VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

LOGIC DESIGN PROJECT

CO3091 - CC02

ASSIGNMENT REPORT

AI-Integrated Image Detection for Autonomous Mecanum Robot Control

---

Instructor: Dr. Lê Trọng Nhân

Students: Nguyễn Văn Thành Đạt - 2152055
Nguyễn Quang Thiện - 2152994

HO CHI MINH CITY, JANUARY 2024

Contents

Member list & Workload

No.	Fullname	Student ID	Problems	Contribution
1	Nguyễn Văn Thành Đạt	2152055	- Train AI model - Generate AI detection - Generate Line detection	50%
2	Nguyễn Quang Thiện	2152994	- Adafruit setting and server update - Design User Interface - Generate Line detection	50%

Abstract

The project, titled "AI-Integrated Image Detection for Autonomous Mecanum Robot Control", will represent an innovative exploration at the intersection of artificial intelligence (AI), Internet of Things (IoT), and robotics. The primary objective is to imbue a Mecanum robot with autonomous capabilities through the integration of advanced AI technologies. That means the project encompasses the training of a sophisticated AI model using Teachable Machine, enabling the recognition of diverse traffic signs. This AI model is integrated with the CamESP32 and Adafruit server, establishing a reliable channel for real-time image capture and data transfer.

A nuanced line-following algorithm is developed to empower the Mecanum robot to autonomously navigate predefined paths. The project also includes the implementation of a user-friendly interface using Tkinter, offering manual control options and real-time visualization of captured images. The integration of AI image detection with autonomous control enables the robot to make informed decisions based on visual cues, showcasing dynamic and context-aware actions.

Challenges faced during the project, such as algorithm fine-tuning and integration complexities, are overcome through iterative processes. Valuable lessons learned include the importance of adaptive control strategies and user interface optimization. The project concludes with a forward-looking perspective, envisioning the application of the AI-integrated Mecanum robot in agriculture for tasks such as plant watering, daily plant status checks, and obstacle removal.

1 Background

In these days and ages, the unrelenting progress of Artificial Intelligence (AI) and the Internet of Things (IoT) has brought about industry-wide changes by opening doors for innovative and flexible solutions to challenging issues. Humanity can observe and experience the effects of all current technology advancements, leading to the fact that robots help to greatly simplify and expedite working processes. Specifically, mobile robots themselves represent a new frontier in robotics technology development and are a common example of the advancement and use of automation control and artificial intelligence technologies. The autonomous vehicles developed by firms such as Tesla are a prominent example of this technical advancement. Beyond only turning the automobile industry upside down, Tesla's inventions which are era groundbreaking like its self-driving capabilities have set the standard for how AI may be fluidly incorporated into daily life. The groundbreaking innovations in autonomous vehicles have redefined the landscape of transportation and technology. Through its Autopilot and Full Self-Driving (FSD) features, Tesla has demonstrated the feasibility and advantages of integrating artificial intelligence into vehicle control systems. The deployment of many advanced sensors, machine learning algorithms, and real-time data processing enables Tesla cars to navigate and make decisions autonomously. This has not only increased road safety by mitigating human errors but has also paved the way for future advancements in the automation of driving powered by AI. Tesla's success serves as a testament to the transformative impact of merging AI with robotics.

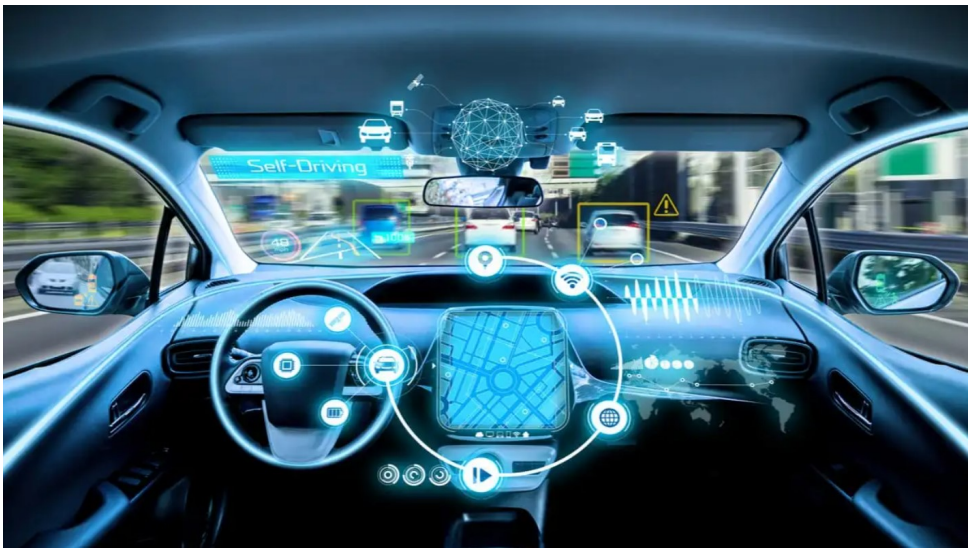

Figure 1: An illustration of autonomous car features

By harnessing the lessons learned from achievements in AI and IoT field, the imperative to contribute to the ongoing technological narrative forms the crux of our project - **AI-Integrated Image Detection for Autonomous Mecanum Robot Control**. In the realm of small-scale robotics, particularly with the Robot Mecanum, we embark on a journey to harness the power of AI with a view to bringing autonomous behavior to a compact and versatile robotic platform, inspired by the strides made in the field of autonomous vehicles as Tesla's self-driving capabilities. The intersection of image detection and Mecanum robot control presents a unique opportunity to tackle challenges related to both perception and navigation. Furthermore, the project aligns with contemporary trends in smart robotics, showcasing the potential for using in Agriculture in the upcoming future. While our project may operate at a miniature level, the potential implications echo the broader societal transformations initiated by larger autonomous systems with the recognition of the profound impact AI-driven autonomy. This endeavor is not merely an exploration of robotics but a testament to the scalable influence of AI in shaping the future.

Regarding to the overarching goals and scope of the project, it is plainly evident that the target is to create a synergistic fusion between AI image detection and autonomous robot control, thereby achieving a level of sophistication that allows the robot to make informed decisions based on its visual perception. At its core, the project aims to tackle the challenges associated with enabling a robot to not only recognize traffic signs—Left, Right, Straight, Stop, U-turn—but also to autonomously respond to these visual cues by dynamically adjusting its path. Hence, the scope of the assignment is expansive, encompassing the design, implementation, and integration of multiple components, including:

- **Integrate advanced AI technologies and showcase a context-aware robotic system**: Fuse AI capabilities into the fabric of autonomous Mecanum robot control, creating a context-aware robotic system capable of making informed decisions based on AI insights.
- **Utilize a meticulously trained AI model**: Make use of a well-trained AI model that can identify and decipher different types of traffic signals.
- **Implement sophisticated line-following mechanisms**: Deploy many intricate line-following mechanisms to enhance precision and adaptability in the robot's navigation.
- **Incorporate a user interface (UI)**: Develop an interactive UI that serves as a platform for both automatic control and real-time monitoring.

