot1 = Adafruit_MQTT_Publish(&mqtt,MQTT_NAME
"/f/ExitSlot1");
Adafruit_MQTT_Publish EntrySlot2 = Adafruit_MQTT_Publish(&mqtt,MQTT_NAME
"/f/EntrySlot2");
Adafruit_MQTT_Publish ExitSlot2 = Adafruit_MQTT_Publish(&mqtt,MQTT_NAME
"/f/ExitSlot2");
Adafruit_MQTT_Publish EntrySlot3 = Adafruit_MQTT_Publish(&mqtt,MQTT_NAME
"/f/EntrySlot3");
Adafruit_MQTT_Publish ExitSlot3 = Adafruit_MQTT_Publish(&mqtt,MQTT_NAME
"/f/ExitSlot3");

void setup() {
  delay(1000);
  Serial.begin (9600);
  mqtt.subscribe(&EntryGate);
  mqtt.subscribe(&ExitGate);
  timeClient.begin();
  myservo.attach(D4);      // servo pin to D6
  myservos.attach(D5);       // servo pin to D5
  pinMode(carExited, INPUT);    // ir as input
  pinMode(carEnter, INPUT);     // ir as input
  pinMode(slot1, INPUT);
  pinMode(slot2, INPUT);
  pinMode(slot3, INPUT);
  WiFi.begin(ssid, pass);                              //try to connect with wifi
  Serial.print("Connecting to ");
  Serial.print(ssid);                    // display ssid
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");                   // if not connected print this
    delay(500);
  }
  Serial.println();
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP Address is : ");
  Serial.println(WiFi.localIP());                          //print local IP address
}

void loop() {

 MQTT_connect();
 timeClient.update();
 hh = timeClient.getHours();
 mm = timeClient.getMinutes();
 ss = timeClient.getSeconds();
 h= String(hh);
 m= String(mm);
 h +" :" + m;

 entrysensor= !digitalRead(carEnter);
 exitsensor = !digitalRead(carExited);
 s1 = digitalRead(slot1);
 s2 = digitalRead(slot2);
 s3 = digitalRead(slot3);

   if (entrysensor == 1) {              // if high then count and send data
   count=  count+1;                     //increment count
   myservos.write(OPEN_ANGLE);
   delay(3000);
   myservos.write(CLOSE_ANGLE);
   }

   if (exitsensor == 1) {                   //if high then count and send
   count= count-1;                          //decrement count
   myservo.write(OPEN_ANGLE);
   delay(3000);
   myservo.write(CLOSE_ANGLE);
  }
  if (! CarsParked.publish(count)) {}
  if (s1 == 1 && s1_occupied == false) {
       Serial.println("Occupied1 ");
       EntryTimeSlot1 =  h +" :" + m;
       //Serial.print("EntryTimeSlot1");
       //Serial.print(EntryTimeSlot1);

       s1_occupied = true;
       if (! EntrySlot1.publish((char*) EntryTimeSlot1.c_str())){}
    }
  if(s1 == 0 && s1_occupied == true) {
      Serial.println("Available1 ");
      ExitTimeSlot1 =  h +" :" + m;
      //Serial.print("ExitTimeSlot1");
      //Serial.print(ExitTimeSlot1);

      s1_occupied = false;
       if (! ExitSlot1.publish((char*) ExitTimeSlot1.c_str())){}
}
  if (s2 == 1&& s2_occupied == false) {
     Serial.println("Occupied2 ");
     EntryTimeSlot2 =  h +" :" + m;
     //Serial.print("EntryTimeSlot2");
     //Serial.print(EntryTimeSlot2);

     s2_occupied = true;
     if (! EntrySlot2.publish((char*) EntryTimeSlot2.c_str())){}
    }
  if(s2 == 0 && s2_occupied == true) {
      Serial.println("Available2 ");
      ExitTimeSlot2 =  h +" :" + m;
      //Serial.print("ExitTimeSlot2");
      //Serial.print(ExitTimeSlot2);

