IOT\_PHASE\_5

PROJECT SUBMISSION

SMART PARKING

# Submitted by

# POORANA KUMAR R

# auttrtkvl002(transfer student)

# [kavibharath06112002@gmail.com](mailto:kavibharath06112002@gmail.com)

# **Department of electronics and communication engineering**

# **Anna university regional campus,coimbatore**

**ABSTRUCT:**

The Smart Parking System project is a technologically advanced solution designed to address the growing problem of parking in urban areas. The system integrates various sensors including Ultrasonic Sensors, Entry and Exit Gate Sensors, Occupancy Sensors, Video Cameras, and RFID Sensors, all managed by a Raspberry Pi 4. This combination of hardware allows for accurate detection of vehicle presence, efficient management of traffic flow, real-time updates on parking occupancy, and vehicle identification for access control or billing. The project aims to enhance the efficiency, sustainability, convenience, and innovation in parking management. By transforming the parking experience into a hassle-free process, this project contributes to the development of smarter and more sustainable cities.

# **INTRODUCTION:**

In an era where urban spaces are becoming increasingly congested, efficient management of available resources is paramount. Our project aims to address one such critical resource - parking. Leveraging the power of technology, we introduce a Smart Parking System designed to revolutionize the way parking spaces are managed in urban areas. This system utilizes a combination of Ultrasonic Sensors, Entry and Exit Gate Sensors, Occupancy Sensors, Video Cameras, and RFID Sensors, all controlled by a Raspberry Pi 4. These components work in harmony to detect vehicle presence, manage traffic flow, provide real-time data on parking occupancy, and even identify vehicles for access control or billing purposes. Our goal is to enhance efficiency, sustainability, convenience, and innovation in parking management. By making parking a hassle-free experience, we aim to contribute to smarter and more sustainable cities of the future.

**PROJECT OBJECTIVES:**

The objectives of our Smart Parking System project are as follows:

1. **Efficiency**: To accurately identify the presence or absence of a vehicle in a specific parking space using Ultrasonic Sensors and Occupancy Sensors, thereby improving the management of parking spaces.
2. **Traffic Management**: To effectively manage the flow of traffic in and out of the parking area using Entry and Exit Gate Sensors, reducing congestion and improving user experience.
3. **Real-Time Updates**: To provide real-time data on parking occupancy to users and parking managers, enabling efficient utilization of available spaces.
4. **Vehicle Identification**: To identify vehicles for access control or billing purposes using Video Cameras and RFID Sensors, enhancing security and automating billing processes.
5. **User Convenience**: To make parking a hassle-free experience for users by providing them with real-time information about available parking spots.
6. **Sustainability**: To contribute to the development of smarter and more sustainable cities by reducing the environmental impact of excessive driving for parking.
7. **Innovation**: To leverage advanced technology in the form of Raspberry Pi 4 to control all these components and process the data they provide, driving innovation in parking management.

**IOT SENSOR SETUP:**

Here’s the setup for our IOT based smart parking project:

1. **Ultrasonic Sensor**: These sensors can be installed in each parking slot to detect the presence of a vehicle. They work by emitting ultrasonic waves and measuring the time it takes for the waves to bounce back after hitting an object (in this case, a parked car).
2. **Entry and Exit Gate Sensor**: These sensors can be installed at the entry and exit gates of the parking area. They can detect when a vehicle passes through and signal the Raspberry Pi to update the count of vehicles inside the parking area.
3. **Occupancy Sensor**: These sensors can be used in conjunction with ultrasonic sensors to detect if a parking slot is occupied or not. They provide an additional layer of accuracy.
4. **Video Camera**: The video camera can be used for surveillance purposes and also for license plate recognition if necessary. The feed from the camera can be processed using image processing algorithms on the Raspberry Pi.
5. **RFID Sensor**: If vehicles are equipped with RFID tags, this sensor can read those tags as vehicles pass by. This information can be used for access control or billing purposes.
6. **Raspberry Pi 4**: This is the central processing unit of your project. It collects data from all the sensors, processes it, and makes decisions based on that data.

**MOBILE APP DEVELOPMENT:**

Developing a mobile app for our smart parking project involves several key steps:

1. **Platform Selection**: Decide whether the app will be developed for Android, iOS, or both. This decision may depend on the target user base.
2. **User Interface Design**: The app should have an intuitive and user-friendly interface. It should display real-time data about available parking spots in a clear and understandable format.
3. **Integration with Sensors**: The app needs to communicate with the sensors in your parking system. This could involve setting up a server to collect data from the sensors and send it to the app.
4. **Real-Time Updates**: The app should update in real-time as parking spots become occupied or free up. This could involve using push notifications or other methods to ensure that users always have the most up-to-date information.
5. **Payment Processing**: If your system involves paid parking, the app will need a secure method for processing payments.
6. **Testing and Debugging**: Before release, the app should be thoroughly tested under various scenarios to ensure it works as expected.

