AI-Driven Autonomous Car

By: Anjan Kumar Gundinapalya Jayaramaiah

Instructor: Bahman Zohuri

Golden Gate University - Fall 2023

MSBA 326: Machine Learning for Predictive Analytics

Abstract

Using Convolutional Neural Networks (CNNs) for complex visual processing, utilizing Python's capabilities for smooth integration and control, and utilizing Raspberry Pi as the computing core are all necessary to develop an AI-driven autonomous vehicle. To develop an intelligent, self-navigating car that is capable of making decisions in real-time and recognizing its surroundings, this study investigates the interactions between these components.

Contents

1.Introduction 4

2.Foundation of the Project 5

3.Hardware Setup 7

4.Software Architecture 9

5.Training the CNN Model 11

6.Real-time Decision Making 14

7.Results and Demonstration 15

8.Conclusion 16

9.References 17

1. Introduction

The Project's Objective:

This project's main goal is to demonstrate how cutting-edge technology are combined to create an autonomous vehicle. We investigate the Raspberry Pi as the central processing unit, leverage the flexibility of Python for smooth integration and management, and apply Convolutional Neural Networks (CNNs) for vision-based decision-making.

What This Project Reveals:

By combining these technologies, we want to create an intelligent car that can recognize its environment, act in real time, and navigate on its own. A future where AI-driven mobility transforms transportation paradigms is being ushered in by the convergence of hardware, software, and sophisticated algorithms.

2. Foundation of the Project

Understanding Raspberry Pi in Autonomous Systems:

The Raspberry Pi serves as the cornerstone of our project, providing a compact yet robust computing platform essential for the autonomy of our vehicle. With its GPIO pins enabling hardware interfacing and a range of supported operating systems, the Raspberry Pi stands as the central processing unit for our autonomous car.

Python's Role in Autonomous Vehicle Development:

Python, renowned for its versatility and extensive libraries, plays a pivotal role in orchestrating the intelligence of our autonomous vehicle. Its simplicity, coupled with powerful machine learning and computer vision libraries, empowers us to seamlessly integrate control mechanisms and algorithms essential for real-time decision-making.

Basics of Convolutional Neural Networks (CNNs) for Vision Processing:

CNNs form the backbone of our car's vision system. Understanding these neural networks is crucial as they enable the vehicle to perceive and comprehend its environment. By training CNNs with extensive datasets, our car gains the ability to identify objects, recognize patterns, and navigate through complex environments with precision.

Integration of Technology:

The integration of these foundational elements—Raspberry Pi's computational capabilities, Python's flexibility and libraries, and CNNs' prowess in vision processing—serves as the bedrock for the intelligence and autonomy of our vehicle. This section lays the groundwork for the subsequent stages where we harness these technologies to build our AI-driven autonomous car.

3. Hardware Setup

1. Raspberry Pi 4:

The Raspberry Pi 4 serves as the brain of our autonomous vehicle. Its compact form factor, increased processing power, and GPIO pins make it an ideal choice for controlling the car's functionalities. We integrate the Pi to handle data processing, decision-making, and control mechanisms.

2. H-Bridge Motor Driver:

The H-bridge motor driver board acts as the interface between the Raspberry Pi and the motors. It facilitates bidirectional control of the motors, allowing precise movement and speed regulation. Wiring the motors through the H-bridge enables seamless control over the car's motion.

3. Chassis and Motors:

The chassis provides the structural support for our vehicle, offering mounting points for the motors and other components. The 4 motors, each connected to a wheel, are instrumental in propelling and maneuvering the car. Proper mounting ensures stability and mobility.

4. Camera Module:

Integrating a camera module enhances our vehicle's perceptual capabilities. Placed strategically, the camera captures visual data crucial for object recognition and environmental understanding. This visual input becomes vital for our CNN-based vision processing system.

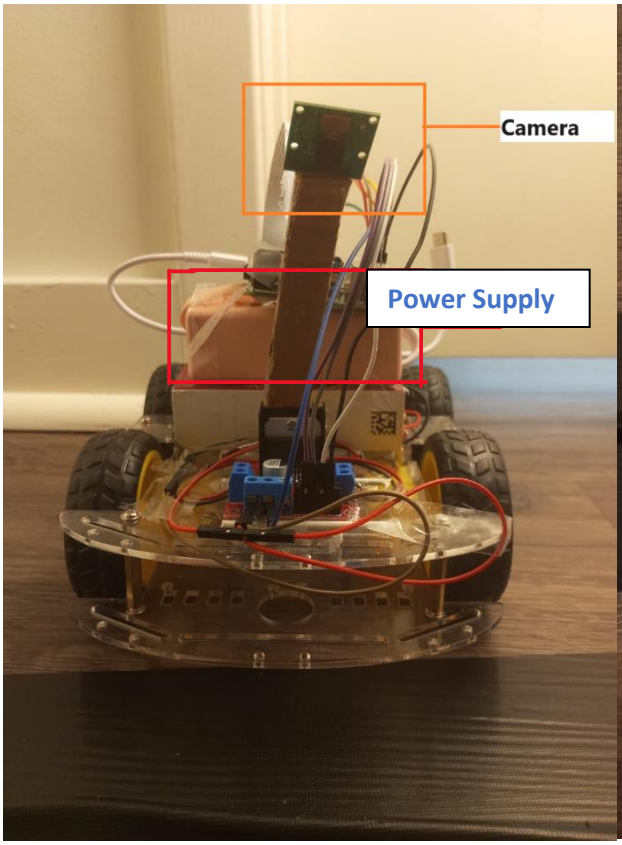
5. Power Supply:

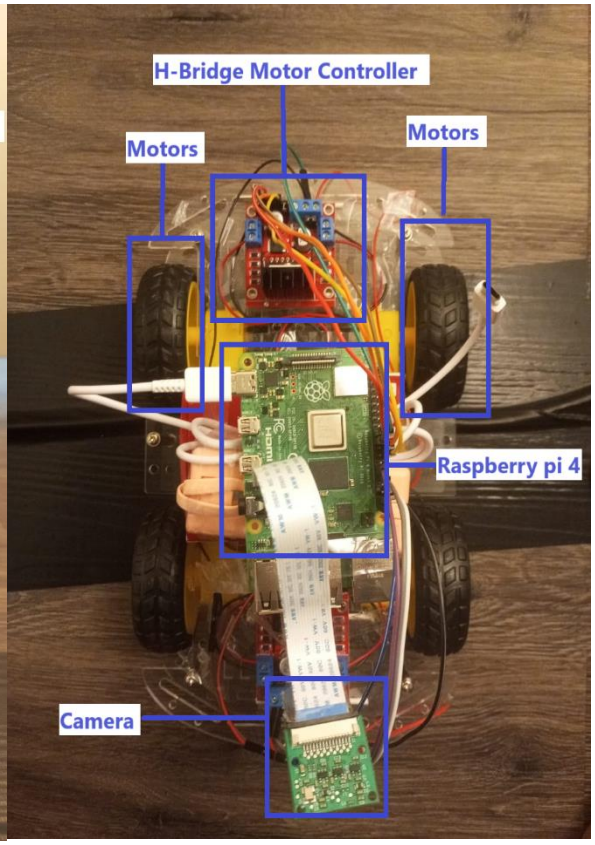
A reliable power supply is fundamental for the vehicle's operation. A suitable battery or power source, providing adequate voltage and current, powers both the Raspberry Pi and the motors. Ensuring the compatibility and stability of the power supply is imperative for smooth functionality.

6. Integration and Wiring:

Integrating these hardware components involves careful wiring and setup. Ensuring proper connections between the Raspberry Pi, H-bridge, motors, camera, and power supply is crucial for seamless communication and functionality.

The below figures 1,2 shows the overall hardware integration.





4. Software Architecture

1. Control Framework:

The software architecture orchestrates the control mechanisms essential for the autonomous vehicle's operation. Python, being our primary programming language, serves as the foundation for implementing these control systems. We utilize Python's libraries and functionalities to establish a robust and responsive control framework.

2. Interaction with Raspberry Pi:

Python scripts act as the intermediaries between the Raspberry Pi and various hardware components. Leveraging GPIO libraries such as RPi.GPIO, we establish communication channels to interface with the H-bridge motor driver, allowing precise control over the motors.

3. Sensor Integration and Data Processing:

The software architecture encompasses algorithms for handling sensor data, particularly from the camera module. Python scripts process the visual data captured by the camera, feeding it into the Convolutional Neural Network (CNN) for object recognition and environmental analysis.

4. CNN Integration:

Integration of the trained CNN model within the software architecture forms a pivotal component. Python's machine learning libraries enable seamless incorporation of the CNN, allowing the car to interpret visual data, recognize objects, and make informed decisions based on the perceived environment.

5. Real-time Decision Making:

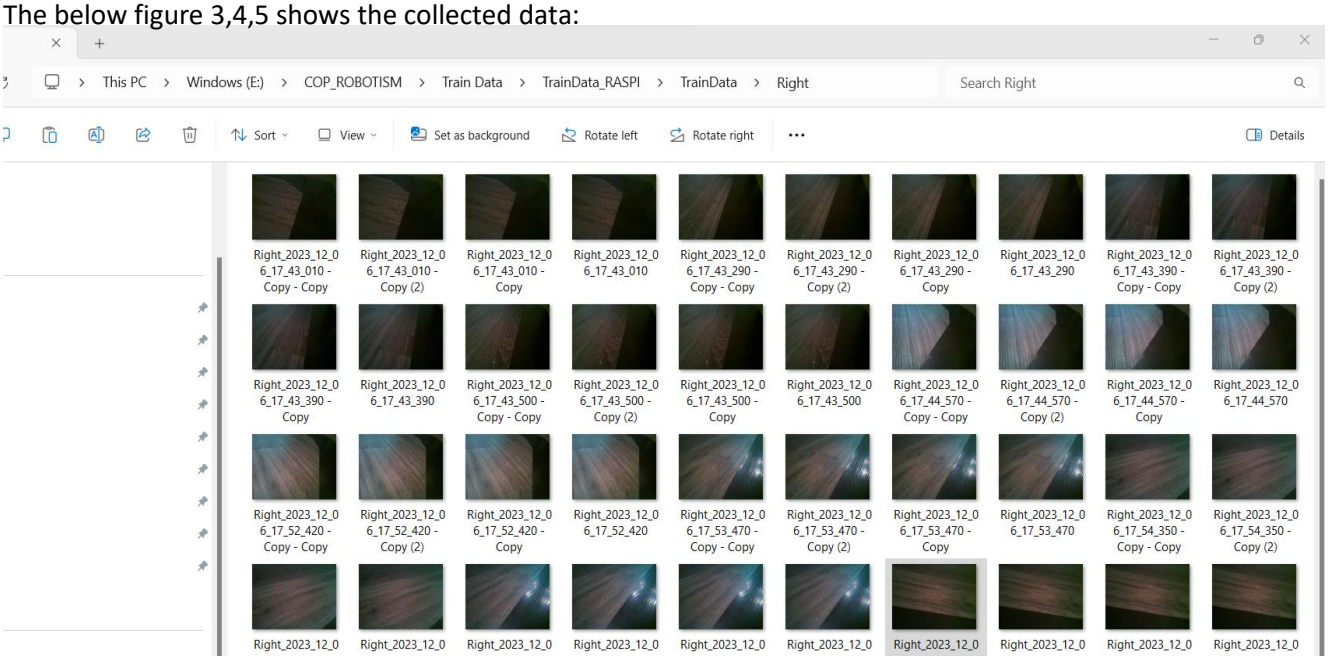
The culmination of these components facilitates real-time decision-making capabilities. The software architecture interprets visual input, processes it through the CNN, and executes control commands for the motors. This orchestrated process enables the autonomous vehicle to navigate and react to its surroundings autonomously.

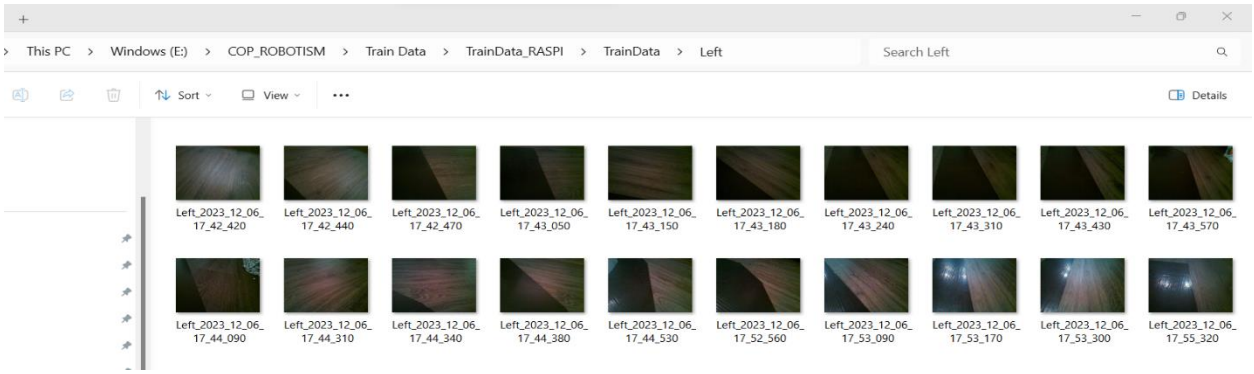
5. Training the CNN Model

1. Data Collection and Preparation:

Training a CNN necessitates a comprehensive dataset. We curate and prepare a diverse dataset comprising images relevant to the vehicle's operational environment. Proper labeling and categorization of these images ensure accurate training for object recognition.

