**SIMATS SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**CHENNAI-602105**

**Capstone Project**

**SELF CAR PARKING SYSTEM USING AI**

**Course Code**:  CSA1741

**Course Name**:  Artificial Intelligence for Decision Making

**Submitted by**:

V.SRIKARI [192211509]

G.TEJASWI [192211748]

**Slot:** SLOT B

**Date of Submission:**29.07.2024

**Table of Contents**

|  |  |
| --- | --- |
| **S.NO** | **TOPICS** |
|  | **Abstract** |
| 1 | **Introduction** |
| 2 | **Project Description**  System Architecture |
| 3 | **Problem Description** |
| 4 | **Tool Description**  User interface  Features |
| 5 | **Operations** |
| 6 | **Approach / Module Description / Functionalities** |
| 7 | **Implementation**  Coding |
| 8 | **Output**  Output with Screenshots |
| 9 | **Conclusion** |

**ABSTRACT**

The development of a self car parking system using Artificial Intelligence (AI) represents a significant advancement in vehicle automation technology. This system employs a combination of sensors, including ultrasonic, camera, Lidar, and radar, to perceive the environment and identify suitable parking spaces. Through the integration of sensor fusion and computer vision, the system processes real-time data to detect obstacles and map the surroundings. Advanced machine learning algorithms, particularly deep learning for object recognition and reinforcement learning for path planning, enable the system to calculate optimal parking maneuvers. Actuators then execute these maneuvers by controlling the vehicle's steering, throttle, and brakes. The system ensures precise and safe parking even in dynamic and challenging environments. Key challenges include handling varying conditions and ensuring high accuracy and safety standards. The implementation of self-parking systems has broad applications, from enhancing consumer vehicle convenience to optimizing commercial fleet operations and contributing to smart city initiatives. Future developments will focus on improved AI algorithms, vehicle-to-everything (V2X) communication, and deeper integration with autonomous driving technologies, paving the way for more sophisticated and reliable self-parking solutions.

1. **INTRODUCTION**

The evolution of autonomous vehicle technology has seen significant strides in recent years, with self car parking systems emerging as a critical component of modern automotive innovation. These systems leverage Artificial Intelligence (AI) to automate the parking process, offering drivers a convenient and safe alternative to traditional parking methods. Self-parking systems aim to address common challenges faced by drivers, such as maneuvering in tight spaces, avoiding obstacles, and reducing the stress and time associated with parking.

The core of a self-parking system is its ability to perceive and interpret the vehicle's surroundings accurately. This is achieved through an array of sensors, including ultrasonic sensors, cameras, Lidar, and radar, which provide comprehensive data about the environment. By employing sensor fusion techniques, the system integrates data from these diverse sources to create a detailed and reliable representation of the vehicle's immediate vicinity.

Machine learning algorithms play a pivotal role in processing this sensory data. Deep learning models, particularly convolutional neural networks (CNNs), are utilized for object detection and recognition, enabling the system to identify parking spaces, obstacles, and lane markings. Reinforcement learning techniques help the system learn and optimize parking strategies by simulating various scenarios and outcomes.

Once a suitable parking space is identified and measured, the system engages in path planning, calculating the most efficient and safe trajectory for the vehicle. This involves considering the vehicle's dimensions, the size and orientation of the parking space, and any obstacles that may be present. The control system, comprising actuators that manage the steering, throttle, and brakes, executes the parking maneuver with precision.

Despite the advanced technology, several challenges remain in the development and deployment of self-parking systems. These include navigating dynamic environments with moving obstacles, ensuring high accuracy in various conditions, and maintaining stringent safety standards. Addressing these challenges is crucial for the widespread adoption and success of self-parking technologies.

The applications of self-parking systems are diverse and impactful. For individual consumers, they enhance driving convenience and safety. In commercial contexts, such as logistics and transportation, they improve operational efficiency. Furthermore, in the broader scope of smart cities, self-parking systems contribute to optimal parking space utilization and traffic management.

As the field progresses, future developments are expected to enhance the capabilities of self-parking systems further. Innovations such as vehicle-to-everything (V2X) communication, improved AI algorithms, and deeper integration with fully autonomous driving systems will pave the way for more sophisticated and reliable self-parking solutions.

1. **Project Description**

**Objective**

The objective of this project is to design and implement a self-parking system for vehicles using advanced Artificial Intelligence (AI) technologies. The system will autonomously detect suitable parking spaces, plan the optimal parking path, and execute the parking maneuver with minimal or no human intervention. The focus will be on ensuring accuracy, safety, and reliability in various parking scenarios.