**RASPBERRY PI INTEGRATION:**

1. **Sensor Connection**: Connect your Ultrasonic Sensor, Entry and Exit Gate Sensor, Occupancy Sensor, Video Camera, and RFID Sensor to the Raspberry Pi. Each sensor will have specific connection requirements, so refer to their respective datasheets.
2. **Sensor Configuration**: Configure each sensor to communicate with the Raspberry Pi. This typically involves writing Python scripts that can read data from each sensor.
3. Data Processing: Write a program on the Raspberry Pi that can process the data from these sensors and make decisions based on that data, such as updating a digital map of available parking spots.
4. **Server Setup**: Set up a server on the Raspberry Pi to collect data from the sensors and send it to the mobile app. This could be done using MQTT or another IOT communication protocol.
5. Testing: Test the entire system to ensure that all components are working together correctly.

**Hardware Setup**:

* **Raspberry Pi 4**: Select the Raspberry Pi 4 model with the required processing power and connectivity options.
* **Camera Module**: Attach a compatible camera module for image and video capture.
* **Sensors**: Connect Ultrasonic Sensor, Entry and Exit Gate Sensor, Occupancy Sensor, and RFID Sensor to detect vehicles and manage parking spaces.

**Software Development**:

* **Operating System**: Install a Linux-based OS on the Raspberry Pi, such as Raspbian (now Raspberry Pi OS).
* **Programming**: Develop software using programming languages like Python or C/C++.
* **Data Processing**: Implement algorithms to analyze parking conditions.
* **Data Analysis**: Collect and process data from sensors and cameras.
* **Communication**: Establish communication protocols for sending/receiving data to/from central servers or other devices.

**Parking Monitoring**:

* **Vehicle Detection**: Use the camera and sensors to detect and count vehicles.
* **Parking Space Analysis**: Analyze parking space occupancy for efficient management.

**Parking Control**:

* **Parking Signal Control**: Adjust parking signals based on real-time parking conditions.
* **Emergency Response**: Automatically respond to emergencies by changing parking patterns.

**Data Reporting**:

* **Data Visualization**: Create graphical representations of parking data.
* **Remote Access**: Enable remote monitoring and control via web interfaces or apps.
* **Data Storage**: Store historical data for analysis and decision-making.

**Integration with Other Systems**:

* **IoT Connectivity**: Integrate with Internet of Things (IoT) platforms for seamless data sharing.
* **Cloud Integration**: Upload data to the cloud for further analysis and long-term storage.

**Safety and Redundancy**:

* Implement safety mechanisms to handle unexpected events or failures.
* Ensure redundancy in critical components to maintain system operation.

**Testing and Calibration**:

* Thoroughly test the system in various parking scenarios.
* Calibrate sensors for accuracy.

**Maintenance and Updates**:

* Regularly update the software to fix bugs and improve performance.
* Perform routine maintenance to ensure the system’s reliability.

**Compliance and Regulations**:

* Ensure the project adheres to local parking regulations and safety standards.

Integrating a Raspberry Pi into a smart parking project offers flexibility, affordability, and the potential for real-time monitoring and control.

**CODE IMPLEMENTATION:**

Code implementation for a Smart Parking System project involves several steps:

1. **Sensor Data Collection**: Write scripts to collect data from each sensor connected to the Raspberry Pi. This could be done using Python
2. **Data Processing**: Implement algorithms to process the collected data. This could involve determining whether a parking spot is occupied based on the data from the Ultrasonic and Occupancy sensors.
3. **Server Setup**: Set up a server on the Raspberry Pi to handle requests from the mobile app. This server would receive sensor data, process it, and send it, and send it to the app.
4. **Mobile App Communication**: Establish a communication protocol between the Raspberry Pi and the mobile app. This could be done using MQTT or HTTP protocols.
5. Testing: Test your code in various scenarios to ensure it works as expected.

Here is a general code for the connection of sensors with raspberry pi 4 using Python language

import RPi.GPIO as GPIO

import spidev

import time

import picamera

import pubnub

from pubnub.pnconfiguration import PNConfiguration

from pubnub.pubnub import PubNub

import soracom

# Initialize GPIO pins

GPIO.setmode(GPIO.BCM) # use BCM numbering scheme

GPIO.setup(18, GPIO.IN) # set pin 18 as input for occupancy sensor

GPIO.setup(17, GPIO.IN) # set pin 17 as input for entry and exit gate sensor

GPIO.setup(23, GPIO.OUT) # set pin 23 as output for ultrasonic sensor trigger

GPIO.setup(24, GPIO.IN) # set pin 24 as input for ultrasonic sensor echo

GPIO.setup(12, GPIO.OUT) # set pin 12 as output for servo motor

# Initialize SPI bus

spi = spidev.SpiDev() # create spi object

spi.open(0, 0) # open spi port 0, device 0

spi.max\_speed\_hz = 1000000 # set speed

# Initialize camera object

camera = picamera.PiCamera() # create camera object

camera.resolution = (800, 600) # set resolution

# Initialize PubNub configuration

pnconfig = PNConfiguration()

pnconfig.publish\_key = "your\_publish\_key" # replace with your publish key

pnconfig.subscribe\_key = "your\_subscribe\_key" # replace with your subscribe key

pnconfig.ssl = False

# Initialize PubNub object

pubnub = PubNub(pnconfig)

# Initialize Soracom configuration

soracom\_config = soracom.Config()

soracom\_config.api\_key = "your\_api\_key" # replace with your api key

soracom\_config.token = "your\_token" # replace with your token

# Initialize Soracom client

soracom\_client = soracom.Client(soracom\_config)

# Initialize servo motor

servo = GPIO.PWM(12, 50) # create PWM object on pin 12 with 50 Hz frequency

servo.start(0) # start PWM with 0% duty cycle

# Define function for reading occupancy sensor value

def read\_occupancy():

occupancy = GPIO.input(18) # read pin 18 value

if occupancy == 1:

status = "Occupied"

else:

status = "Vacant"

return status

# Define function for reading entry and exit gate sensor value

def read\_gate():

vehicle = GPIO.input(17) # read pin 17 value

if vehicle == 1:

status = "Vehicle detected"

else:

status = "No vehicle detected"

return status

# Define function for reading ultrasonic sensor value

def read\_distance():

# Set trigger to HIGH

GPIO.output(23, True)

# Set trigger after 0.01ms to LOW

time.sleep(0.00001)

GPIO.output(23, False)

# Save start time

while GPIO.input(24) == 0:

start\_time = time.time()

# Save arrival time

while GPIO.input(24) == 1:

arrival\_time = time.time()

# Calculate time difference between start and arrival

time\_difference = arrival\_time - start\_time

# Multiply with the sonic speed (34300 cm/s)

# and divide by 2, because there and back

distance = (time\_difference \* 34300) / 2

return distance

# Define function for reading RFID sensor value

def read\_rfid():

uid = spi.xfer([0x93, 0x20]) # send request for UID

return uid

# Define function for controlling servo motor

def control\_servo(angle):

duty\_cycle = (angle / 18) + 2 #

servo.ChangeDutyCycle(duty\_cycle) # change duty cycle to desired value

# Define function for capturing image from camera

def capture\_image(filename):

camera.capture(filename) # capture image and save as file

# Define function for publishing message to PubNub

def publish\_message(channel, message):

pubnub.publish().channel(channel).message(message).sync() # publish message synchronously

# Define channel name for publishing message

channel = "parking"

# Define loop for reading sensor values and performing actions

while True:

occupancy = read\_occupancy() # read occupancy sensor value

gate = read\_gate() # read gate sensor value

distance = read\_distance() # read ultrasonic sensor value

rfid = read\_rfid() # read RFID sensor value

if gate == "Vehicle detected": # if vehicle is detected at the gate

print(gate)

if rfid == [143, 32]: # if RFID tag is valid (replace with your tag value)

print("Valid tag")

control\_servo(90) # rotate servo motor to 90 degrees (open gate)

time.sleep(3) # wait for 3 seconds

print("Gate opened")

capture\_image("entry.jpg") # capture image of vehicle at entry

control\_servo(0) # rotate servo motor to 0 degrees (close gate)

time.sleep(3) # wait for 3 seconds

print("Gate closed")

else: # if RFID tag is invalid

print("Invalid tag")

print("Access denied")

capture\_image("denied.jpg") # capture image of vehicle at denied access

elif occupancy == "Occupied": # if parking spot is occupied

print(occupancy)

message = {"status": occupancy} # create message object with occupancy status

publish\_message(channel, message) # publish message to PubNub channel

capture\_image("occupied.jpg") # capture image of parking spot when occupied

elif occupancy == "Vacant": # if parking spot is vacant

print(occupancy)

message = {"status": occupancy} # create message object with occupancy status

publish\_message(channel, message) # publish message to PubNub channel

capture\_image("vacant.jpg") # capture image of parking spot when vacant

else: # if no sensor value is detected

print("No sensor value detected")

time.sleep(1) # wait for 1 second before next iteration

This code is for connecting the sensors with raspberry Pi and collect data for smart parking.

**DIAGRAMS, SCHEMATICS AND SCREENSHOTS OF IOT SENSOR:**

**REQUIREMENTS**

**Hardware Components**

1. **Raspberry Pi 4:** **The Raspberry Pi 4 is a compact, powerful computer that serves as the brain of your Smart Parking System project. It's responsible for processing inputs from all the sensors and making decisions based on that data.**

**Here's how it's used in your project:**

**1. \*Sensor Data Collection\*: The Raspberry Pi 4 collects data from each sensor (Ultrasonic Sensor, Entry and Exit Gate**