The below figure 3,4,5 shows the collected data:





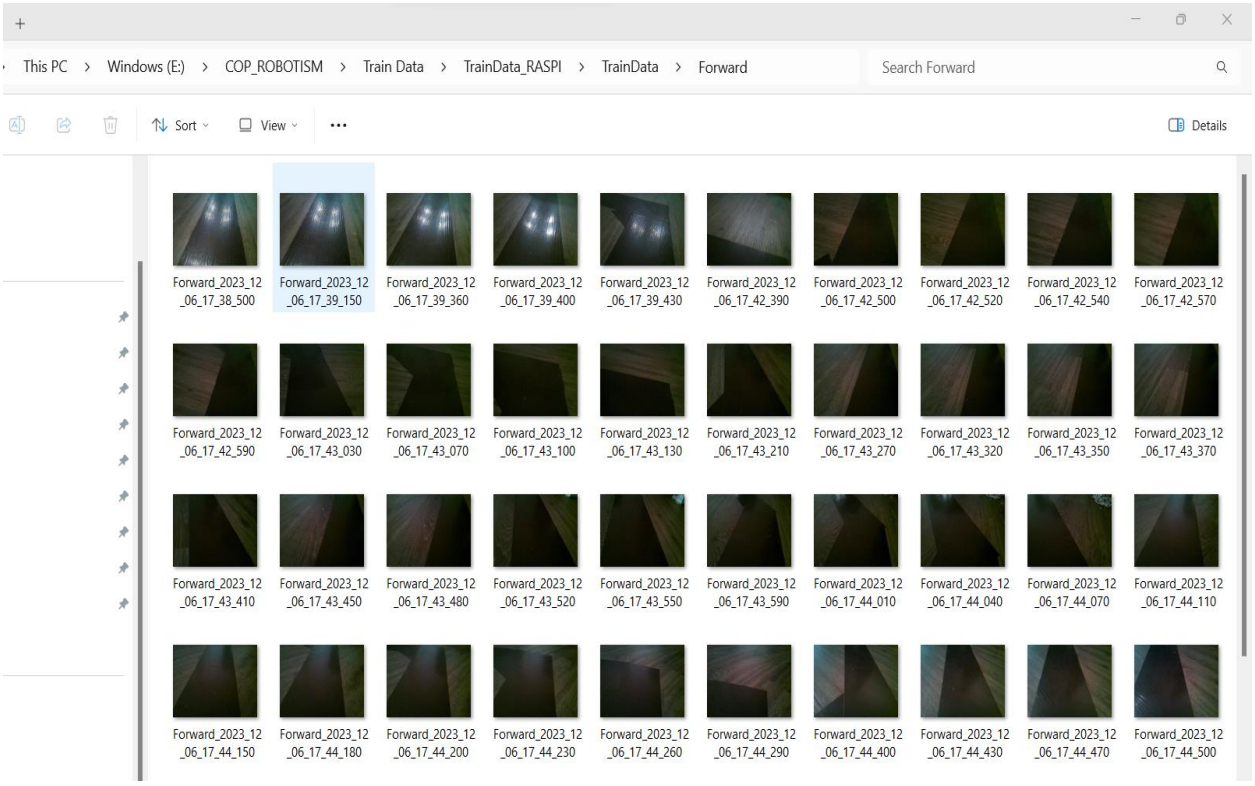


Figure 5. Forward Directions.

2. Preprocessing the Dataset:

Preparing the dataset involves preprocessing steps to standardize image sizes, remove noise, and augment the dataset for diverse scenarios. Techniques such as normalization, resizing, and augmentation enhance the CNN's ability to generalize patterns and features.

3. Architecture Selection and Training Setup:

We select an appropriate CNN architecture, considering factors like depth, complexity, and computational efficiency. Utilizing Python's machine learning libraries like TensorFlow or PyTorch, we set up the training environment, defining hyperparameters, loss functions, and optimization techniques.

4. Training Process:

The CNN is trained iteratively on the prepared dataset. The training process involves feeding batches of preprocessed images into the network, adjusting weights and biases to minimize error, and iteratively optimizing the model's performance through forward and backward propagation.

5. Validation and Fine-tuning:

Validation sets are employed to assess the model's performance on unseen data, preventing overfitting and ensuring generalizability. Fine-tuning involves adjusting model parameters based on validation results, optimizing for accuracy and robustness.

6. Model Evaluation and Testing:

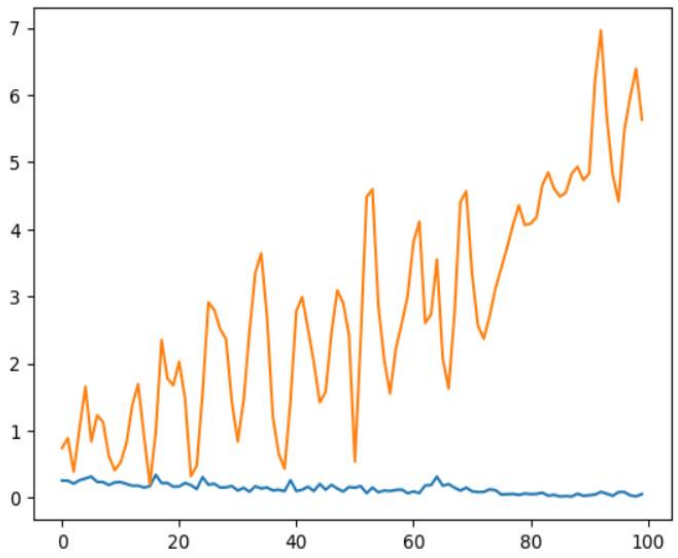
The trained CNN undergoes rigorous evaluation, testing its ability to recognize objects and patterns accurately. We assess metrics such as accuracy to gauge the model's efficacy before deploying it in the vehicle.

The below figure 6 illustrates the model performance.

```
In [ ]: #Evaluating the performance
        from matplotlib import pyplot as plt

        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
```

```
Out[ ]: [<matplotlib.lines.Line2D at 0x20188c9f130>]
```



6. Real-time Decision Making

1. Visual Data Processing:

The first step in real-time decision-making involves capturing visual data through the onboard camera. The acquired images are processed in real-time by the trained CNN, extracting features and patterns crucial for environmental understanding.

2. Object Recognition and Classification:

Utilizing the CNN's learned features, the vehicle identifies and classifies objects within its field of vision. Whether it's detecting road , the CNN's object recognition capabilities play a pivotal role in understanding the surroundings.

3. Decision Inference and Control Commands:

The interpreted visual data feeds into the decision-making process. The CNN infers decisions based on recognized objects and their spatial relationships. This inference generates control commands such as **Left, Right and Forward**, instructing the vehicle on appropriate actions, such as adjusting speed, changing direction, or stopping.

4. Feedback Loop for Autonomous Navigation:

The control commands generated by the CNN initiate a feedback loop, where the vehicle's actuators, directed by the Raspberry Pi, execute the instructed actions. This continuous loop ensures the vehicle navigates autonomously, responding to real-time environmental changes based on the CNN's decisions.

7. Results and Demonstration

The culmination of hardware integration, software frameworks, and CNN-based vision processing resulted in a functional prototype of an AI-driven autonomous vehicle. The vehicle showcased capabilities in perceiving its surroundings, recognizing objects, and making real-time decisions for navigation.

The below URI has the working demonstration of the car.

https://youtu.be/XYsQMrzDtBg

The below gif illustrates sample of a working car.

8. Conclusion

Summary of Achievements:

This project marks a significant milestone in the development of an AI-driven autonomous vehicle. Leveraging the computational power of Raspberry Pi, the versatility of Python, and the advanced vision processing capabilities of Convolutional Neural Networks (CNNs), we've successfully created a functional prototype capable of autonomous navigation.

Technological Synergy:

The integration of these cutting-edge technologies enabled our vehicle to interpret its environment, recognize objects, and navigate autonomously. The synergy between hardware components and software frameworks established a robust foundation for real-time decision-making and responsive control mechanisms.

Challenges and Innovations:

Throughout the project, challenges were met with innovative solutions, whether in hardware integration, CNN training, or software architecture design. Overcoming these hurdles allowed us to refine the vehicle's capabilities and drive progress in the field of autonomous mobility.

9. References

https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

https://docs.arducam.com/Raspberry-Pi-Camera/Native-camera/Libcamera-User-Guide/

https://www.tensorflow.org/tutorials/images/cnn

https://docs.python.org/3.12/library/index.html

https://projects.raspberrypi.org/en/projects/build-a-buggy/1