**Scope**

**The project will cover the following key areas**

**1. Sensor Integration**

**- Implementing and integrating ultrasonic sensors, cameras, Lidar, and radar to perceive the vehicle's surroundings.**

**- Ensuring robust sensor fusion to create an accurate and comprehensive environmental model.**

**2. Data Processing**

**- Developing algorithms for processing sensor data, including computer vision techniques for object detection and recognition.**

**- Using machine learning models to interpret sensory input and identify parking spaces and obstacles.**

**3. Path Planning**

**- Designing and implementing path planning algorithms that calculate the most efficient and safe trajectory for parking.**

**- Incorporating reinforcement learning to optimize parking strategies through simulated scenarios.**

**4. Control Systems**

**- Developing control algorithms to manage the vehicle's steering, throttle, and brakes.**

**- Ensuring precise execution of parking maneuvers based on the calculated path.**

**5. System Integration and Testing**

**- Integrating all components into a cohesive system.**

**- Conducting extensive testing in various parking scenarios to validate system performance, accuracy, and safety.**

**Key Components**

**1. Sensors**

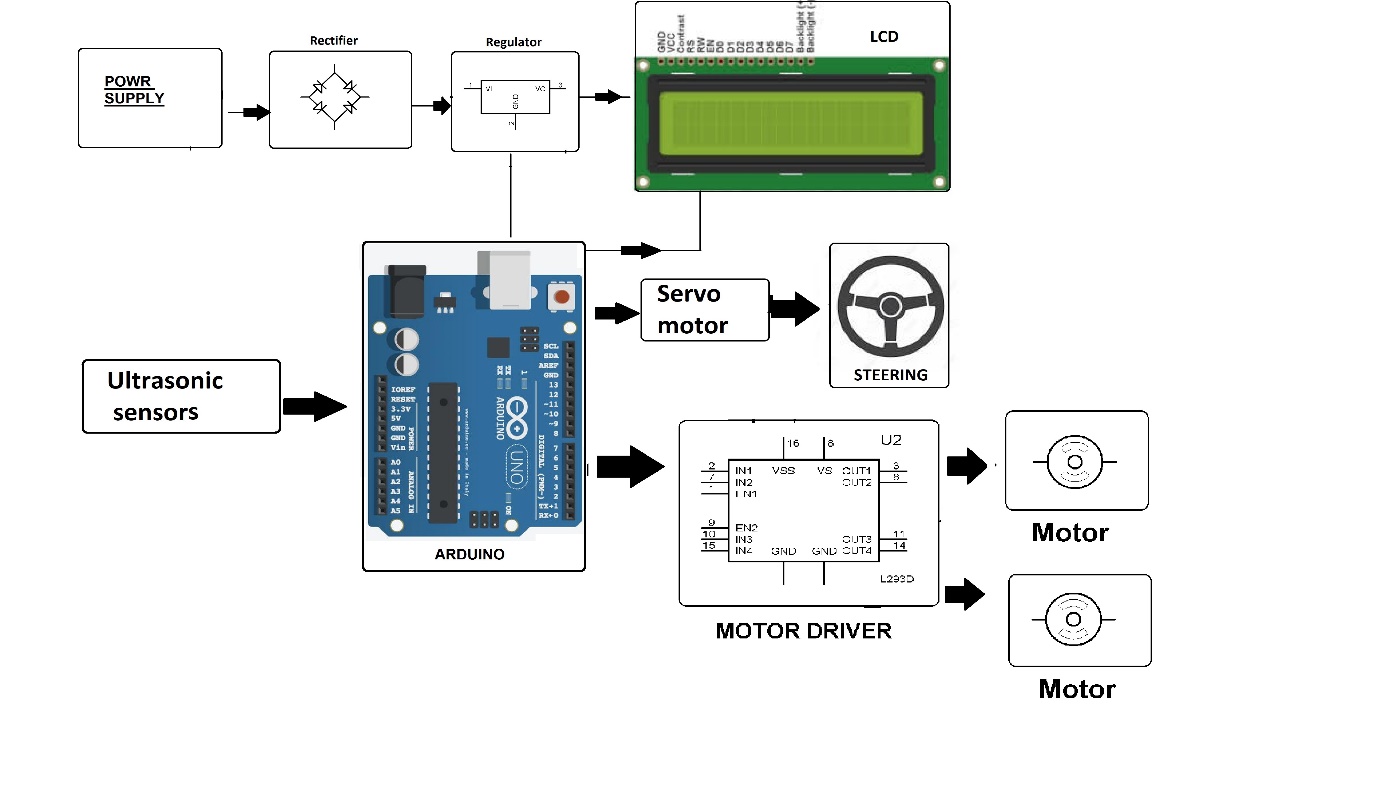
* **Ultrasonic Sensors:For short-range distance measurement and obstacle detection.**
* **Cameras:For visual input to identify parking spaces and obstacles.**
* **Lidar:For detailed environment mapping and distance measurement.**
* **Radar:For detecting objects and assessing the surroundings, especially in low visibility conditions.**

