DATE:31.10.2023

Project id:proj\_223339\_team\_12

Project title: Smart parking

IOT:

Documentation:

Objectives:

The goal of parking management is to ensure that parking spaces are available and accessible to those

Who need them while also preventing overcrowding and promoting safety. Parking management can

Encompass a wide range of activities, including: Design and construction of parking facilities.

# I )IOT device setup

Select a Communication Protocol :Decide on a communication protocol (e.g., Wi-Fi, Bluetooth, NB-IOT)

Based on your specific requirements, such as range, power consumption, and data transmission speed.

Power Supply :Ensure the device has a reliable power source. Depending on the device type, this could

Be a battery, solar panel, or wired power supply.

Device Placement :Install the device in a strategic location within the parking space. For sensors, this is

Typically on the ground. Cameras or other visual devices may require mounting at an elevated position

For a clear view.

Connectivity Setup: Connect the device to the chosen communication network. This might involve

Configuring Wi-Fi credentials, setting up a LORAWAN gateway, or establishing a cellular connection.

Configure Device Parameters: Depending on the specific device, configure parameters such as sensitivity,

Reporting intervals, and threshold values. This ensures accurate data collection.

Integration with Centralized System: Integrate the IOT device with the centralized smart parking system.

This involves registering the device on the platform and ensuring it can send and receive data.

Remember, specific setup steps may vary depending on the type and brand of the IOT device you’re

Using. Always refer to the manufacturer’s guidelines and documentation for detailed instructions.

# ii) Platform Development:

Developing a smart parking platform involves integrating various technologies to optimize parking

Spaces, enhance user experience, and improve overall efficiency. Here are key components and

Considerations for smart parking platform development:

Sensors and IOT Devices: Utilize sensors and IOT devices to detect occupancy in parking spaces. These

Devices can include cameras, ultrasonic sensors, and magnetic sensors, which provide real-time data

About available parking spots.

Data Collection and Analysis: Gather data from sensors and other sources to analyze parking patterns,

Peak usage times, and user behavior. Machine learning algorithms can be employed to predict parking

Availability and optimize resource allocation.

Mobile Apps and User Interfaces: Develop user-friendly mobile applications that allow users to check

Parking availability, reserve spots, and make payments. Implement intuitive interfaces and smooth

Navigation to enhance user experience.

Data Privacy and Compliance: Adhere to data privacy regulations and ensure that user data is securely

Stored and processed. Implement encryption and other security measures to protect sensitive

Information.

Feedback Mechanism: Implement a feedback mechanism within the app for users to report issues,

Provide suggestions, or rate their parking experience. Use this feedback to continually improve the

System.

By addressing these aspects in your smart parking platform development, you can create a seamless and

Efficient parking experience for users while optimizing resource utilization and contributing to smart city

Initiative

# iii)Code implementation:

Creating a complete smart parking system involves various technologies, including sensors,

Microcontrollers, databases, and sometimes mobile apps or web interfaces for user interaction. Below is

An example of a simple Python-based implementation for a smart parking system using Raspberry Pi,

Ultrasonic sensors, and a database (SQLite in this case). This script assumes you have installed the

Necessary libraries and set up your Raspberry Pi with the sensors.

Python Copy code

Import RPI. GPIO as GPIO

Import time

import sqlite3

# Initialize GPIO and database

GPIO. Set mode(GPIO.BOARD)

conn = sqlite3.connect('parking. Db ‘)

cursor = conn. .cursor()

# Create table if not exists

Cursor .execute('''CREATE TABLE IF NOT EXISTS parking\_ status (id INTEGER PRIMARY KEY

AUTOINCREMENT, spot\_ number INT, status INT)''') conn. commit()

# Set up ultrasonic sensor pins

TRIG = 11

ECHO = 13

GPIO. .setup(TRIG, GPIO.OUT)

GPIO. setup(ECHO, GPIO.IN)

Def get\_ distance():

# Function to measure distance from the ultrasonic sensor

GPIO. output(TRIG ,True)

Time .sleep(0.00001)

GPIO. output(TRIG ,False)

Pulse \_start=time. time()