      s2_occupied = false;
       if (! ExitSlot2.publish((char*) ExitTimeSlot2.c_str())){}
  }
  if (s3 == 1&& s3_occupied == false) {
     Serial.println("Occupied3 ");
     EntryTimeSlot3 =  h +" :" + m;
    //Serial.print("EntryTimeSlot3: ");
     //Serial.print(EntryTimeSlot3);
     s3_occupied = true;
      if (! EntrySlot3.publish((char*) EntryTimeSlot3.c_str())){}
    }
  if(s3 == 0 && s3_occupied == true) {
       Serial.println("Available3 ");
       ExitTimeSlot3 =  h +" :" + m;
       //Serial.print("ExitTimeSlot3: ");
       //Serial.print(ExitTimeSlot3);
       s3_occupied = false;
        if (! ExitSlot3.publish((char*) ExitTimeSlot3.c_str())){ }
  }

  Adafruit_MQTT_Subscribe * subscription;
  while ((subscription = mqtt.readSubscription(5000)))
    {

   if (subscription == &EntryGate)
    {
     //Print the new value to the serial monitor
     Serial.println((char*) EntryGate.lastread);

   if (!strcmp((char*) EntryGate.lastread, "ON"))
      {
       myservos.write(OPEN_ANGLE);
       delay(3000);
       myservos.write(CLOSE_ANGLE);
    }
}
  if (subscription == &ExitGate)
    {
     //Print the new value to the serial monitor
     Serial.println((char*) EntryGate.lastread);