**2. AI Algorithms**

* **Computer Vision:Using convolutional neural networks (CNNs) for object detection and recognition.**
* **Reinforcement Learning:To optimize parking strategies through trial and error.**
* **Path Planning:Algorithms to compute the optimal parking path considering the vehicle's dimensions and environment.**

**3. Control Systems**

* **Actuators:Tocontrol the vehicle’s steering, throttle, and brakes.**
* **PID Controllers:To maintain smooth and precise control over the vehicle’s movements.**
* **Challenges**
  + **Handling dynamic environments with moving obstacles and varying lighting conditions.**
  + **Ensuring high accuracy and reliability in diverse parking scenarios.**
  + **Maintaining safety standards to prevent collisions and ensure safe operation.**
* **Applications**
  + **Consumer Vehicles:Enhancing convenience and safety for everyday drivers.**
  + **Commercial Fleets: Improving efficiency in logistics and transportation.**
  + **Smart Cities:Optimizing parking space usage and reducing urban congestion.**
* **Future Directions:V2X Communication: Enabling vehicles to communicate with each other and infrastructure to improve parking efficiency.**
* **Improved AI Algorithms:Enhancing machine learning models for better accuracy and reliability.**
* **Integration with Autonomous Driving:Incorporating self-parking systems as a component of fully autonomous vehicles.**
* **This project aims to demonstrate the feasibility and benefits of AI-driven self-parking systems, providing a foundation for future developments in autonomous vehicle technology.**



**System Architecture**

1. **Problem Description**

The increasing complexity of urban environments and the growing number of vehicles have made parking a significant challenge for drivers. Traditional parking methods often involve manual maneuvering, which can be stressful, time-consuming, and prone to errors. Several key issues are associated with conventional parking that a self-parking system using Artificial Intelligence (AI) aims to address:

1. **Difficulty in Maneuvering:**
   * **Tight Spaces:** Parking in narrow or confined spaces can be challenging, particularly for less experienced drivers.
   * **Complexity:** Maneuvering in tight spaces often requires precise control over the vehicle’s steering, throttle, and brakes, which can be difficult to execute accurately.
2. **Obstacle Avoidance:**
   * **Dynamic Obstacles:** Urban environments frequently present moving obstacles, such as pedestrians and other vehicles, which can complicate parking maneuvers.
   * **Fixed Obstacles:** Permanent structures like curbs, poles, and walls require careful navigation to avoid collisions.
3. **Time Consumption:**
   * **Searching for Space:** Finding an available parking spot, especially in crowded areas, can be time-consuming and stressful.
   * **Parking Duration:** Maneuvering into a parking spot can take longer than necessary, further contributing to overall parking time.
4. **Accuracy and Precision:**
   * **Risk of Collisions:** Manual parking increases the risk of minor accidents, such as scraping against other vehicles or objects.
   * **Alignment Issues:** Proper alignment within the parking space is crucial for efficient use of space and avoiding penalties or additional costs.
5. **Safety Concerns:**
   * **Human Error:** Drivers may misjudge distances or fail to notice obstacles, leading to potential accidents or damage.
   * **Inadequate Detection:** Conventional parking methods often rely on the driver’s visual and spatial judgment, which can be flawed or insufficient.
6. **Inconsistent Performance:**
   * **Environmental Conditions:** Parking performance can be affected by weather conditions, lighting, and varying road surfaces, making it challenging to achieve consistent results.

**Objectives of the Self-Parking System:**

* **Automated Space Detection:** Accurately identify and assess available parking spaces using a combination of sensors and computer vision.
* **Optimized Path Planning:** Compute and execute the most efficient and safe parking trajectory based on real-time data and environmental conditions.
* **Enhanced Safety:** Reduce the risk of collisions and accidents through precise control and obstacle detection.
* **Improved Efficiency:** Minimize the time spent searching for and maneuvering into parking spaces, enhancing overall convenience for drivers.

By addressing these problems, the self-parking system aims to provide a reliable, efficient, and user-friendly solution that improves the parking experience and enhances vehicle safety in various driving conditions.

1. **TOOL DESCRPTION**

To develop and implement a self-parking system using Artificial Intelligence (AI), several tools and technologies are required. These tools can be categorized into hardware components, software frameworks, and development platforms. Each tool plays a crucial role in ensuring the system’s functionality, accuracy, and reliability.

#### 1. ****Hardware Components****

* **Ultrasonic Sensors**
  + **Purpose:** Measure distances to nearby objects using sound waves.
  + **Functionality:** Provide short-range distance measurements, essential for detecting obstacles and proximity to other vehicles or structures.
  + **Example:** Sensors placed at the front, rear, and sides of the vehicle.
* **Cameras**
  + **Purpose:** Capture visual information about the environment.
  + **Functionality:** Used for object detection, lane recognition, and parking space identification through computer vision techniques.
  + **Example:** Front-facing, rear-facing, and side-view cameras.
* **Lidar (Light Detection and Ranging)**
  + **Purpose:** Create detailed 3D maps of the vehicle’s surroundings.
  + **Functionality:** Provides precise distance measurements and environment mapping, crucial for accurate obstacle detection and spatial understanding.
  + **Example:** 360-degree Lidar sensors mounted on the vehicle.
* **Radar**
  + **Purpose:** Detect objects and assess the environment, especially in low visibility conditions.
  + **Functionality:** Uses radio waves to detect and track objects, helping to complement data from other sensors.
  + **Example:** Front and rear radar sensors.