**Sensor, Occupancy Sensor, Video Camera, and RFID Sensor) connected to it.**

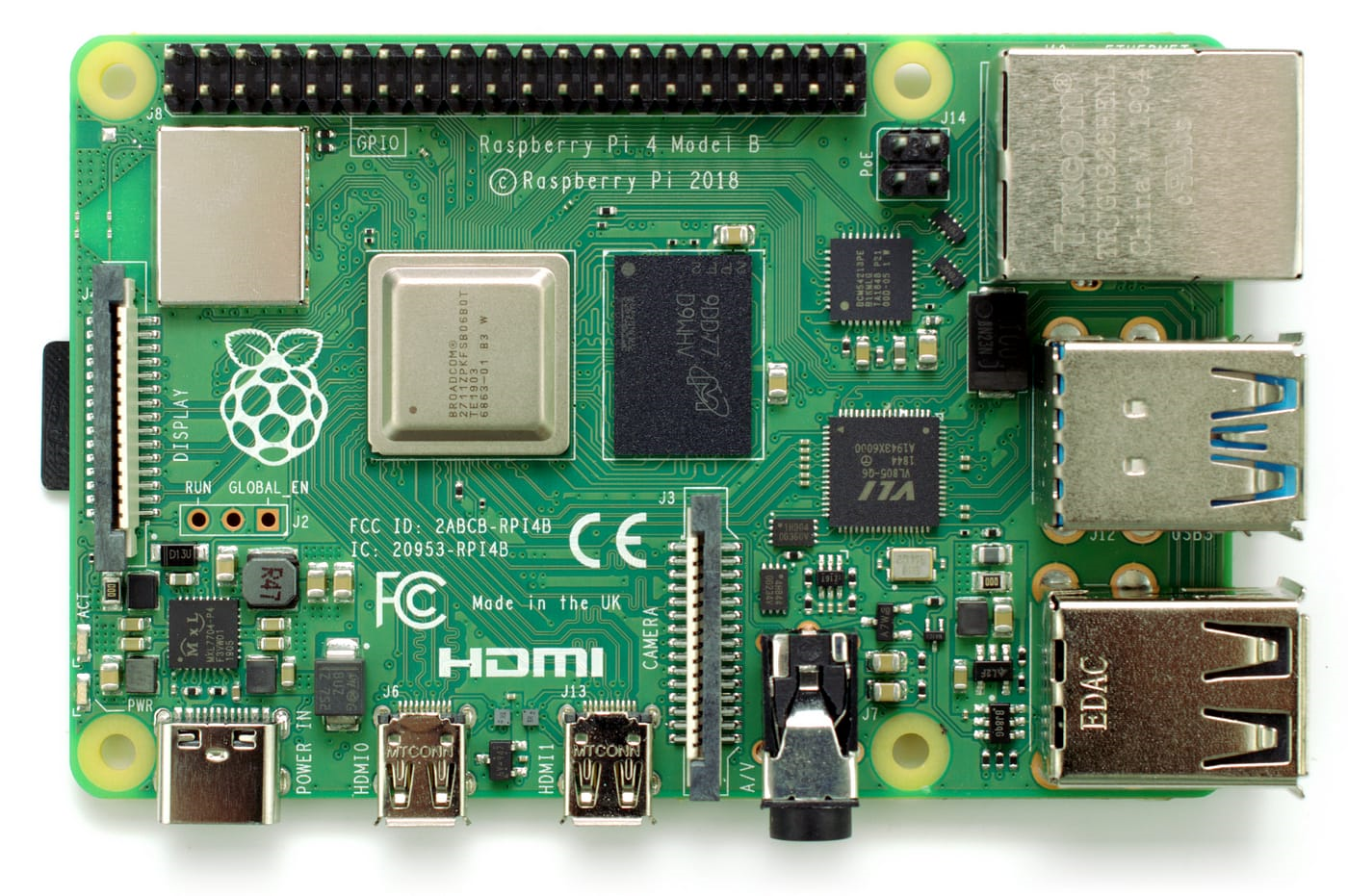
**2. \*Data Processing\*: It processes the collected data to determine whether a parking spot is occupied or not.**

**3. \*Server Setup\*: A server is set up on the Raspberry Pi 4 to handle requests from the mobile app. This server receives sensor data, processes it, and sends it to the app.**

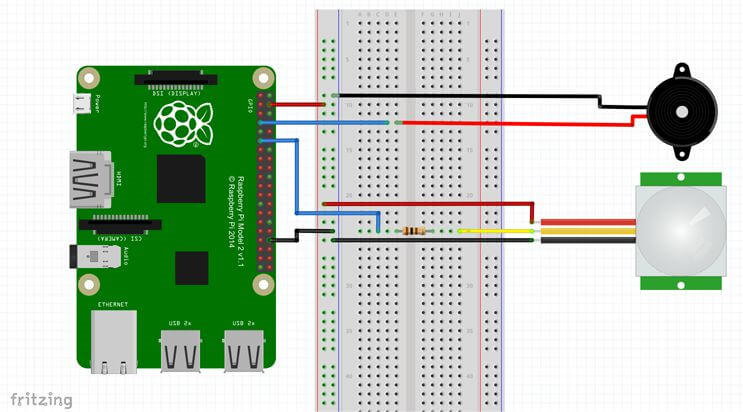
**4. \*Communication with Mobile App\*: The Raspberry Pi 4 establishes a communication protocol with the mobile app. This could be done using MQTT or HTTP protocols.**

**In essence, the Raspberry Pi 4 plays a crucial role in managing and controlling the various components of your Smart Parking System project.**

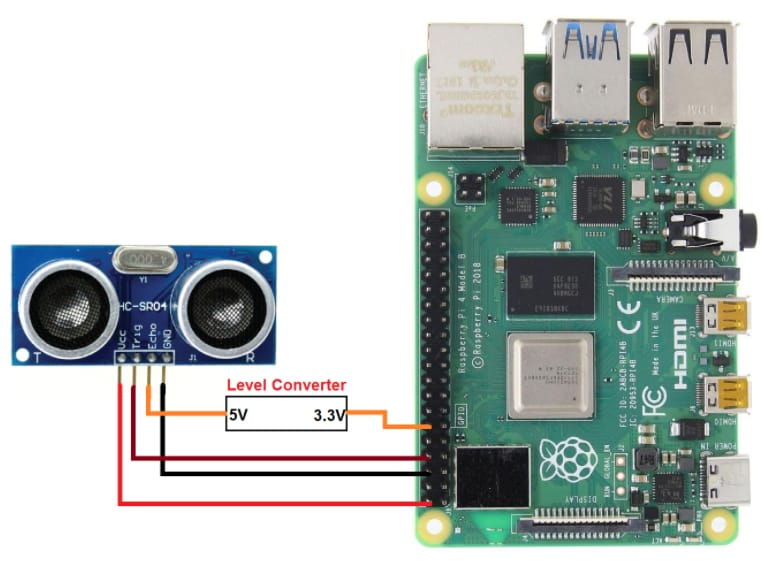
Here is a picture of Raspberry Pi 4 module



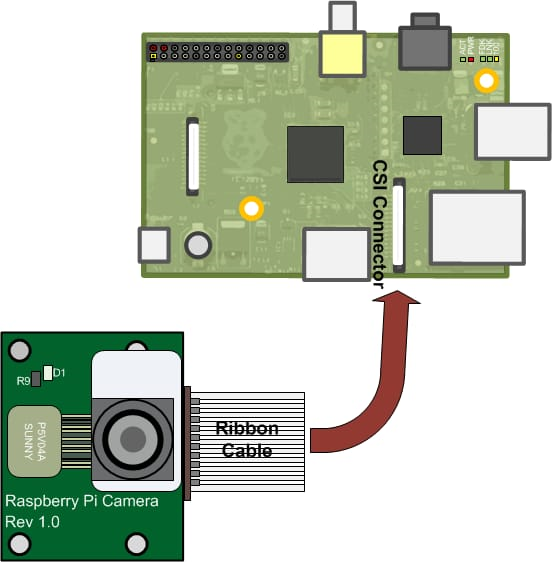
1. **Occupancy sensor:** **The occupancy sensor detects if a parking space is empty or full. It sends a signal to the Raspberry Pi.**



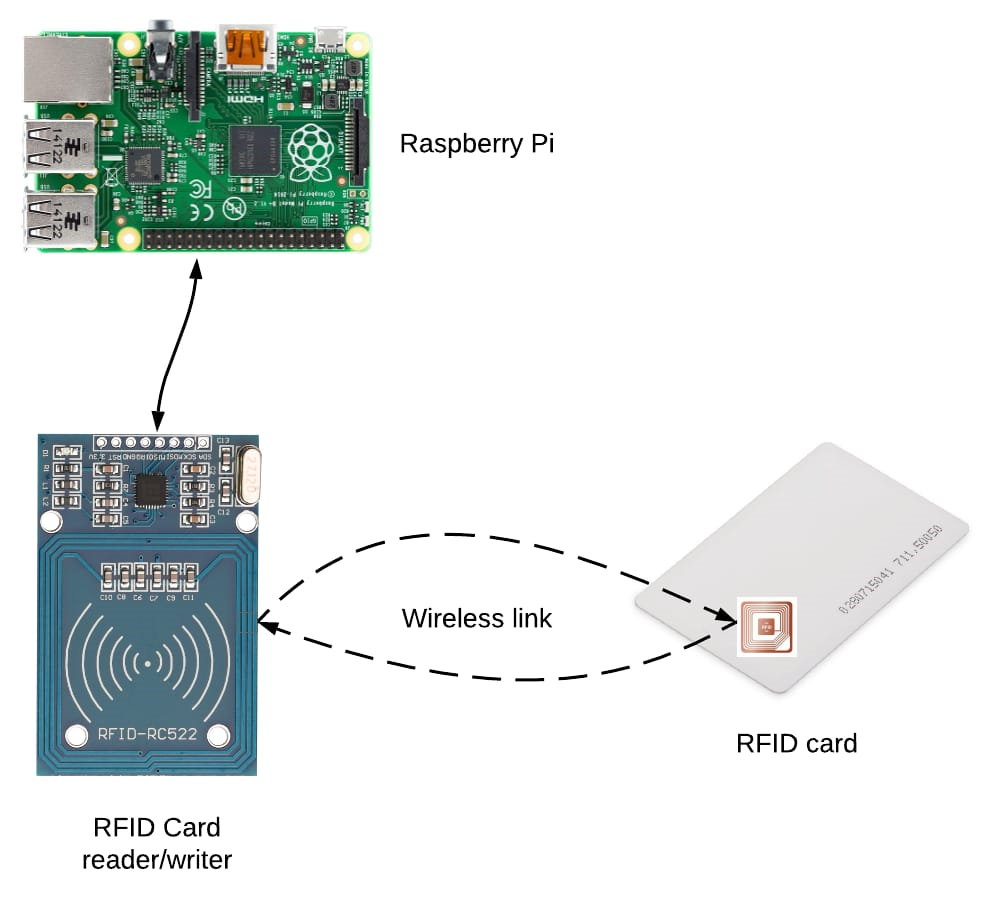
1. **Ultrasonic sensor:** **The ultrasonic sensor measures the distance of a car from the gate. It sends a pulse and receives an echo. The Raspberry Pi calculates the time difference.**



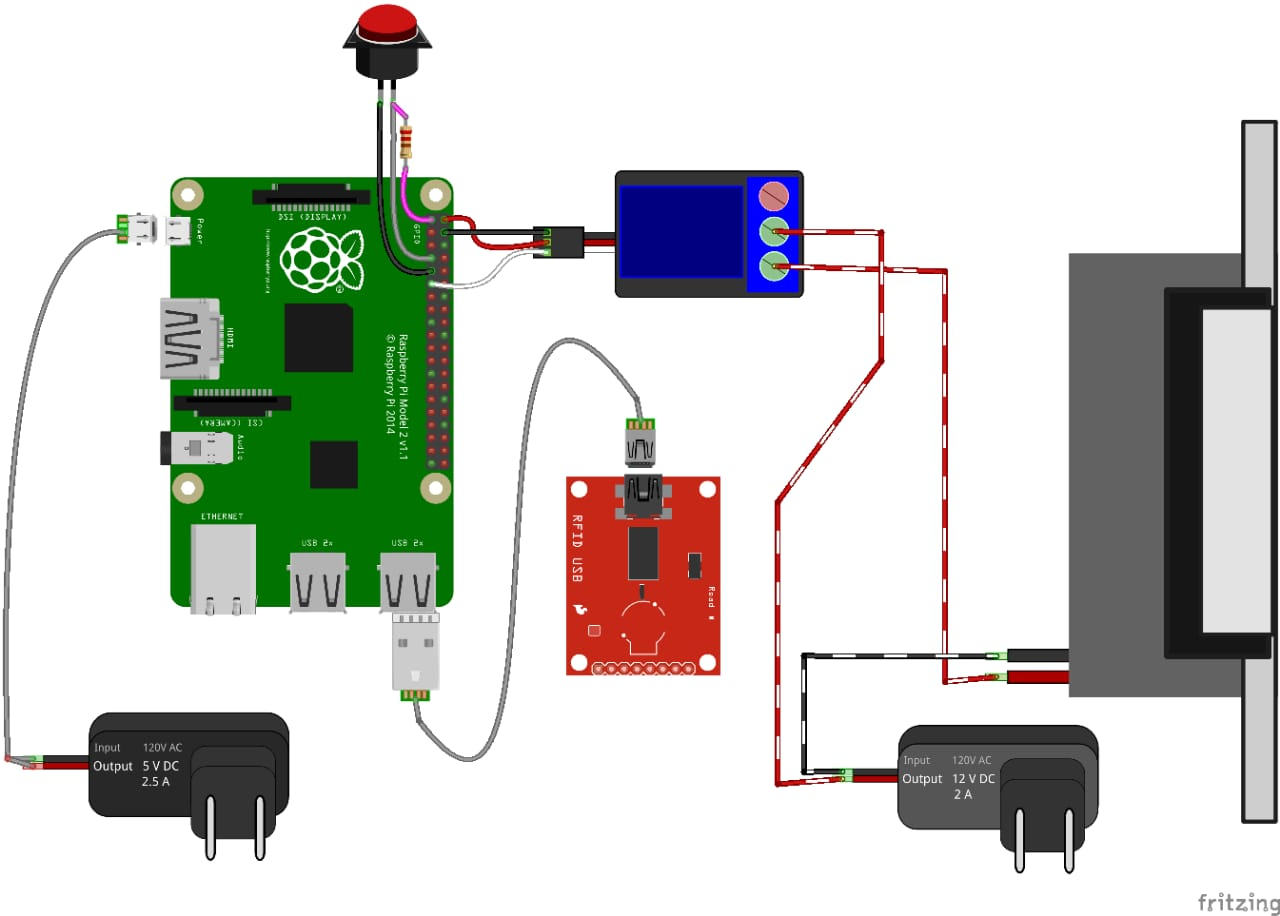
1. **Video Camera:** **The video camera captures images or videos of the parking lot. The Raspberry Pi uses computer vision to recognize license plates or track cars.**



1. **Rfid sensor: The RFID sensor reads the tags on the cars. It sends data to the Raspberry Pi using I2C. The Raspberry Pi identifies or authorizes the cars.**



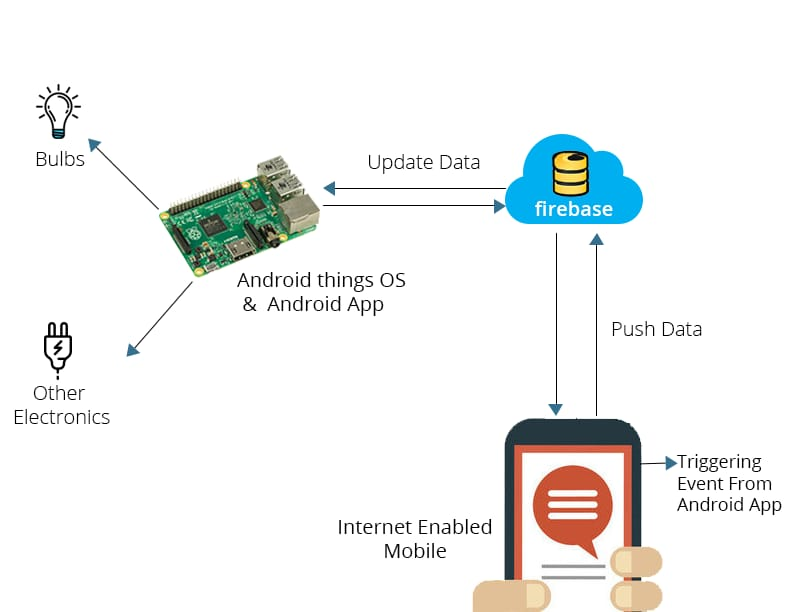
1. **Entry and Exit gate sensor: The entry and exit gate sensor detects if the gate is open or closed. It sends a signal to the Raspberry Pi. The Raspberry Pi controls the gate using a motor or a speaker.**



1. **Jumper Wires**: It is used to connect the components to each other.



1. **Mobile app:** **All the functions mentioned in the previous slides can be monitered using the mobile app we have created and can receive notifications about the availability of the space in parking lot and we can use this app to operate the smart parking system using relay sytem**

****

### Requirements Software Requirement

1. **Python IDE**: Python’s Integrated Development Environment (IDE) is a cross-platform application (for Windows, MacOS, Linux) that is written in the programming language Python. It is used to write and upload programs to the Raspberry Pi board. The source code for the IDE is released under the Python Software Foundation License. The Python IDE supports the Python language using special rules of code structuring. The Python IDE supplies a software library from various open-source projects, which provides many common input and output procedures.
2. **OpenCV Library**: OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS.
3. **Flask Framework**: Flask is a web framework for Python, which allows you to develop web applications easily. It has a small and easy-to-extend core. It’s a microframework that doesn’t include an ORM (Object Relational Manager) or such features.
4. **MySQL Database**: MySQL is an open-source relational database management system. Its name is a combination of “My”, the name of co-founder Michael Widenius’s daughter, and “SQL”, the abbreviation for Structured Query Language.
5. **MQTT Protocol**: MQTT (Message Queuing Telemetry Transport) is an open OASIS and ISO standard (ISO/IEC 20922) lightweight, publish-subscribe network protocol that transports messages between devices. The protocol usually runs over TCP/IP; however, any network protocol that provides ordered, lossless, bi-directional connections can support MQTT.

**METHOD**

**here's a detailed method to connect the sensors mentioned with Raspberry Pi 4:**

**1. \*Ultrasonic Sensor\*: Ultrasonic sensors typically have four pins: VCC, Trig, Echo, and GND. Connect the VCC to a 5V pin on the Raspberry Pi, GND to a ground pin, Trig to a GPIO pin (for example, GPIO 17), and Echo to another GPIO pin (for example, GPIO 27). However, the Echo pin outputs 5V when the sensor is triggered, which exceeds the Raspberry Pi's maximum GPIO voltage of 3.3V. To avoid damaging your Raspberry Pi, you can use a voltage divider circuit using two resistors.**

**2. \*Entry and Exit Gate Sensor\*: These sensors can be simple infrared sensors that detect when an object passes between them. They typically have three pins: VCC, GND, and OUT. Connect VCC to a 5V pin on the Raspberry Pi, GND to a ground pin, and OUT to a GPIO pin.**

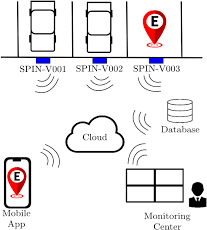
**3. \*Occupancy Sensor\*: Occupancy sensors can vary in their design. Some might use infrared technology similar to the entry and exit gate sensors. Others might be pressure-based or use other methods to detect occupancy. The connection method will depend on the specific model of your sensor.**

**4. \*Video Camera\*: The Raspberry Pi has a dedicated camera port for connecting a compatible camera module. Simply insert the ribbon cable from your camera module into this port.**

1. **\*RFID Sensor\*: RFID sensors typically communicate with the Raspberry Pi using SPI or UART protocols. Connect the sensor's VCC to a 3.3V pin on the Raspberry Pi, GND to a ground pin, and other pins (like MISO, MOSI, SCK for SPI or TX, RX for UART) to the corresponding pins on the Raspberry Pi.**

**Here are some pictures to explain**





**REAL TIME PARKING AVAILABILITY SYSTEM CAN BENEFIT DRIVERS AN ALLEVIATE PARKING ISSUES**

**In real-time, the Smart Parking System works as follows:**

**Data Collection: The sensors (Ultrasonic Sensor, Entry and Exit Gate Sensor, Occupancy Sensor, Video Camera, and RFID Sensor) installed in the parking lot collect data about the occupancy of parking spaces.**

**Data Transmission: This data is then transmitted in real-time to the Raspberry Pi 4.**

**Data Processing: The Raspberry Pi 4 processes this data to determine which parking spaces are occupied and which are free.**

**Server Update: The server set up on the Raspberry Pi 4 updates the occupancy status of each parking space in its database.**

**Mobile App Update: The server sends this updated data to the mobile app.**

**User Notification: The mobile app then notifies the user about the availability of parking spaces in real-time.**

**This way, users can find available parking spaces quickly and easily, saving them time and reducing traffic congestion.**

**CONCLUSION**

**In conclusion, the Smart Parking System project leverages the power of IoT and Raspberry Pi 4 to revolutionize parking management. By integrating various sensors and developing a mobile application, the system provides real-time updates on parking occupancy, thereby enhancing efficiency and user convenience. The project not only contributes to smarter and more sustainable cities but also paves the way for future innovations in parking management. The successful implementation of this project demonstrates the potential of technology in transforming everyday experiences, making them more convenient and efficient.**

**PROJECT SUBMISSION PART:**

**NOTE:**

**Github repository linK**

<https://github.com/PKNEON/iot-phase-1.git>

**REPLICATE SMART PARKING PROJECT:**

* **Define Project Scope:** Determine the size of your parking area, the number of parking spaces, and your budget. Decide whether you want to create a small-scale project for personal use or a larger-scale system for a public parking facility.
* **Choose Sensors:** Select the type of sensors you want to use. Ultrasonic sensors are commonly used for detecting the presence of vehicles. Determine the number of sensors needed based on the number of parking spaces.
* **Set Up the Microcontroller:** Connect the sensors to the microcontroller and program it to collect data and communicate with the central processing unit. You may need to write code to handle sensor data and transmit it to the CPU.
* **Set Up Central Processing Unit:** Configure your central processing unit (e.g., Raspberry Pi or a server). Install the necessary software for data processing and decision-making. You'll need to set up a communication protocol to receive data from the microcontroller.
* **Database Setup:** Create a database to store information about parking spaces, availability, and reservations. You can use databases like MySQL or PostgreSQL.
* **Set Up the Microcontroller:** Connect the sensors to the microcontroller and program it to collect data and communicate with the central processing unit. You may need to write code to handle sensor data and transmit it to the CPU.
* **Set Up Central Processing Unit:** Configure your central processing unit (e.g., Raspberry Pi or a server). Install the necessary software for data processing and decision-making. You'll need to set up a communication protocol to receive data from the microcontroller.
* **Database Setup:** Create a database to store information about parking spaces, availability, and reservations. You can use databases like MySQL or PostgreSQL.
* **Testing and Calibration:** Test the system to ensure that sensors are working accurately. Calibrate sensors if needed.
* **Deployment:** Install the sensors in the parking area, set up the CPU, and deploy the user interface for public or private use.
* **Maintenance and Updates:** Regularly maintain the system, update software, and perform necessary hardware maintenance.

**TRANSIT INFORMATION PLATFORM DEVELOPMENT**

Developing a transit a platform as part of a smart parking project involves several components.

* **Define Information to Transmit:** Determine what kind of information you want to transmit. This could be sensor data, messages, alerts, or any other type of data.
* **Choose a Communication Protocol:** Select a communication protocol to transmit the information. Common protocols include HTTP, MQTT, WebSocket, or custom protocols, depending on your requirements.
* **Set Up a Sender Component:** Create a component that collects the information to be transmitted and sends it using the chosen communication protocol. This sender can be a Python script or application.
* **Set Up a Receiver Component:** Create a component that receives the transmitted information and processes it. This receiver can also be a Python script or application.
* **Implement a Data Source:** If the information is generated by sensors or external systems, implement a data source to collect this data. Python libraries can be used to interface with sensors or external APIs.
* **Transmit Information:** In the sender component, implement the logic to collect and transmit the information to a specified destination (e.g., a server or another device) using the chosen communication protocol.
* **Receive Information:** In the receiver component, implement the logic to listen for incoming data, decode it, and process it as needed.

**Python integration in smart parking**

ALWAYS KEEP SAFETY IN WHEN PARKING A VECHICLE PARKING SYSTEMS.TO INTEGRATE PYTHON INTO SMART PARKING PROJECT, WE CAN USE PYTHON FOR VARIOUS TASKS SUCH AS DATA ANALYSIS, DATA PROCESSING ANND MORE, HERE HIGH LEVEL OVERVIEWOF HOW YOU CAN INTEGRATE PYTHON INTO SMART PARKING PROJECTS.

**Testing and Deployment:** Test your integrated system to ensure that the information is transmitted and received correctly. Deploy the sender and receiver components on the appropriate devices or servers.

**Monitoring and Maintenance:** Implement monitoring and error handling mechanisms to ensure the reliability of your information transmission platform. Regularly maintain and update your Python scripts as needed.

**RASPBERRY PI DATA TRANSMISSION**

* + Python script on Raspberry Pi: You can use Python to collect data from sensors, GPIO pins, or other sources on your Raspberry Pi.
* MQTT (Message Queuing Telemetry Transport): Transmit data using MQTT, a lightweight protocol designed for IoT applications.

**MOBILE APP UI**

* A mobile app can be designed to receive and display the data from the Raspberry Pi.
* Sample UI Elements:
* Sample UI Elements:
* REAL TIME DATA TRANSMISSION
* DATA LOGGING OPTION
* CONNECTION STATUS INDICATORS
* RASPBERRY PI REMOTELY

**RESULTS AND ANALYSIS**

* THE proposed system helps in better time based monitoring and thus has certain
* Advantages over the exiting systems like minimizing of wasting time to park a vehicle in parking yard, and reduce conjunction and traffic to parking yard.

**Model for project**