   if (!strcmp((char*) ExitGate.lastread, "ON"))
      {
       myservo.write(OPEN_ANGLE);
       delay(3000);
       myservo.write(CLOSE_ANGLE);
    }
}
}
}
void MQTT_connect()
{
  int8_t ret;
  // Stop if already connected.
  if (mqtt.connected())
  {
    return;
  }
  uint8_t retries = 3;
  while ((ret = mqtt.connect()) != 0) // connect will return 0 for connected
  {
       mqtt.disconnect();
       delay(5000);  // wait 5 seconds
       retries--;
       if (retries == 0)
       {
         // basically die and wait for WDT to reset me
         while (1);
       }
  }
}
```

CHAPTER 5

TESTING

5.1 TESTING

Testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understandthe risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects), and verifying that the software product is fit for use. Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- Meets the requirements that guided its design and development,
- Responds correctly to all kinds of inputs,
- Performs its functions within an acceptable time,
- It is sufficiently usable,
- Can be installed and run in its intended environments, andAchieves the general result its stakeholder's desire.

5.1.1 Software Testing

Software testing for a smart parking system involves a comprehensive and systematic evaluation of the system's functionalities, performance, and user experience. It ensures that the system operates as intended, meets the specified requirements, and provides a reliable and seamless parking management solution. The testing process typically includes the following aspects:

5.1.1.1 Functional Testing:

This type of testing verifies that the smart parking system functions correctly and performs its intended operations. It includes validating features such as parking availability display, reservation, payment processing, ticketing, and integration with entry and exit barriers.

5.1.1.2 Performance Testing:

Performance testing assesses the system's responsiveness, scalability, and stability under varying load conditions. It examines factors like response times, system capacity, concurrent user handling, and stress testing to ensure that the system can handle the expected parking demand without performance degradation.

5.1.1.3 Usability Testing:

Usability testing focuses on evaluating the user-friendliness and intuitiveness of the smart parking system. It involves assessing the ease of navigation, clarity of user interfaces, effectiveness of error handling, and overall user experience. Usability testing helps ensure that the system is accessible and straightforward for both administrators and end users.

5.1.1.4 Security Testing:

Security testing is essential to identify vulnerabilities and potential risks to the smart parking system. It includes measures to prevent unauthorized access, protect user data, and ensure data integrity. Security testing may involve techniques like penetration testing, vulnerability scanning, and encryption validation to safeguard the system from potential threats.

5.1.1.5 Integration Testing:

Integration testing verifies the smooth integration and interoperability of the smart parking system with other related systems or components. This includes testing the integration with payment gateways, entry/exit barriers, mobile applications, and any other external interfaces.

5.1.1.6 Regression Testing:

Regression testing ensures that system updates, enhancements, or bug fixes do not introduce new issues or disrupt existing functionalities. It involves retesting previously tested features to ensure their continued functionality.

By conducting thorough software testing for the smart parking system, stakeholders can identify and rectify any issues, validate system performance, and ensure a reliable and user-friendly experience. It helps in building confidence in the system's capabilities and contributes to the overall success and customer satisfaction of the smart parking solution.

5.1.2 Hardware Testing

Hardware testing for a smart parking system involves evaluating the functionality, performance, and reliability of the physical components and devices that comprise the system. It ensures that the hardware elements work seamlessly and meet the requirements of the smart parking solution. The hardware testing process typically includes the following aspects:

5.1.2.1 Sensor Testing:

Sensors play a crucial role in detecting the presence or absence of vehicles in parking spaces. Sensor testing involves validating the accuracy and responsiveness of the sensors in detecting vehicles and providing real-time data on parking availability. This testing ensures that the sensors function correctly and provide reliable information for parking management.

5.1.2.2 Communication Testing:

The smart parking system relies on communication networks to transmit data between the various hardware components and the central control system. Communication testing ensures the stability and efficiency of data transmission, verifying that the communication channels are reliable and can handle the expected traffic volume.

5.1.2.3 Barrier/Gate Testing:

Entry and exit barriers or gates are integral components of a smart parking system. Testing these hardware elements ensures their smooth operation, including opening and closing mechanisms, response times, and integration with the control system. Barrier testing also includes validating functionalities like ticket dispensing and payment validation.

5.1.2.4 Power and Backup Testing:

The hardware components of the smart parking system should be tested for their power requirements and backup capabilities. This includes verifying the stability of power supply, testing backup power sources like batteries or generators, and ensuring seamless transitions during power disruptions.

5.2 Static vs Dynamic Testing

There are many approaches available in software testing. Reviews, walkthroughs, or inspections are referred to as static testing, whereas executing programmed code with a given set of test cases is referred to as dynamic testing.

Static testing is often implicit, like proofreading, plus when programming tools/text editors check source code structure or compilers (pre-compilers) check syntax and data flow as static program analysis. Dynamic testing takes place when the program itself is run. Dynamic testingmay begin before the program is 100% complete in order to test particular sections of code andare applied to discrete functions or modules. Typical techniques for these are either using stubs/drivers or execution from a debugger environment.

5.2.1 White-Box testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing and structural testing) verifies the internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing, an internal perspective of the system (the source code), as well as programming skills, are used to designtest cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g., in-circuit testing (ICT).

While white-box testing can be applied at the unit, integration, and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, pathsbetween units during integration, and between subsystems during a system–level test. Thoughthis method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.
Techniques used in white-box testing include:

- API testing – testing of the application using public and private APIs (application programming interfaces)

- Code coverage – creating tests to satisfy some criteria of code coverage (e.g., the testdesigner can create tests to cause all statements in the program to be executed at least once)

- Fault injection methods – intentionally introducing faults to gauge the efficacy of testing strategies.
- Mutation testing methods

- Static testing methods

Code coverage tools can evaluate the completeness of a test suite that was created with any method, including black-box testing. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested. Code coverage as a software metric can be reported as a percentage for:

- Function coverage, which reports on functions executed
- Statement coverage, which reports on the number of lines executed to complete the test
- Decision coverage, which reports on whether both the True and the False branch of a given tens.
- it has been executed

100% statement coverage ensures that all code paths or branches (in terms of control flow) are executedat least once. This is helpful in ensuring correct functionality, but not sufficient since the same code may process different inputs correctly or incorrectly. Pseudo-tested functions and methods are those that arecovered but not specified (it is possible to remove their body without breaking any test case).

5.2.2 Black-Box testing

Input
Blackbox
Output

Figure 5.1 Black-Box Testing

Black-box testing (also known as functional testing) treats the software as a "black box," examining functionality without any knowledge of internal implementation, without seeing thesource code. The testers are only aware of what the software is supposed to do, not how it doesit. Black-box testing methods include: equivalence partitioning, boundary value analysis, all- pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing, and specification-based testing.

Specification-based testing aims to test the functionality of software according to the applicable requirements This level of testing usually requires thorough test cases to be providedto the tester, who then can simply verify that for a given input, the output value (or behavior),either "is" or "is not" the same as the expected value specified in the test case. Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designsto derive test cases. These tests can be functional or non-functional, though usually functional.

Specification-based testing may be necessary to assure correct functionality, but it is insufficient to guard against complex or high-risk situations.

One advantage of the black box technique is that no programming knowledge is required. Whatever biases the programmers may have had, the tester likely has a different set and may emphasize different areas of functionality. On the other hand, black-box testing has been said to be "like a walk in a dark labyrinth without a flashlight.” Because they do not examine the source code, there are situations when a tester writes many test cases to check something that could have been tested by only one test case or leaves some parts of the program untested.

This method of test can be applied to all levels of software testing: unit, integration, system and acceptance. It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well.

CHAPTER 6

CONCLUSION

6.1 Conclusion

In conclusion, the implementation of a Smart Parking System utilizing Internet of Things (IoT) technology brings numerous benefits and advancements to the parking industry. This transformative solution tackles the persistent challenges associated with urban parking by leveraging IoT devices, connectivity, and data analytics.

Firstly, the Smart Parking System optimizes the utilization of parking spaces through real-time monitoring and management. By utilizing sensors and cameras, the system collects data on parking availability and guides drivers to the nearest vacant spot, reducing the time spent searching for parking. This not only enhances convenience for drivers but also alleviates traffic congestion and reduces carbon emissions caused by unnecessary circling.

Secondly, the IoT-enabled system provides valuable insights and analytics. By analyzing the data collected, parking operators can identify patterns, peak hours, and popular parking areas, enabling them to make informed decisions and optimize parking facility operations. These insights can also be shared with city planners and policymakers to enhance urban planning, traffic management, and infrastructure development.

Furthermore, the Smart Parking System offers seamless integration with mobile applications and online platforms, enabling drivers to access real-time parking information, reserve spots in advance, and make cashless payments. This enhances user experience, improves convenience, and reduces the need for physical interactions.

The security aspect of the system is also noteworthy. With surveillance cameras and monitoring mechanisms, the Smart Parking System ensures the safety of vehicles and deters criminal activities, enhancing overall security in parking facilities.

In summary, the Smart Parking System using IoT technology revolutionizes the parking experience by optimizing space utilization, providing valuable insights, enhancing user convenience, and improving security. Its implementation promises to address the growing challenges of urban parking, leading to reduced congestion, improved environmental sustainability, and increased efficiency in urban transportation systems. With continued advancements and integration with smart city initiatives, the future of parking looks promising as we strive for smarter, more connected, and sustainable urban environments.

6.2 Scope for Future Improvement

The limitations associated with infrared sensors in smart parking systems present an opportunity for future project improvement. Here are some ways to address and improve these disadvantages:

1. Wide detection range:

Future improvements may focus on expanding the detection range of the infrared sensor. Research and development efforts may explore advanced sensor designs, signal amplification techniques, or integrate additional sensor technologies to improve coverage and improve accuracy of detection. vehicle detection, even at longer distances.

2. Environmental durability:

Progress can be made to improve the environmental durability of infrared sensors. This could involve developing sensor designs that are less sensitive to ambient light interference, such as using filters or shielding techniques to reduce the effects of direct sunlight or strong artificial light. In addition, a temperature compensation mechanism can be implemented to ensure consistent sensor performance in different weather conditions.

3. Detect out of sight:

Exceeding the vision requirement can be an important improvement goal. Future iterations may explore the use of advanced sensor configurations, such as multiple sensors per parking space, or the integration of other sensor technologies (e.g., ultrasonic or radar.) to enable more powerful and accurate detection even in the presence of obstacles or obstacles.

4. Improve vehicle differentiation:

Improvements can be made to better distinguish between different vehicle types and conditions of use. This may involve integrating additional sensors or technologies, such as cameras or machine vision systems, to capture more detailed information about the vehicle, including size, type or number of people in the car. That way, parking management can be more efficient and tailored to specific needs.

5. Automatic maintenance and calibration:

Future systems may focus on automating infrared sensor maintenance and calibration processes. This could involve developing self-diagnostic capabilities in sensors, where they can detect and report problems or deviations in their operation. Automatic calibration mechanisms can also be explored to reduce the need for manual intervention, making the system more efficient and reducing operating costs.

6. Noise Reduction:

Future improvements may be aimed at minimizing the interference potential of the infrared sensor. This may involve implementing advanced signal processing techniques, signal filtering mechanisms, or frequency modulation methods to reduce the effects of external infrared sources and improve resistance to interference. of the sensor.

CHAPTER 7

REFRENCES

1. Floris, Alessandro, et al. "A Social IoT-based platform for the deployment of a smart parking solution." Computer Networks 205 (2022): 108756.

2. Al-Turjman, Fadi, and Arman Malekloo. "Smart parking in IoT-enabled cities: A urvey." Sustainable Cities and Society 49 (2019): 101608.

3. Balhwan, Suman, et al. "Smart parking—a wireless sensor networks application using IoT." Proceedings of 2nd International Conference on Communication, Computing and Networking: ICCCN 2018, NITTTR Chandigarh, India. Springer Singapore, 2019.

4. Gopal, D. Ganesh, M. Asha Jerlin, and M. Abirami. "A smart parking system using IoT." World Review of Entrepreneurship, Management and Sustainable Development 15.3 (2019): 335-345.

5. Singh, Ashutosh Kumar, et al. "Smart parking system using IoT." International Research Journal of Engineering and Technology 6.4 (2019).

6. Ali, Ghulam, et al. "IoT based smart parking system using deep long short memory network." Electronics 9.10 (2020): 1696.

7. Canli, Hikmet, and Sinan Toklu. "Deep learning-based mobile application design for smart parking." IEEE Access 9 (2021): 61171-61183.

8. Agarwal, Yash, et al. "IoT based smart parking system." 2021 5th international conference on intelligent computing and control systems (ICICCS). IEEE, 2021.

9. Thakre, Mohan P., et al. "IOT based smart vehicle parking system using RFID." 2021 International Conference on Computer Communication and Informatics (ICCCI). IEEE, 2021.

10. Ashok, Denis, Akshat Tiwari, and Vipul Jirge. "Smart parking system using IoT technology." 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE). IEEE, 2020.

11. Aditya, Amara, et al. "An IoT assisted Intelligent Parking System (IPS) for Smart

Cities." Procedia Computer Science 218 (2023): 1045-1054.

12. Rodić, Lea Dujić, et al. "Privacy leakage of LoRaWAN smart parking occupancy sensors." Future Generation Computer Systems 138 (2023): 142-159.

13. Perković, T., et al. "Smart parking sensors: State of the art and performance evaluation." Journal of Cleaner Production 262 (2020): 121181.

14. Kanakaraja, P., et al. "An implementation of advanced IoT in the car parking system." Materials Today: Proceedings 37 (2021): 3143-3147.

15. Babu, M. V. S. S., et al. "Smart Parking System Using IOT." Dogo Rangsang Research Journal 14.6 (2021).