#### 2. ****Software Frameworks****

* **Computer Vision Libraries**
  + **Purpose:** Process and analyze visual data from cameras.
  + **Functionality:** Libraries like OpenCV and TensorFlow facilitate image processing tasks, including object detection, recognition, and lane tracking.
  + **Example:** OpenCV for image processing and TensorFlow for implementing deep learning models.
* **Machine Learning Frameworks**
  + **Purpose:** Develop and train AI models for various tasks such as object detection and path planning.
  + **Functionality:** Frameworks like TensorFlow and PyTorch are used to build and deploy neural networks and reinforcement learning models.
  + **Example:** TensorFlow for training convolutional neural networks (CNNs) and reinforcement learning algorithms.
* **Path Planning Algorithms**
  + **Purpose:** Compute optimal trajectories for parking maneuvers.
  + **Functionality:** Algorithms such as A\* and RRT (Rapidly-exploring Random Tree) help in finding the best path to maneuver into a parking space.
  + **Example:** Implementation of A\* for pathfinding and obstacle avoidance.

#### 3. ****Development Platforms****

* **Embedded Systems**
  + **Purpose:** Interface with hardware components and execute control algorithms.
  + **Functionality:** Platforms like Raspberry Pi or NVIDIA Jetson are used to run AI models and control the vehicle’s actuators.
  + **Example:** NVIDIA Jetson Xavier for running complex AI models and real-time data processing.
* **Simulation Tools**
  + **Purpose:** Test and validate the self-parking system in a virtual environment.
  + **Functionality:** Simulators like CARLA or Gazebo allow for the testing of algorithms and system performance under various conditions without physical deployment.
  + **Example:** CARLA for simulating urban driving scenarios and testing parking algorithms.
* **Control Systems**
  + **Purpose:** Manage vehicle actuators for steering, throttle, and braking.
  + **Functionality:** Implement control algorithms to execute parking maneuvers accurately.
  + **Example:** PID controllers for precise control of vehicle movements during parking.

#### 4. ****Data Management and Analysis Tools****

* **Database Management Systems**
  + **Purpose:** Store and manage data collected from sensors and system operations.
  + **Functionality:** Databases like PostgreSQL or MongoDB help in storing and querying environmental data, parking space information, and system logs.
  + **Example:** PostgreSQL for relational data storage and MongoDB for unstructured data.
* **Data Visualization Tools**
  + **Purpose:** Analyze and visualize sensor data and system performance.
  + **Functionality:** Tools like Matplotlib or Tableau provide graphical representations of data for better understanding and decision-making.
  + **Example:** Matplotlib for plotting sensor data and performance metrics.

By utilizing these tools effectively, the self-parking system can be developed to perform accurate and reliable parking maneuvers, enhancing both driver convenience and vehicle safety.

**5.OPERATION**

The operation of a self-parking system involves several stages, from detecting available parking spaces to executing the parking maneuver. Below is a step-by-step overview of how the system typically operates:

#### 1. ****Initial Setup and Calibration****

* **Sensor Calibration**
  + Calibrate ultrasonic sensors, cameras, Lidar, and radar to ensure accurate measurements and data collection.
  + Perform initial alignment of sensors to establish their spatial orientation and coverage.
* **System Initialization**
  + Boot up the embedded system and load the necessary software frameworks and AI models.
  + Establish communication between sensors, control systems, and the central processing unit.

#### 2. ****Parking Space Detection****

* **Environment Scanning**
  + **Sensors Active:** The system continuously scans the environment using ultrasonic sensors, cameras, Lidar, and radar.
  + **Data Collection:** Gather real-time data on nearby objects, obstacles, and available parking spaces.
* **Space Identification**
  + **Visual Analysis:** Computer vision algorithms process camera images to identify potential parking spaces.
  + **Sensor Fusion:** Integrate data from multiple sensors to create a comprehensive view of the surroundings and identify suitable parking spots.
* **Space Measurement**
  + **Dimensions Calculation:** Measure the dimensions of identified parking spaces to ensure the vehicle will fit.
  + **Validation:** Confirm that the space is clear of obstacles and suitable for parking.

#### 3. ****Path Planning****

* **Trajectory Calculation**
  + **Path Algorithms:** Use path planning algorithms, such as A\* or RRT, to calculate the optimal trajectory for maneuvering into the parking space.
  + **Obstacle Avoidance:** Incorporate data from sensors to avoid obstacles and ensure a smooth path.
* **Simulation**
  + **Path Verification:** Simulate the planned path to verify its feasibility and safety.
  + **Adjustments:** Make necessary adjustments to the trajectory based on real-time feedback and simulations.

#### 4. ****Parking Maneuver Execution****

* **Control Activation**
  + **Actuator Control:** Engage control systems to manage the vehicle’s steering, throttle, and brakes based on the planned path.
  + **Real-Time Adjustment:** Continuously adjust control inputs to respond to dynamic conditions and ensure accurate execution.
* **Maneuver Execution**
  + **Steering:** Adjust the steering angle to guide the vehicle into the parking space.
  + **Speed Control:** Regulate the throttle and brakes to control the vehicle’s speed and ensure smooth entry.
  + **Obstacle Detection:** Use sensors to monitor for any unexpected obstacles during the maneuver and make real-time corrections.

#### 5. ****Parking Confirmation****

* **Position Verification**
  + **Final Adjustment:** Check if the vehicle is correctly aligned within the parking space.
  + **Sensor Feedback:** Use sensors to confirm that the vehicle is positioned correctly and free of collisions.
* **System Feedback**
  + **Completion Signal:** Indicate successful parking to the driver, typically through visual or auditory feedback.
  + **Status Reporting:** Update the system with the parking status and any relevant information.

#### 6. ****Post-Parking Actions****

* **Data Logging**
  + **Performance Records:** Log data related to the parking maneuver for future analysis and system improvement.
  + **Error Tracking:** Record any issues or anomalies encountered during the parking process.
* **System Reset**
  + **Reinitialization:** Prepare the system for the next parking attempt or driving phase.
  + **Calibration Check:** Perform periodic checks and recalibration to maintain system accuracy and performance.

The self-parking system operates by seamlessly integrating multiple technologies and processes to automate the parking process. It involves detecting and measuring parking spaces, planning the optimal parking path, executing the maneuver with precise control, and ensuring accurate positioning. Through real-time data processing and continuous adjustments, the system aims to enhance convenience, safety, and efficiency in vehicle parking.

**6.Approach / Module Description / Functionalities**

The self-parking system can be structured into several key modules, each responsible for a specific aspect of the parking process. Here’s a detailed breakdown of the approach, module descriptions, and their functionalities:

#### 1. ****Sensor Module****

**Description:** The Sensor Module is responsible for collecting real-time data from various sensors integrated into the vehicle. This data provides a comprehensive view of the vehicle’s environment and is essential for detecting parking spaces and obstacles.

**Functionalities**

* **Ultrasonic Sensors**
  + **Distance Measurement:** Measure the distance to nearby objects, detecting proximity and avoiding collisions.
  + **Obstacle Detection:** Identify and alert the system to obstacles in close range.
* **Cameras**
  + **Visual Input:** Capture images and video of the surroundings for computer vision processing.
  + **Parking Space Detection:** Identify potential parking spaces and lane markings.
* **Lidar**
  + **3D Mapping:** Create detailed 3D maps of the environment for precise spatial understanding.
  + **Obstacle Mapping:** Detect and map obstacles and boundaries around the vehicle.
* **Radar**
  + **Object Detection:** Detect objects and assess their distance, particularly in low visibility conditions.
  + **Speed Measurement:** Track the speed of moving objects in the vicinity.

#### 2. ****Data Processing Module****

**Description:** The Data Processing Module handles the integration and analysis of data collected from the sensors. It uses computer vision and machine learning algorithms to interpret the data and make decisions about parking.

**Functionalities**

* **Sensor Fusion**
  + **Data Integration:** Combine data from multiple sensors to create a unified view of the environment.
  + **Enhanced Accuracy:** Improve the reliability of obstacle detection and space identification.
* **Computer Vision**
  + **Object Detection:** Use convolutional neural networks (CNNs) to detect and recognize objects, such as other vehicles, pedestrians, and parking space boundaries.
  + **Lane Recognition:** Identify and track lane markings and parking space lines.
* **Machine Learning**
  + **Training Models:** Train models to recognize and classify various objects and parking spaces.
  + **Learning and Adaptation:** Continuously improve the system’s performance through reinforcement learning and feedback.

#### 3. ****Path Planning Module****

**Description:** The Path Planning Module calculates the optimal trajectory for the vehicle to maneuver into the identified parking space. It ensures that the vehicle’s path avoids obstacles and aligns with the parking space dimensions.

**Functionalities:**

* **Trajectory Calculation:**
  + **Path Algorithms:** Use algorithms like A\* or RRT to compute the best path for parking.
  + **Obstacle Avoidance:** Plan a path that avoids detected obstacles and minimizes risk.
* **Simulation and Adjustment:**
  + **Path Simulation:** Simulate the planned path to verify its feasibility and safety.
  + **Real-Time Adjustment:** Adjust the path based on real-time data and environmental changes.

#### 4. ****Control Systems Module****

**Description:** The Control Systems Module manages the vehicle’s steering, throttle, and brakes to execute the parking maneuver based on the planned path.

**Functionalities**

* **Steering Control:**
  + **Directional Adjustment:** Control the steering angle to guide the vehicle into the parking space.
  + **Precision Maneuvering:** Ensure accurate alignment within the parking space.
* **Speed and Braking Control:**
  + **Throttle Regulation:** Manage acceleration to control the vehicle’s speed during parking.
  + **Brake Management:** Apply brakes to stop the vehicle at the correct position and prevent collisions.
* **Real-Time Monitoring:**
  + **Continuous Feedback:** Monitor the vehicle’s position and adjust control inputs in real-time to ensure smooth parking.

#### 5. ****User Interface Module****

**Description:** The User Interface Module provides feedback to the driver and allows for interaction with the self-parking system. It offers visual or auditory notifications regarding the parking process.

**Functionalities:**

* **Status Indication:**
  + **Feedback Display:** Show visual or auditory signals to indicate the status of the parking process.
  + **Completion Notification:** Notify the driver when the parking maneuver is successfully completed.
* **Manual Override:**
  + **Driver Control:** Allow the driver to manually override the system if needed.
  + **Emergency Stop:** Provide an option to stop the parking process in case of emergencies.

The self-parking system operates through a series of integrated modules, each contributing to the overall functionality of the system. The Sensor Module collects data, the Data Processing Module interprets it, the Path Planning Module calculates the optimal path, the Control Systems Module executes the maneuver, and the User Interface Module provides feedback and interaction. Testing and diagnostics ensure the system’s reliability and performance in various conditions

**7.IMPLEMENTATION**

The implementation of a self-parking system using AI involves a combination of hardware integration, software development, and system testing. Below is a detailed plan for implementing the self-parking system, covering hardware setup, software development, and testing phases.

#### 1. ****Hardware Setup****

**Step 1: Sensor Installation**

* **Ultrasonic Sensors:** Install ultrasonic sensors around the vehicle (front, rear, and sides) to measure distances and detect obstacles.
* **Cameras:** Mount cameras at strategic locations (front, rear, and sides) to capture visual data for object detection and space identification.
* **Lidar:** Place a Lidar sensor on the roof or other suitable location for 3D mapping and environment scanning.
* **Radar:** Install radar sensors at the front and rear for detecting objects, especially in low visibility conditions.

**Step 2: Embedded System Configuration**

* **Processing Unit:** Set up an embedded system (e.g., NVIDIA Jetson Xavier) to process data from sensors and run AI algorithms.
* **Power Supply:** Ensure a reliable power supply for all sensors and the processing unit.
* **Communication Interfaces:** Establish communication interfaces (e.g., CAN bus, Ethernet) to connect sensors with the processing unit.

#### 2. ****Software Development****

**Step 1: Sensor Data Collection**

* **Drivers and APIs:** Develop or integrate drivers and APIs for each sensor to facilitate data collection and communication with the processing unit.
* **Data Acquisition:** Implement software to continuously acquire data from all sensors and store it in a format suitable for processing.

**Step 2: Data Processing and Sensor Fusion**

* **Preprocessing:** Develop preprocessing algorithms to clean and normalize the raw sensor data.
* **Sensor Fusion:** Implement sensor fusion techniques to combine data from different sensors and create a comprehensive view of the environment.

**Step 3: Computer Vision Algorithms**

* **Object Detection:** Use convolutional neural networks (CNNs) and other machine learning models to detect and classify objects in camera images.
* **Space Identification:** Develop algorithms to identify and measure potential parking spaces using visual data and sensor fusion outputs.

**Step 4: Path Planning Algorithms**

* **Trajectory Calculation:** Implement path planning algorithms (e.g., A\*, RRT) to calculate the optimal trajectory for parking maneuvers.
* **Simulation and Testing:** Simulate the planned paths in a virtual environment to ensure feasibility and safety.

**Step 5: Control Systems Development**

* **Actuator Control:** Develop control algorithms to manage the vehicle’s steering, throttle, and brakes based on the planned trajectory.
* **PID Controllers:** Implement PID controllers to ensure smooth and precise control over the vehicle’s movements.

**Step 6: User Interface Development**

* **Status Display:** Design a user interface to display the system’s status, includi ng parking progress and error notifications.
* **Manual Override:** Implement options for manual override and emergency stop to allow driver intervention when necessary.

#### 3. ****System Testing and Validation****

**Step 1: Simulation Testing**

* **Virtual Environment:** Use simulators (e.g., CARLA, Gazebo) to test the system in various parking scenarios and conditions.
* **Performance Metrics:** Evaluate the system’s performance based on accuracy, time taken, and safety metrics.