Pulse \_end=time. .time()

while GPIO .input(ECHO) == 0:

pulse\_ start = time. .time()

while GPIO. input(ECHO) == 1:

pulse. \_end = time. time()

pulse\_ duration = pulse. \_end - pulse\_ start

distance = pulse\_ duration \* 17150

distance = round(distance, 2)

return distance

try:

while True:

distance = get\_ instance()

print("Distance: {} cm “.format(distance))

if distance < 30:

# Parking spot occupied

cursor. execute("INSERT INTO parking \_status (spot\_ number, status) VALUES (?, ?)", (1, 1))

conn. commit()

else:

# Parking spot vacant

Cursor .execute("INSERT INTO parking. \_status (spot\_ number, status) VALUES (?, ?)", (1, 0))

conn. commit()

time. sleep(5) # Check parking status every 5 seconds

except Keyboard interrupt:

# Clean up GPIO and close the database connection

GPIO .cleanup()

Conn .close()

In this script, the ultrasonic sensor measures the distance. If the distance is less than 30 cm, it

considers the parking spot occupied (status = 1) and stores this information in the database.

Otherwise, it marks the spot as vacant (status = 0). You can modify and expand this code to handle

multiple parking spots, integrate it with a web interface, or use IoT platforms for real-time monitoring.

Additionally, make sure to handle exceptions and errors for a more robust implementation.

return distance

try:

while True:

distance = get\_ instance()

print("Distance: {} cm “.format(distance))

if distance < 30:

# Parking spot occupied

cursor. execute("INSERT INTO parking \_status (spot\_ number, status) VALUES (?, ?)", (1, 1))

conn. commit()

else:

# Parking spot vacant

Cursor .execute("INSERT INTO parking. \_status (spot\_ number, status) VALUES (?, ?)", (1, 0))

conn. commit()

time. sleep(5) # Check parking status every 5 seconds

except Keyboard interrupt:

# Clean up GPIO and close the database connection

GPIO .cleanup()

Conn .close()

In this script, the ultrasonic sensor measures the distance. If the distance is less than 30 cm, it

considers the parking spot occupied (status = 1) and stores this information in the database.

Otherwise, it marks the spot as vacant (status = 0). You can modify and expand this code to handle

multiple parking spots, integrate it with a web interface, or use IoT platforms for real-time monitoring.

Additionally, make sure to handle exceptions and errors for a more robust implementation.

# iv.diagram:

# V) schematics

Certainly! Smart parking systems use various technologies like sensors, IoT, and

mobile apps to optimize parking space usage. Here's a basic schematic

representation:

Plaintext Copy code

Smart Parking System Schematic: +----------------------------------------+ | Parking Area

| | | | +-----------------------------+ | | | Parking Spaces | | | | | | | | +------+ +------+

+------+ | | | | | 01 | | 02 | | 03 | | | | | +------+ +------+ +------+ | | | | | | | +--------

---------------------+ | | | +----------------------------------------+ +---------------------------------

-------+ | Sensor Nodes | | +------+ +------+ +------+ | | | S1 | | S2 | | S3 | | | +------+

+------+ +------+ | +----------------------------------------+ +----------------------------------------

+ | Control Unit | | | | +--------------------------------+ | | | Data Processing | | | | and

Decision Making | | | +--------------------------------+ | | | | | | +----+ +----+ | | | | | |

+--------+ +--------+ | | | LED | | Mobile | | | | Display| | App | | | +--------+ +--------

+ | | | +----------------------------------------+

In this schematic:

Parking Area: Represents the physical parking spaces where vehicles are parked.

Sensor Nodes (S1, S2, S3, etc.): These are sensors placed in each parking space to

Detect the presence of a vehicle. They send data to the control unit.

Control Unit: Processes data from sensor nodes, makes decisions (like whether a

Space is occupied or vacant), and communicates with LED displays or a mobile app

To inform drivers about parking space availability.

This schematic illustrates a basic setup. Real-world implementations might include

Additional features like payment systems, security cameras, or integration with

Navigation apps for finding available parking spaces.

# Vi data sharing

# Vii.title details

Overhead sensors scan the vicinity and identify the

Position of a parked car or an empty parking space while

Also measuring the length of an available parking space

And registering whether vehicles have been in an illegal

Space (e.g. in front of a fire hydrant or other no-parking

Zone)

# Conclusion:

