# Smart Parking

## Sensor Specifications and Technology Development

### 1. Vehicle Detection Sensors:

**Type:** Ultrasonic Sensors

**Specifications:**

- Operating Range: 2cm - 400cm
- Detection Angle: 15 - 200 degrees
- Precision: ± 1 cm

**Purpose:**

To accurately detect the presence of vehicles and their distance from the sensor, aiding in parking spot allocation.

### 2. IR (Infrared) Sensors:

**Type:** Infrared Proximity Sensors

**Specifications:**

- Detection Range: Up to 20cm
- Operating Voltage: 3.3V - 5V

**Purpose:**

To detect nearby objects or obstacles in the parking area.

### 3. Camera Systems:

**Type:** High-Resolution Cameras (e.g., Raspberry Pi Camera)

**Specifications:**

- Resolution: 5MP or higher
- Field of View: Adjustable for capturing parking spot occupancy

**Purpose:**

To capture images for advanced vehicle recognition and monitoring parking space availability.

### 4. Environmental Sensors:

**Type:** Environmental Monitoring Sensors (e.g., Gas and Air Quality Sensors)

**Specifications:**

- Parameters: $CO_2$ levels, PM2.5, PM10, Noise levels

- Communication: MQTT for data transmission

## Purpose:

To monitor and provide real-time data on the environmental conditions in the parking area.

# The Technology Development

## Hardware Setup:

- **Raspberry Pi:** Utilize a Raspberry Pi board as the core processing unit to control sensors, capture video, and run the software.
- **Ultrasonic Sensor:** Connect ultrasonic sensors to measure the distance between the sensor and the vehicle.
- **Infrared Sensor:** Integrate infrared sensors to detect the presence of vehicles in parking spots.
- **Camera:** Attach a high-resolution camera (e.g., Raspberry Pi Camera) to capture video for vehicle detection.
- **Environmental Sensor (DHT11):** Connect an environmental sensor to measure temperature and humidity in the parking area.
- **Sensor Integration:**
- Write scripts to interface with the ultrasonic sensor, infrared sensor, camera, and environmental sensor using appropriate libraries and GPIO pins.
- Implement functions to read data from these sensors (e.g., distance, infrared presence, temperature, and humidity).

## Vehicle Detection:

- Utilize computer vision techniques (e.g., Haar cascades) to develop a vehicle detection model.
- Train the model using a dataset of vehicle images and integrate it into the system.

## Data Processing:

- Process sensor data (distance, infrared presence) and vehicle detection results to determine parking space occupancy and calculate parking vacancies.
- Analyze the data to identify incoming and outgoing vehicles.

## User Interface:

- Develop a user interface, which can be a web application or a mobile app, to display real-time parking status, incoming and outgoing vehicles, and parking vacancies.
- Provide a simple and intuitive interface for users to interact with the system, reserve parking spots, and get guidance.

## Integration and Communication:

- Establish communication protocols (e.g., MQTT) for seamless data transmission between the Raspberry Pi and the user interface.
- Ensure a secure and reliable connection between the IoT devices and the user interface.

## Testing and Optimization:

- Conduct thorough testing to validate the functionality, accuracy, and responsiveness of the system.
- Optimize the software for efficiency, speed, and low resource usage on the Raspberry Pi.

## Deployment:

- Deploy the system in the parking area, ensuring all sensors are properly calibrated and positioned.
- Monitor the system in a real-world environment, making necessary adjustments for optimal performance.

# Raspberry Pi Python Script
# Vehicle Detection using OpenCV:

```
import RPi.GPIO as GPIO

import time

import cv2

import board

import adafruit_dht
```

```
# Initialize and configure sensors
ultrasonic_trig_pin = 18
ultrasonic_echo_pin = 24
infrared_pin = 17

# Initialize ultrasonic sensor
GPIO.setmode(GPIO.BCM)
GPIO.setup(ultrasonic_trig_pin, GPIO.OUT)
GPIO.setup(ultrasonic_echo_pin, GPIO.IN)

# Initialize infrared sensor
GPIO.setup(infrared_pin, GPIO.IN)

# Initialize DHT11 environmental sensor
dht_sensor = adafruit_dht.DHT11(board.D4)

# Load the pre-trained vehicle detection model
vehicle_cascade = cv2.CascadeClassifier('vehicle_cascade.xml')

# Initialize counters for vehicles
prev_vehicle_count = 0
total_parking_spaces = 20  # Total parking spaces in the lot
```

```
def measure_distance():
    # Function to measure distance using ultrasonic sensor
    # ... (Same as before)

def read_infrared():
    # Function to read infrared sensor
    infrared_value = GPIO.input(infrared_pin)
    return infrared_value

def read_environmental_sensor():
    # Function to read data from the environmental sensor (DHT11)
    # ... (Same as before)

def detect_vehicles(frame):
    # Function to detect vehicles using OpenCV
    # ... (Same as before)

# Initialize the camera
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

try:
    while True:
```

```
# Read sensor data
distance = measure_distance()
infrared_value = read_infrared()
temperature, humidity = read_environmental_sensor()

# Print sensor data
print(f"Ultrasonic Sensor Distance: {distance} cm")
print(f"Infrared Sensor Value: {infrared_value}")

if temperature is not None and humidity is not None:
    print(f"Temperature: {temperature}°C, Humidity: {humidity}%")

# Capture video from the camera and detect vehicles
ret, frame = cap.read()
if ret:
    frame_with_vehicles = detect_vehicles(frame)

    # Count vehicles
    vehicles = len(vehicle_cascade.detectMultiScale(
        cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY), 1.1, 5))
    print(f"Detected vehicles: {vehicles}")

    # Calculate parking vacancies
    parking_vacancies = total_parking_spaces - vehicles
```

```
            print(f"Parking Vacancies: {parking_vacancies}")

            # Determine incoming and outgoing vehicles
            if vehicles > prev_vehicle_count:
                incoming_vehicles = vehicles - prev_vehicle_count
                print(f"Incoming Vehicles: {incoming_vehicles}")
            elif vehicles < prev_vehicle_count:
                outgoing_vehicles = prev_vehicle_count - vehicles
                print(f"Outgoing Vehicles: {outgoing_vehicles}")

            # Update previous vehicle count
            prev_vehicle_count = vehicles

            cv2.imshow('Vehicle Detection', frame_with_vehicles)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

finally:
    cap.release()
    cv2.destroyAllWindows()
    GPIO.cleanup()
```

The script continuously displays this information. Adjust the **total_parking_spaces** variable to match the actual total parking spaces in your parking lot.