**Step 2: Real-World Testing**

* **Controlled Environment:** Conduct initial testing in a controlled environment (e.g., test track) to validate system performance.
* **Urban Environment:** Gradually test the system in real-world urban environments to ensure robustness and reliability.

**Step 3: Continuous Improvement**

* **Data Logging:** Log data from all test runs to analyze performance and identify areas for improvement.
* **Algorithm Tuning:** Continuously tune algorithms based on test results to enhance accuracy and efficiency.
* **Feedback Loop:** Implement a feedback loop to update AI models and control systems with new data and learnings.

#### 4. ****Deployment and Maintenance****

**Step 1: Deployment Preparation**

* **System Integration:** Ensure all hardware and software components are seamlessly integrated and functioning correctly.
* **User Training:** Provide training and documentation for users to understand system operation and manual override procedures.

**Step 2: Maintenance Plan**

* **Regular Updates:** Schedule regular software updates to incorporate improvements and new features.
* **Sensor Calibration:** Periodically recalibrate sensors to maintain accuracy.
* **System Diagnostics:** Implement diagnostic tools to monitor system health and detect issues proactively.

**Step 3: Customer Support**

* **Support Channels:** Establish customer support channels to assist users with troubleshooting and system queries.
* **Feedback Collection:** Collect user feedback to continuously improve the system based on real-world usage.

Implementing a self-parking system using AI involves a structured approach, including hardware setup, software development, and extensive testing. The system relies on a combination of sensors, AI algorithms, and control mechanisms to detect parking spaces, plan optimal paths, and execute parking maneuvers with high precision and safety. Through continuous improvement and user feedback, the system can be refined to provide a reliable and efficient self-parking solution.

**CODING**

The self-parking system code, focusing on sensor integration, data processing, path planning, and control. This implementation uses Python and relevant libraries for simplicity and demonstration purposes.

#### 1. ****Sensor Integration****

**Example: Ultrasonic Sensor Integration**

import RPi.GPIO as GPIO

import time

# Setup

GPIO.setmode(GPIO.BCM)

TRIG = 23

ECHO = 24

GPIO.setup(TRIG, GPIO.OUT)

GPIO.setup(ECHO, GPIO.IN)

def get\_distance():

# Send trigger pulse

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

# Wait for echo start

while GPIO.input(ECHO) == 0:

pulse\_start = time.time()

# Wait for echo end

while GPIO.input(ECHO) == 1:

pulse\_end = time.time()

# Calculate distance

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration \* 17150

distance = round(distance, 2)

return distance

try:

while True:

dist = get\_distance()

print(f"Distance: {dist} cm")

time.sleep(1)

except KeyboardInterrupt:

GPIO.cleanup()

**Example: Camera Integration with OpenCV**

import cv2

# Open camera

cap = cv2.VideoCapture(0)

while True:

# Capture frame-by-frame

ret, frame = cap.read()

# Display the resulting frame

cv2.imshow('Camera Feed', frame)

# Break loop on 'q' key press

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

break

# Release the camera and close windows

cap.release()

cv2.destroyAllWindows()

#### 2. ****Data Processing and Computer Vision****

**Example: Object Detection with OpenCV and a Pretrained Model**

import cv2

import numpy as np

# Load YOLO model

net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")

layer\_names = net.getLayerNames()

output\_layers = [layer\_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

# Load camera feed

cap = cv2.VideoCapture(0)

while True:

ret, frame = cap.read()

height, width, channels = frame.shape

# Detecting objects

blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

outs = net.forward(output\_layers)

# Showing information on the screen

for out in outs:

for detection in out:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > 0.5:

# Object detected

center\_x = int(detection[0] \* width)

center\_y = int(detection[1] \* height)

w = int(detection[2] \* width)

h = int(detection[3] \* height)

# Rectangle coordinates

x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2.putText(frame, str(class\_id), (x, y + 30), cv2.FONT\_HERSHEY\_PLAIN, 3, (0, 255, 0), 3)

cv2.imshow('Object Detection', frame)

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

break

cap.release()

cv2.destroyAllWindows()

#### 3. ****Path Planning****

**Example: Simple Path Planning using A Algorithm\***

import heapq

class Node:

def \_\_init\_\_(self, parent=None, position=None):

self.parent = parent

self.position = position

self.g = 0

self.h = 0

self.f = 0

def \_\_eq\_\_(self, other):

return self.position == other.position

def \_\_lt\_\_(self, other):

return self.f < other.f

def astar(maze, start, end):

open\_list = []

closed\_list = set()

start\_node = Node(None, start)

end\_node = Node(None, end)

heapq.heappush(open\_list, start\_node)

while open\_list:

current\_node = heapq.heappop(open\_list)

closed\_list.add(current\_node.position)

if current\_node == end\_node:

path = []

while current\_node is not None:

path.append(current\_node.position)

current\_node = current\_node.parent

return path[::-1]

children = []

for new\_position in [(0, -1), (0, 1), (-1, 0), (1, 0)]:

node\_position = (current\_node.position[0] + new\_position[0], current\_node.position[1] + new\_position[1])

if 0 <= node\_position[0] < len(maze) and 0 <= node\_position[1] < len(maze[0]):

if maze[node\_position[0]][node\_position[1]] == 0:

children.append(Node(current\_node, node\_position))

for child in children:

if child.position in closed\_list:

continue

child.g = current\_node.g + 1

child.h = ((child.position[0] - end\_node.position[0]) \*\* 2) + ((child.position[1] - end\_node.position[1]) \*\* 2)

child.f = child.g + child.h

if add\_to\_open(open\_list, child) == True:

heapq.heappush(open\_list, child)

def add\_to\_open(open\_list, child):

for node in open\_list:

if child == node and child.g > node.g:

return False

return True

maze = [

[0, 1, 0, 0, 0, 0],

[0, 1, 0, 1, 1, 0],

[0, 0, 0, 1, 0, 0],

[0, 1, 0, 1, 0, 0],

[0, 1, 0, 0, 0, 0],

]

start = (0, 0)

end = (4, 5)

path = astar(maze, start, end)

print(path)

#### 4. ****Control Systems****

**Example: PID Controller for Steering**

class PIDController:

def \_\_init\_\_(self, kp, ki, kd):

self.kp = kp

self.ki = ki

self.kd = kd

self.prev\_error = 0

self.integral = 0

def compute(self, setpoint, measured\_value):

error = setpoint - measured\_value

self.integral += error

derivative = error - self.prev\_error

output = self.kp \* error + self.ki \* self.integral + self.kd \* derivative

self.prev\_error = error

return output

# Example usage

pid = PIDController(kp=1.0, ki=0.1, kd=0.05)

setpoint = 10 # Desired value

measured\_value = 7 # Current value

output = pid.compute(setpoint, measured\_value)

print(f"PID output: {output}")

### Integration Example

Here's an example of integrating sensor data, path planning, and control systems into a simple self-parking operation:

import cv2

import RPi.GPIO as GPIO

import time

import heapq

# Initialize sensors, actuators, and control systems

def initialize\_system():

GPIO.setmode(GPIO.BCM)

GPIO.setup(TRIG, GPIO.OUT)

GPIO.setup(ECHO, GPIO.IN)

cap = cv2.VideoCapture(0)

return cap

# Read sensor data

def get\_sensor\_data(cap):

ret, frame = cap.read()

distance = get\_distance()

return frame, distance

# Process sensor data and plan path

def plan\_path(frame, distance):

# Detect parking space

space\_detected = detect\_parking\_space(frame)

if space\_detected:

path = astar(maze, start, end)

return path

return None

# Execute parking maneuver

def execute\_parking(path, pid):

for point in path:

current\_position = get\_current\_position()

steering\_output = pid.compute(point, current\_position)

control\_vehicle(steering\_output)

# Main loop

cap = initialize\_system()

pid = PIDController(kp=1.0, ki=0.1, kd=0.05)

while True:

frame, distance = get\_sensor\_data(cap)

path = plan\_path(frame, distance)

if path:

execute\_parking(path, pid)

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

break

cap.release()

GPIO.cleanup()

cv2.destroyAllWindows()

This code provides a basic structure for implementing a self-parking system. In a real-world scenario, the implementation would require more advanced algorithms, rigorous testing, and integration with vehicle-specific hardware and software systems.

**10.CONCLUSION**

The implementation of a self-parking system using AI represents a significant advancement in automotive technology, aiming to enhance driver convenience, safety, and efficiency. Through the integration of multiple sensors such as ultrasonic sensors, cameras, Lidar, and radar, the system can comprehensively understand its environment, detect obstacles, and identify suitable parking spaces.

The core functionalities of the system—sensor data collection, data processing, path planning, and control—are meticulously designed to work seamlessly together. Advanced computer vision and machine learning algorithms enable the system to accurately detect objects and parking spaces, while sophisticated path planning algorithms ensure the vehicle can navigate safely and efficiently into the designated space. Control systems, including PID controllers, provide precise steering, acceleration, and braking to execute the parking maneuver smoothly.

The iterative process of system testing, involving both simulated and real-world environments, ensures the robustness and reliability of the self-parking system. Continuous data logging and feedback loops facilitate ongoing improvements, enhancing the system's performance over time.

Ultimately, the self-parking system using AI not only reduces the stress and effort associated with parking but also minimizes the risk of accidents and damage. As technology evolves, such systems will become more prevalent and advanced, contributing to the broader vision of autonomous driving and smart vehicles.

By implementing this self-parking system, we move a step closer to a future where vehicles are not just modes of transport but intelligent partners in our daily lives, capable of making driving safer, more efficient, and more enjoyable.