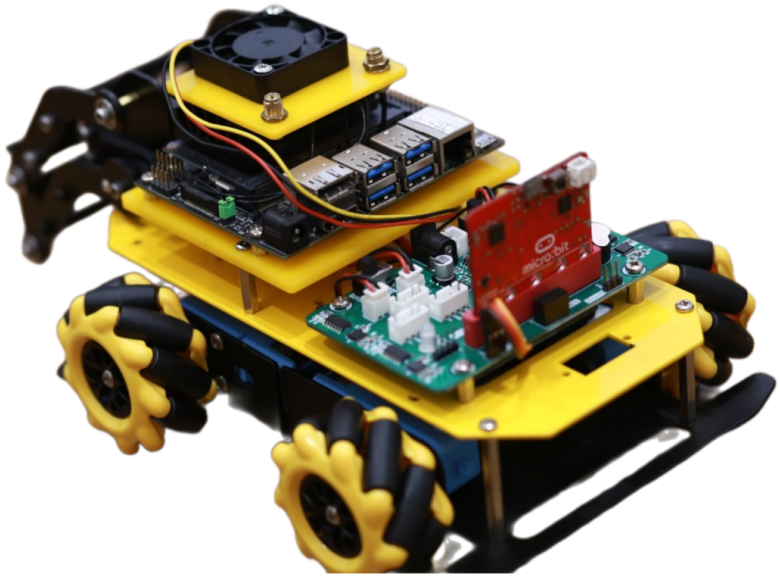

Figure 2: Robot Mecanum

Therefore, the project unfolds at the intersection of advanced AI technologies and autonomous robotics, drawing inspiration from the pioneering works and achievements in AI and IOT field all around the world. The integration of cutting-edge AI image detection with the nimble movements of a Mecanum robot sets the stage for a context-aware robotic system capable of interpreting and responding to real-world cues. With a focus on traffic sign recognition and autonomous decision-making, the project aspires to showcase the potential of AI-driven automation in enhancing the adaptability and precision of robotic systems. Also exploring the complex relationship between AI insights and real-time motor changes, the journey promises to be both technologically innovative and a glimpse into the future of smart and responsive robotic navigation for society.

2 Theory and Analysis

2.1 Robot Mecanum

Mecanum Wheel, also known as Omni Wheel, is a cutting-edge methodology designed with motion in either direction, which offers unparalleled advantages, providing exceptional maneuverability, precise navigation, and enhanced versatility. With the ability to move in various directions fluidly, these robots can navigate complex environments with ease, making them ideal for applications in industries ranging from logistics and manufacturing to healthcare and beyond. The wheel-type movement method maximizes efficiency and adaptability, contributing to increased productivity and improved operational capabilities in diverse robotic applications.

The Mecanum Wheel has come to light as a crucial component in improving robot dexterity and agility. Mecanum Wheels allow robots to move omnidirectionally with amazing accuracy because of its unusual design, which consists of drum-shaped rollers positioned at a 45-degree angle to the axis of its rotation. Based on which wheels revolve in which direction, Robot Mecanum may move forward, backward, sideways, diagonally, turn left, turn right, or it can spin in stationary. The Omni platform's range of flexibility makes it easy to operate in any area, especially around curves, narrow lanes, and intricate passageways.

Figure 3: Mecanum Wheel

Mecanum wheels have three different degrees of freedom. The wheel spins on three axes: its own, driven by the motor; it also rotates around the point where it contacts the ground; and finally, it spins around its own axis. While the rollers around the wheel are free to move along their separate axes, the wheel revolves typically along a path perpendicular to the drive shaft under the motor's power.

By utilizing the cooperative rotation speed and steering between three or more Mecanum wheels, it is possible to synthesize moment in any direction. This allows the platform to move in any direction and achieve full range of motion for in-plane mobility. The rollers on the wheel body's circumference, however, are subject to large axial forces, are more prone to damage, and generally slide when the wheels move because of the oblique distribution of the rollers on the Mecanum wheel, which causes the force acting on the rollers to be inconsistent with the direction in which the wheels are advancing. Its processing and manufacturing are relatively difficult, and the point where the wheel and ground make contact moves

periodically on the cylindrical surface, which is prone to vibration on uneven ground. However, these issues can be partially resolved with advancements in manufacturing technology.

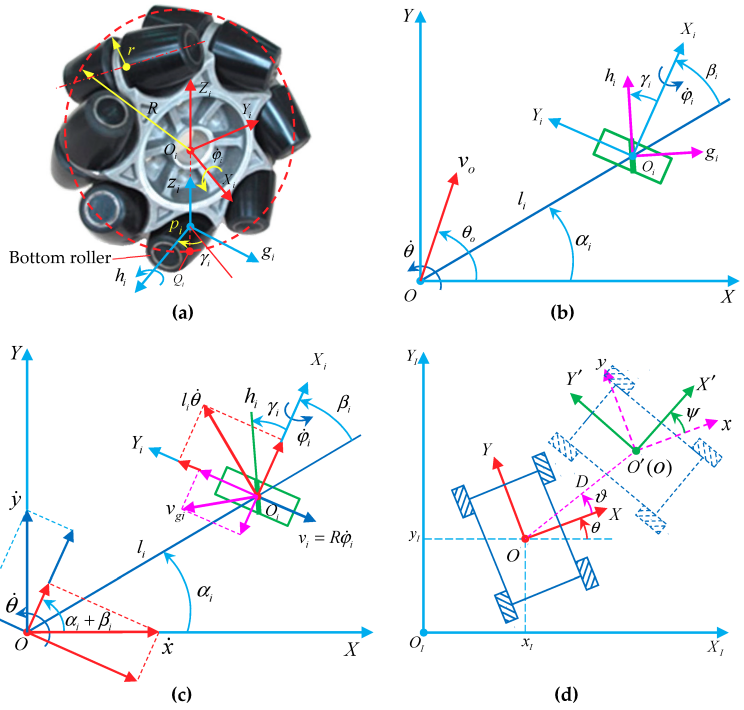

Figure 4: Kinematic constraints of a Mecanum wheel and the coordinate systems of a mobile system

2.2 Overview about Artificial Intelligence

Artificial Intelligence (AI)

- **Definition:** AI is the field of research and development in computer science aimed at creating machines capable of performing intelligent tasks similar to humans.
- **Scope:** AI includes both machine learning and deep learning, as well as other methods such as logical decision-making, natural language processing, computer vision, and many other areas.

Machine Learning (ML)

- **Definition:** Machine learning is a method within the AI field where computers learn from data they have experienced without being explicitly programmed for each task.
- **Scope:** Machine learning can use various techniques such as supervised learning, unsupervised learning, and reinforcement learning to create predictive or classification models.

Deep Learning (DL)

- **Definition:** Deep learning is a machine learning technique that layers algorithms and computing units—or neurons—into what is called an artificial neural network to learn complex features and represent abstract levels of data. (Take inspiration from the structure of the human brain)
- **Scope:** Deep learning is often used in tasks such as image recognition, natural language processing, and many high-complexity applications.

In summary, AI is the overarching field that encompasses both machine learning and deep learning. Machine learning is a subset of AI, and deep learning is a specific technique within machine learning that utilizes deep neural networks for learning complex representations from data.

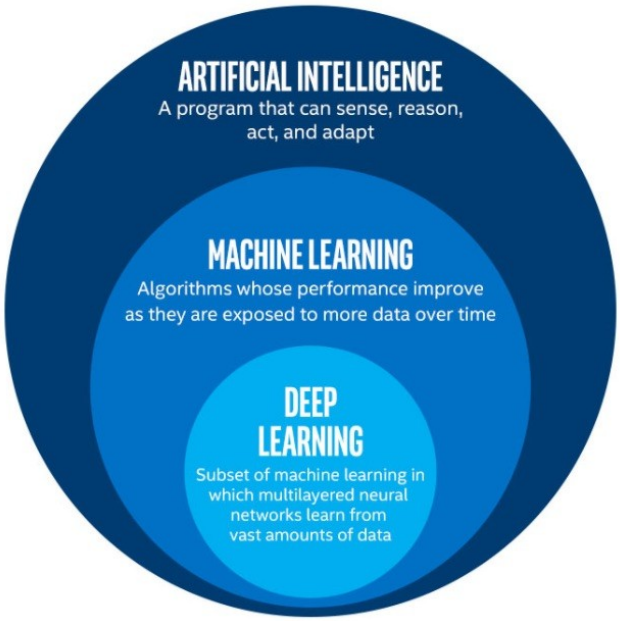

Figure 5: Relationships between AI, DL, and ML

Nowadays, AI is receiving growing attention from experts and is advancing rapidly, prominently featured in numerous breakthroughs spanning from AI to Deep Learning. Thanks to Deep Learning, computers are demonstrating enhanced capabilities, approaching or surpassing human-level performance. The diagram above illustrates that Deep Learning is a subset of Machine Learning. Machine Learning encompasses a diverse set of robust methods and algorithms, while Deep Learning specifically utilizes deep neural networks to autonomously learn complex data representations, empowering computers to address intricate problems. This capability extends to effectively harnessing large datasets (Big Data) with remarkable accuracy.

2.3 Deep Learning: Convolutional Neural Network (CNN)

Teachable Machine's AI model (using TensorFlow and Keras) is built based on convolutional neural networks (CNN - Convolutional Neural Networks) - one of the advanced Deep Learning models, helping us build intelligent systems with high accuracy. TensorFlow is a powerful library for mining and training learning models, while Keras is a high-level API with a reasonable depth of integration within TensorFlow to simplify the process of building and training models.

Convolutional Neural Network (CNN) is a term used to refer to convolutional neural networks. This is a highly accurate model and is commonly used in Deep learning. The idea behind this is that we will pass the image through filters before training the neural network. After going through the filter, the image's characteristics are clearly shown and we will use them to identify the image.

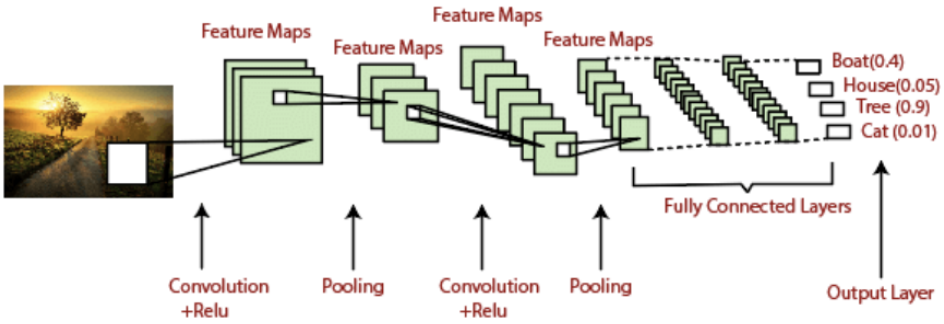

Figure 6: Architecture of CNN Algorithm

The input to the model is an image. Then comes the convolutional layer, which means we let the first image perform convolve with the Kernels (filters) using the Kernel sliders from left to right from top to bottom. Also, select arguments, apply filters to jumps, padding if needed, and apply non-linear activation to matrix image.

After each activation function, we will receive information for the next layers. There are activation functions like tanh and sigmoid. However, the Relu function is the most popular because it gives better performance. The formula of the Relu function is $\mathbf{f(x) = max(0, x)}$ which ensures the output data is not negative because the data we find is a linear property with non-negative values.

In the next step, we conduct Pooling to reduce the image size but still retain important information. Types of Pooling information are Max Pooling, Average Pooling, and Sum Pooling. The number of convolutional layers can have more or less optional usage targets that should be chosen accordingly. Finally, we build the output and data input into Fully Connected layers.

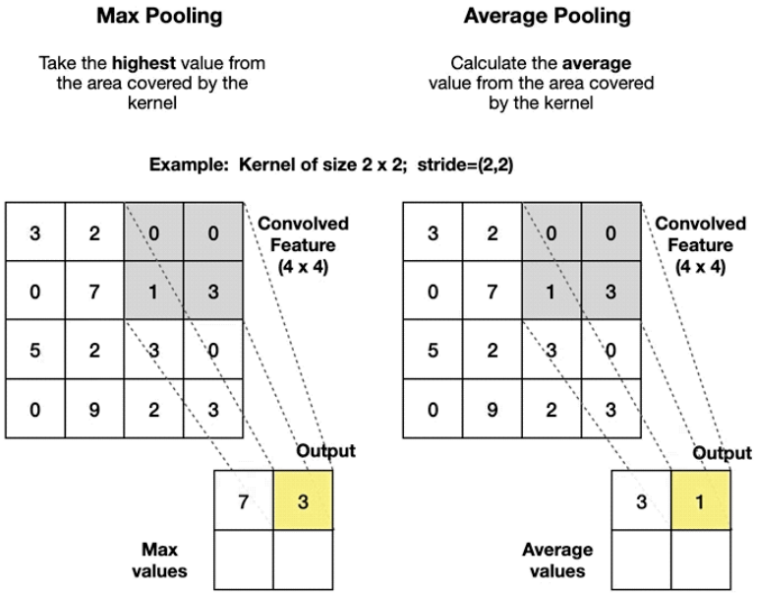

Figure 7: Example of Max Pooling and Average Pooling

The consequence of adding pooling layers is the reduction of overfitting, increased efficiency, and faster training times in a CNN model. While the max pooling layer draws out the most prominent features of an image, average pooling smooths the image retaining the essence of its features. In addition, Global pooling layers often replace the Flatten or Dense output layers.

2.4 Proportional–integral–derivative (PID) Controller and Algorithm

The PID algorithm, which stands for Proportional-Integral-Derivative, is a feedback control system algorithm widely used in engineering and industrial control systems. It aims to regulate a process by continuously adjusting a control output based on the difference between a desired setpoint and the current process variable.

Here are the three components of the PID algorithm:

Proportional (P)

- The proportional term is directly proportional to the current error, which is the difference between the setpoint and the process variable. The proportional action contributes to a control output that is proportional to the current error. This term provides an immediate response to changes in the system which can adjust the error to the lowest possible level, increase response speed, reduce overspeed, and limit oscillation modes.

$$P = K_p \times e(t)$$

where:

- P is the proportional term,
- K_p is the proportional gain (a tuning parameter),
- $e(t)$ is the current error.

Integral (I)

- The integral term is proportional to the accumulated sum of past errors over time. It helps eliminate steady-state errors and responds to long-term trends. The integral action accumulates the error over time, pushing the system towards the setpoint.

$$I = K_i \times \int_0^t e(\tau)\, d\tau$$

where:

- I is the integral term,
- K_i is the integral gain,
- $e(\tau)$ is the error at time τ,
- t is the current time.

Derivative (D)

- The derivative term is proportional to the rate of change of the error. It helps prevent the system from overshooting the setpoint and provides damping. The derivative action anticipates future behavior based on the rate of change of the error.

$$D = K_d \times \frac{de(t)}{dt}$$

where:

- D is the derivative term,
- K_d is the derivative gain,
- $\frac{de(t)}{dt}$ is the rate of change of the error for time.

The final control output is the sum of the proportional, integral, and derivative terms:

$$\text{Control Output} = P + I + D$$

The values of the tuning parameters K_p, K_i, and K_d are adjusted based on the specific characteristics of the controlled system to achieve the desired control performance.

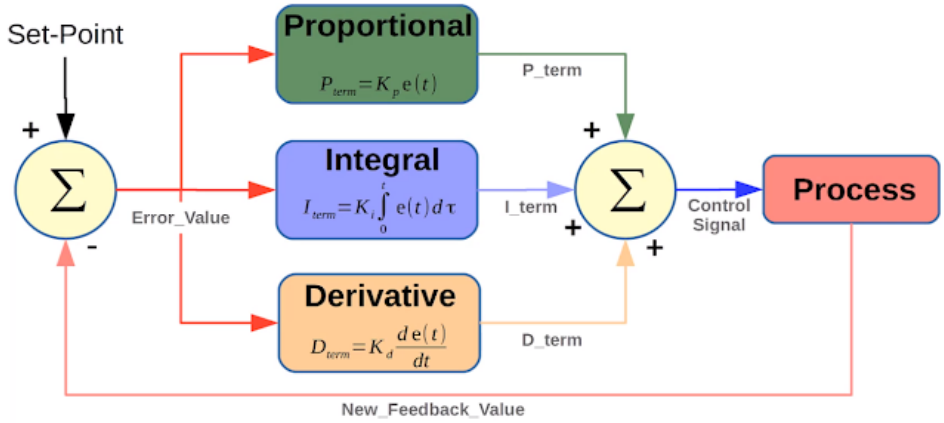

Figure 8: PID Algorithm

2.5 Methodologies

2.5.1 PyCharm

PyCharm, a powerful integrated development environment (IDE), emerges as a key tool for training AI models, particularly in scenarios where Mecanum robots navigate dynamic environments. The focus of this report lies in harnessing PyCharm's capabilities to train an AI model dedicated to identifying and responding to traffic signs. We can consider one of the most useful characteristics of this modern method inside the project:

- **PyCharm's Robust IDE Environment:** PyCharm provides users a robust environment that facilitates seamless development, debugging, and deployment of AI models. Its intuitive interface and comprehensive toolkit make it an ideal choice for engineers and developers working on complex robotics projects.
- **Training AI for Traffic Sign Recognition:** The integration of AI becomes particularly crucial when designing Mecanum robots that need to interpret and respond to traffic signs. PyCharm's support for popular AI frameworks allows engineers to train models capable of recognizing and interpreting traffic signs in real-time.
- **Enabling Mecanum Robots to Navigate Traffic:** As Mecanum robots become integral to smart urban mobility, the ability to interpret and respond to traffic signs is paramount. PyCharm, through its AI model training capabilities, empowers these robots to navigate through traffic scenarios, ensuring safety and efficiency in dynamic environments.

Figure 9: PyCharm

2.5.2 Teachable Machine

Teachable Machine stands at the forefront of accessible and user-friendly platforms for training (ML) Machine Learning models, making it an invaluable resource for initial stages in the development of AI applications. In the context of preparing AI models for Mecanum robot control, Teachable Machine serves as a foundational step before transitioning to advanced environments like PyCharm:

- **Fostering Accessibility for Novices in AI:** Teachable Machine offers an easy-to-use interface that eliminates the need for any prior coding knowledge, democratizing the AI training process. It enables machine learning novices to participate in model training with ease.
- **Intuitive Training for Image Recognition:** With the help of supplied examples, users can easily train models on Teachable Machine, which is good at picture recognition tasks. This facilitates robust model generation through simple image-based training in the context of Mecanum robots detecting traffic signs.

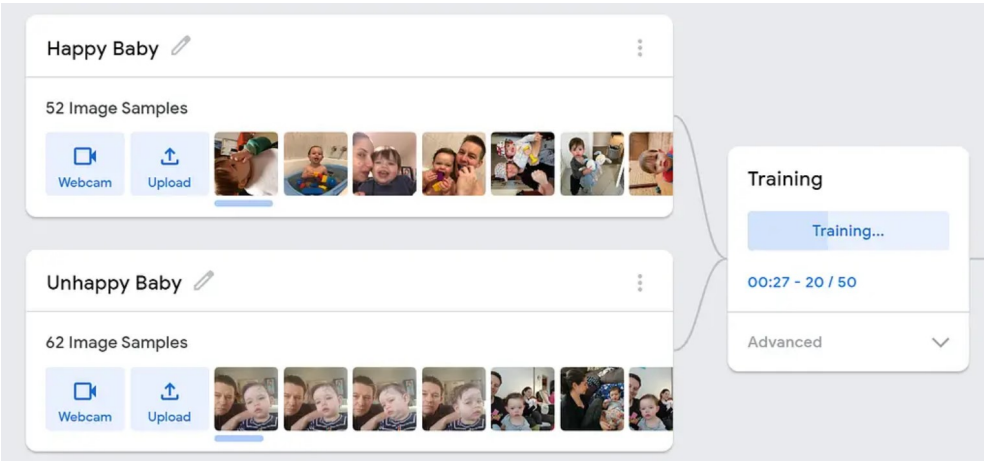

Figure 10: An example of Teachable Machine with visual machine learning

Upon successful training, Teachable Machine generates essential files, including the crucial **keras models** and **labels**, which encapsulate the learned patterns and corresponding labels. Therefore, Mecanum robots' initial preparation for intelligent responses to traffic signs is greatly aided by Teachable Machine, which navigates the delicate confluence of human accessibility and AI model efficacy.

2.5.3 Adafruit

Adafruit, renowned for its comprehensive Internet of Things (IoT) solutions, has emerged as a global leader in innovation, providing makers and engineers with easily accessible tools, guides, and parts. As we embark on this report, we discover the features and functions of Adafruit, specifically focusing on the utilization of its versatile Feeds feature for transmitting vital data from the outputs of traffic sign detection into a centralized dashboard.

For **Feeds - An Engine for Data Transmission**, Adafruit's Feeds feature serves as the engine powering the transmission of critical information. In this project, we may initialize some useful variables for capturing AI image recognition text, pushing images to the server, and denoting the accuracy of the image type, so as to form the bedrock of this data exchange mechanism.

Also, we can utilize its notable and prominent feature **Dashboard Creation for Comprehensive Insights**, which is the heart of Adafruit's integration lying in the generation of a dynamic dashboard. This centralized hub is designed to provide a holistic view of the Mecanum robot's interactions with traffic signs. By leveraging the capabilities of Adafruit's Feeds, the dashboard becomes an invaluable tool for monitoring and analyzing real-time data.

On the other hand, with **Enhanced Layout Customization**, Adafruit's dashboard design is not merely functional but also highly customizable. Through the incorporation of blocks such as Live Image, AI Detection, Accuracy, and History, the layout is tailored to offer a visually intuitive representation of traffic sign encounters on the road. This enhances the interpretability of data for users, allowing for swift decision-making based on accurate and timely insights.

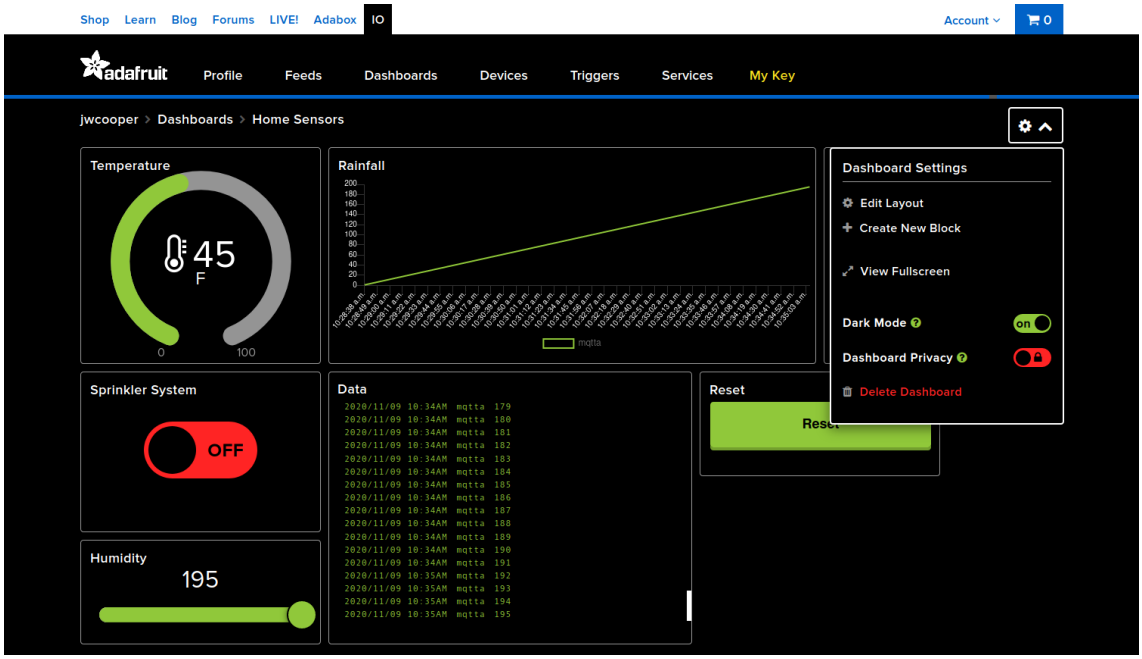

Figure 11: An example of Dashboard for Home Sensors

In the process of exploring Adafruit's Feeds and building a feature-rich dashboard, the paper seeks to clarify how IoT technologies and AI-powered robotics may work together harmoniously to create a more connected and knowledgeable future.

2.5.4 Arduino IDE

Arduino Integrated Development Environment (IDE), launched in 2005, stands as a cornerstone in the landscape of embedded systems, offering a user-friendly platform that plays a pivotal role in developing and programming microcontroller-based projects. The project will aim to explore the multifaceted functionalities of Arduino IDE, focusing on its significance in facilitating flawless access to the CamESP32's IP address and enhancing the development process for Internet of Things (IoT) applications.

Arduino has garnered widespread acclaim for its simplicity and effectiveness. It provides a versatile space where developers of varying expertise levels can write, compile, and upload code to a range of microcontrollers, making it an ideal choice for projects involving the CamESP32 and other IoT devices.

Figure 12: Arduino IDE

With the principal purpose of **CamESP32 Connectivity**, Arduino IDE proves instrumental when establishing communication with the CamESP32. Through Arduino's libraries and tools, developers can efficiently navigate the complexities of networking protocols and seamlessly access the CamESP32's IP address. This connectivity lays the foundation for developing IoT applications that demand real-time data exchange and remote device control. Moreover, one of Arduino IDE's strengths lies in its extensive library support. This vast collection of pre-written code modules simplifies the programming process, enabling developers to focus on the specific functionalities they want to implement with the CamESP32. Such a rich repository facilitates accelerated development and ensures a robust framework for IoT projects.

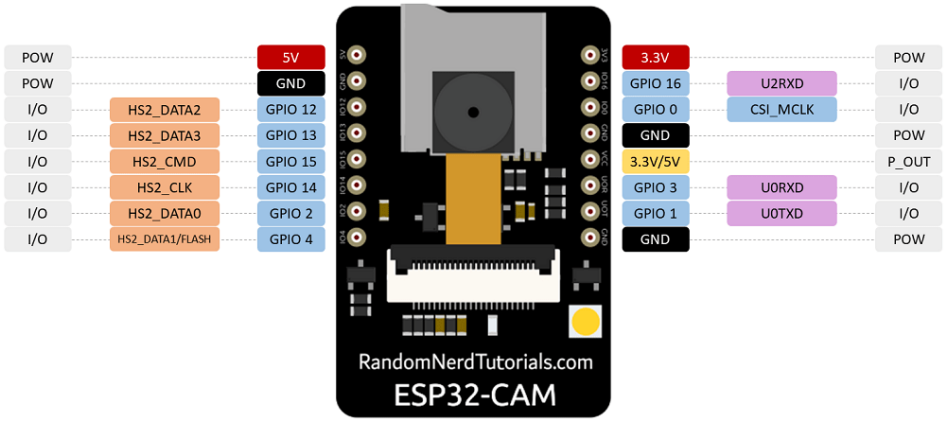

Figure 13: ESP32 Camera

2.5.5 Figma

Figma stands as a web-based design and prototyping tool, gaining popularity for its collaborative features and real-time editing capabilities. The user interface may be simultaneously shaped by design teams, guaranteeing immediate input and doing away with version control issues. So stakeholders are effortlessly included in the creation process through this live collaboration, which goes beyond designers.

Interactive prototypes from Figma improve the design review procedure. Enabling stakeholders to experience the user journey, these prototypes act as dynamic previews. User experiences are improved by this interaction, which also lowers the possibility of misunderstandings during development and results in more polished designs.

Figma takes a comprehensive approach to the whole development lifecycle, not just design that developers could instantly produce code snippets, export assets, and evaluate designs as a result of its design-to-code features. The smoother the prototype-to-product transition is, the more closely the design and development teams collaborate, thanks to this well-organized workflow.

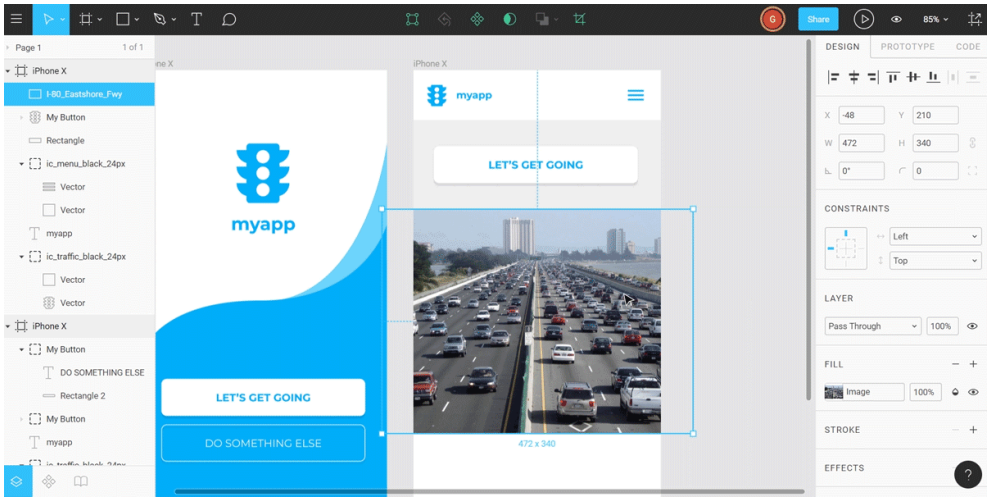

Figure 14: Figma Interface

In essence, Figma's intuitive interface enables designers to craft intricate layouts, define interactions, and establish a visual hierarchy for elements. The design-to-code process ensures that ideas will be implemented in a more fluid way, which paves the way for including the created user interface (UI) into the final application.

2.5.6 Tkinter

Tkinter's prominence in Python GUI development stems from its fluid integration with the language, which guarantees that developers may design GUI applications without the need for extra dependencies because of being a part of the Python standard library. Due to its ability to streamline deployment and distribution, Tkinter has become the preferred option for Python developers.

Tkinter's event-driven paradigm is a powerful example of simplicity. User interactions, such as button clicks or key presses, trigger specific functions. The responsive apps that result from this design paradigm offer a dynamic as well as captivating user experience by smoothly converting user actions into the programmed answers.

Tkinter's strength lies in its alignment with Pythonic principles. The intuitive syntax conveniently combines with Python programming, allowing developers to build graphical interfaces with the same ease as writing Python scripts. The widget set provided by Tkinter, which offers a wide range of elements to create aesthetically pleasing and useful interfaces, is similarly straightforward.

Figure 15: Tkinter

In the context of this project, Tkinter serves as the backbone for implementing the user interface of the autonomous Mecanum robot control system. Its integration with Figma-designed templates ensures a cohesive and user-friendly experience for monitoring and controlling the robot.

3 Implementation and Result

3.1 Setting Adafruit Server and Accessing CamESP32 IP Address

Feeds and Dashboard: Adafruit Server

Initializing some variables in **Feeds**: ai, image, tile variables which mean AI Image Recognition text, Image is pushed to Server and Accuracy of the Image Type respectively.

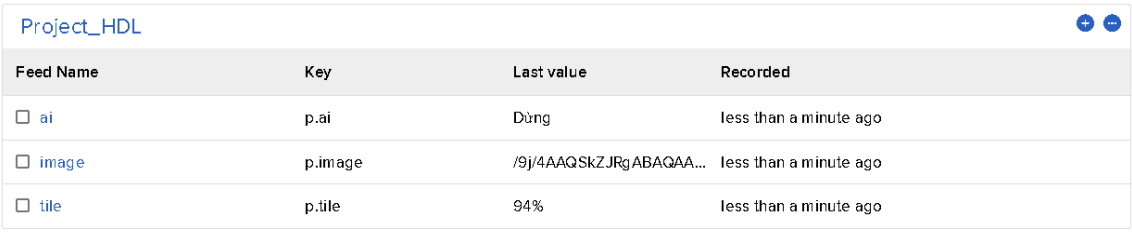

Figure 16: The Variables in Feeds

After that, generate the DASHBOARD, and edit Layout with some blocks such as **Live Image, AI Detection, Accuracy, History** blocks.

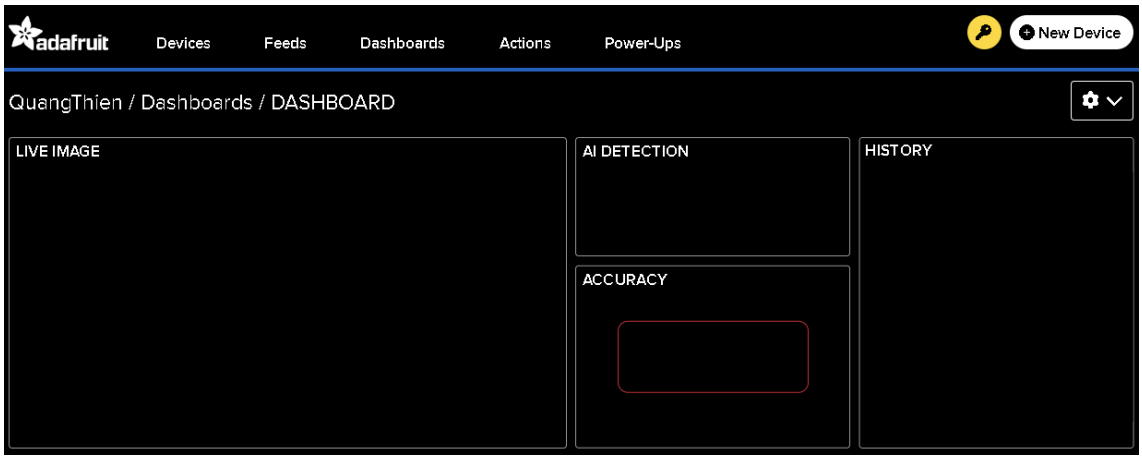

Figure 17: The Dashboard without Data

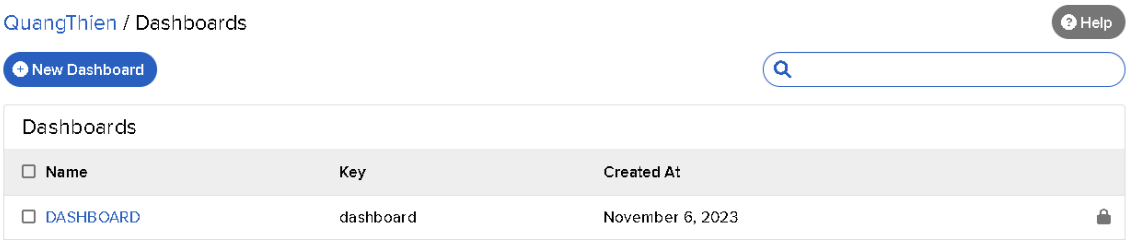

Figure 18: The Dashboard in Adafruit Server

Access CamESP32 IP Address: Arduino IDE

In the first time connect with CamEsp32, using **ESP32cam_setup.INO** file to upload and activate it, and saving **CameraWebServer.INO** file to connect the camera IP address for more rapid access next time. Moreover, setting true WIFI_ID and WIFI_PASSWORD is crucial for a successful Camera upload.

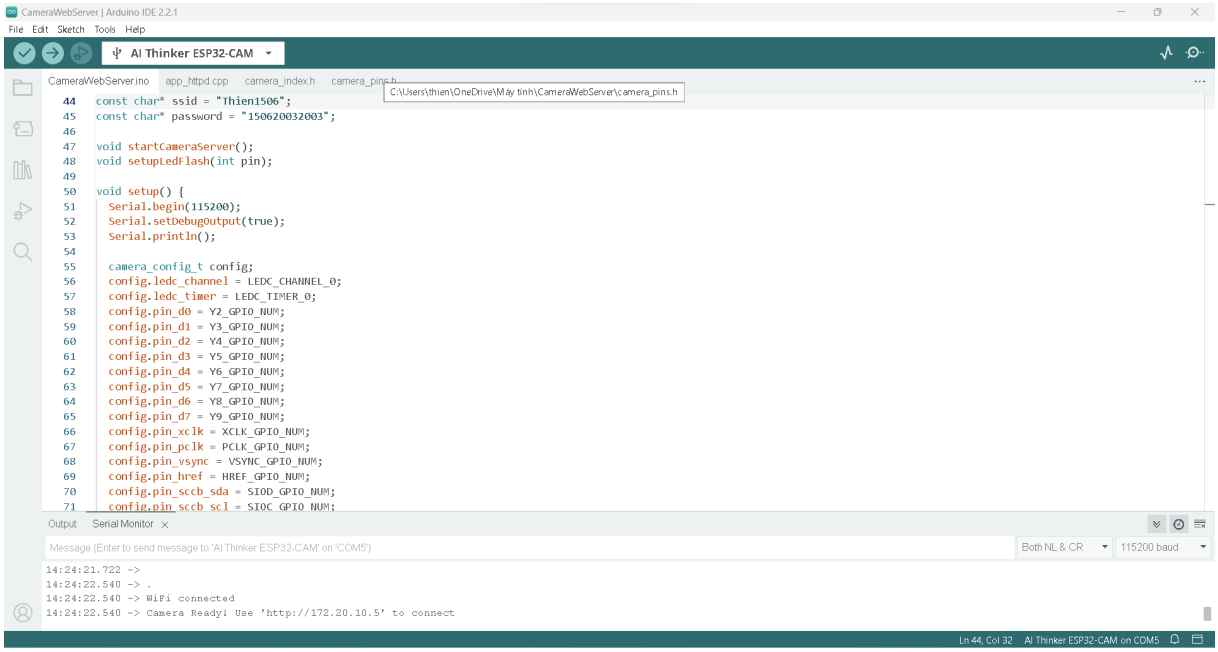

Figure 19: Getting IP Address of CamEsp32

3.2 Training AI model to Identify Traffic Signs

Owing to having the CamESP32 IP Address, we can use the code below to capture images and save them into the Computer to push data to the Teachable Machine model for training AI.

```
import requests
import time

img_url = 'Link IP CamEsp32 + /capture'
counter = 0
while True:
    print("Capturing...", counter + 1)
    counter = counter + 1
    response = requests.get(img_url)
    if response.status_code:
        fp = open('filepath: store images to train' + str(counter) + '.
   png', 'wb')
        fp.write(response.content)
        fp.close()
    time.sleep(1)
```

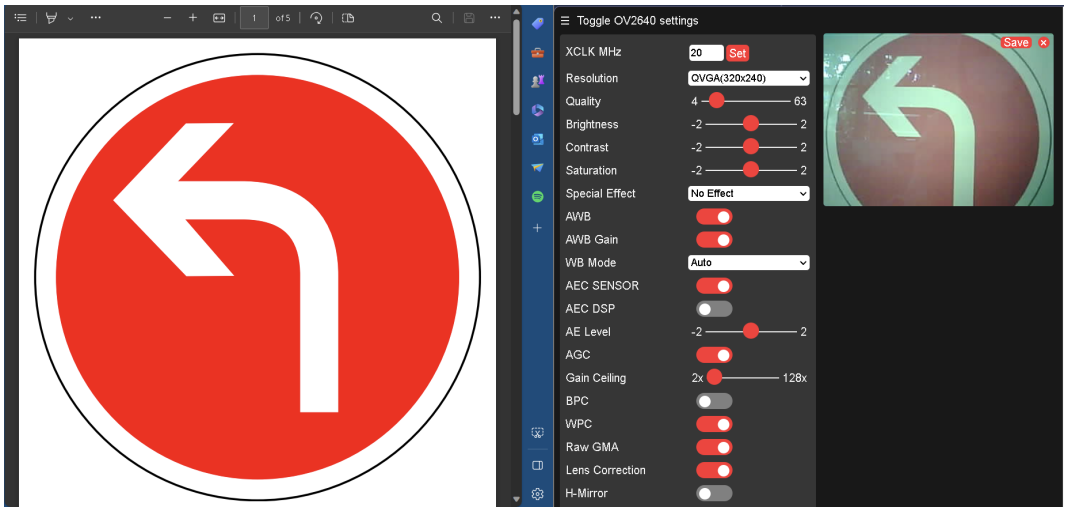

Figure 20: Accessing Camera IP Address for Checking Image before Capture

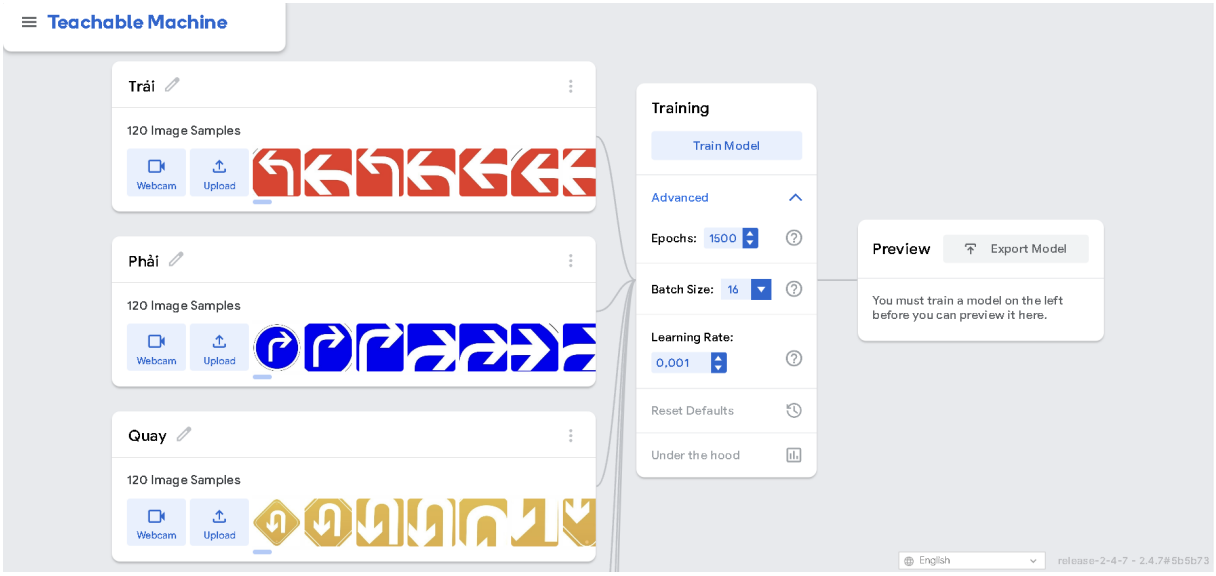

Figure 21: Using Teachable Machine

Having 5 Traffic Sign Classes: **"Trái", "Phải", "Thẳng", "Dừng", "Quay", "Không Biển Báo"**. Each class has more than 100 samples. In particular, class "Thẳng" requires up to 400 samples due to the fact that it is quite similar to class "Phải". **Furthermore**, setting Epochs: 1500, Batch Size: 16, Learning Rate: 0.001 for more accuracy when detecting.

Important Notice

When training a convolutional neural network (CNN) using more epochs in Python with TensorFlow and Keras, the test accuracy may fall due to overfitting. Overfitting occurs when the model learns the training data too well, so it performs poorly on new, unseen data. As the number of epochs increases, the model may memorize the training data instead of learning general patterns. This can lead to reduced performance on the test data.

Moreover, monitoring the validation accuracy during training can help determine the optimal number of epochs to prevent overfitting. Finding a reasonable epoch value which is suitable for the amount of data want to train greatly affects the accuracy of the model.

3.3 Identifying and Transferring Data to Adafruit Server in PyCharm

Having the **keras_model.h5** and **labels.txt** files after training in Teachable Machine: Traffic Sign Detection with 6 classes, each class has at least 100 samples. Creating **AI_Detection_Traffic_Signs.py** and **main.py** files to connect CamESP32 with Adafruit Server:

```
# AI_Detection_Traffic_Signs File
from keras.models import load_model  # TensorFlow is required for Keras to work
import cv2  # Install opencv-python
import numpy as np
import base64
import requests

# Disable scientific notation for clarity
np.set_printoptions(suppress=True)

# Load the model
model = load_model("filepath: keras_model.h5", compile=False)

# Load the labels
class_names = open("filepath: labels.txt", "r", encoding="utf-8").readlines()

def image_detector():

    camera_url = 'Link IP CamEsp32 + /capture'
    response = requests.get(camera_url)
    image = cv2.imdecode(np.frombuffer(response.content, dtype=np.uint8), -1)

    # Convert the image to base64
    res, frame = cv2.imencode('.jpg', image)
    image_data = base64.b64encode(frame).decode('utf-8')

    if len(image_data) > 102400:
        print("Image is too big!")
    else:
        print("Publish image:")
    print(len(image_data))

    # Resize the raw image into (224-height,224-width) pixels
    image = cv2.resize(image, (224, 224), interpolation=cv2.INTER_AREA)

    # Make the image a numpy array and reshape it to the models input shape.
    image = np.asarray(image, dtype=np.float32).reshape(1, 224, 224, 3)

    # Normalize the image array
```

```
    image = (image / 127.5) - 1

    # Predicts the model
    prediction = model.predict(image)
    index = np.argmax(prediction)
    class_name = class_names[index]
    confidence_score = prediction[0][index]

    # Print prediction and confidence score
    return int(class_name[0]), class_name[2:], image_data, str(np.round(
   confidence_score * 100))[:-2]
```

Automatically checking the **image_detector** function once per second. For the robot to perform functions more accurately, the Traffic Sign data received by the server must have a high accuracy rate to limit incorrect function performance. (The accuracy of the Image Detector must be equal to or more than 75% in this project)

```
# main.py File
print("AIoT Project")
import time
import sys
from Adafruit_IO import MQTTClient
from AI_Decision import *
from AI_Detection_Traffic_Signs import *

AIO_USERNAME = "QuangThien"
AIO_KEY = "Key ID"

def connected(client):
    print("Ket noi thanh cong ...")

def subscribe(client , userdata , mid , granted_qos):
    print("Subscribe thanh cong ...")

def disconnected(client):
    print("Ngat ket noi ...")
    sys.exit (1)

def message(client , feed_id , payload):
    print("Nhan du lieu: " + payload + " , feed id: " + feed_id)

client = MQTTClient(AIO_USERNAME , AIO_KEY)
client.on_connect = connected
client.on_disconnect = disconnected
client.on_message = message
client.on_subscribe = subscribe
client.connect()
client.loop_background()
```

```
counter_ai = 1
ai_res = ""
prev_res = ""
accuracy = "0"
prev_accuracy = "0"

while True:
    counter_ai = counter_ai - 1
    if counter_ai <= 0:
        counter_ai = 1
        prev_accuracy = accuracy
        if int(prev_accuracy) >= 75:
            prev_res = ai_res
        new_state, ai_res, image, accuracy = image_detector()
        print("AI Output: ", ai_res.strip() + " " + accuracy + "%")
        if prev_res != ai_res and int(accuracy) >= 75:
            # Publish data to Adafruit Server
            client.publish("p.ai", ai_res)
            client.publish("p.image", image)
            client.publish("p.tile", accuracy + "%")
            client.publish("ai", new_state)

    time.sleep(1)
    pass
```

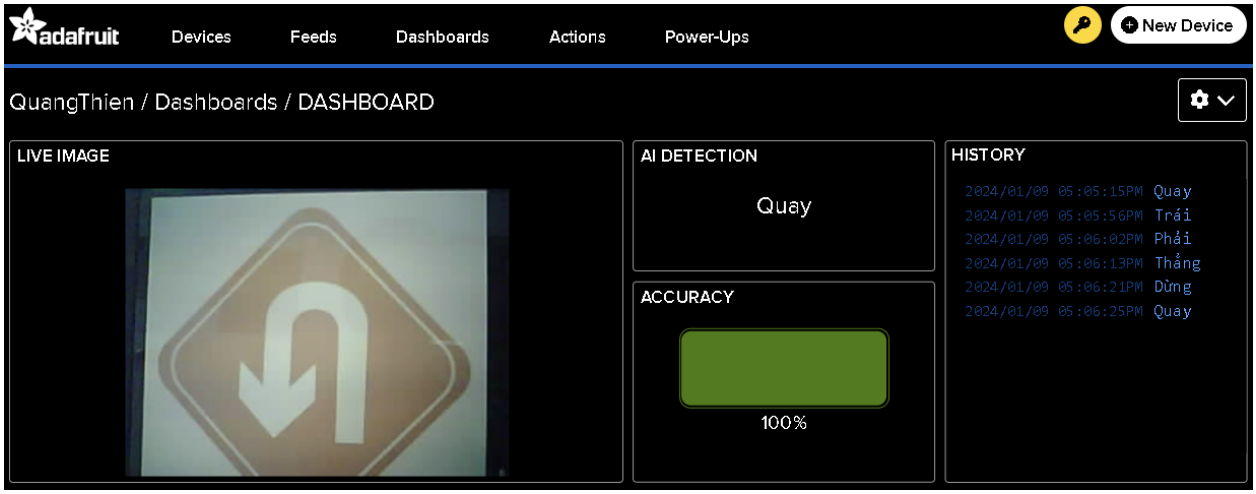

Figure 22: Output after publishing data

3.4 Coding detect line using PID Algorithm

Having the values of the sensor and identifying the position of the Robot in the map line:

Sensor Values	Robot Positions
1 1 1 1 0 1 1 0	On Line
1 0 0 0 1 1 0 0 / 0 1 00	Deflect Right
0 0 0 1 0 0 1 1 / 0 0 1 0	Deflect Left
0 0 0 0	Out Line

Figure 23: Sensor Values and Corresponding Positions

In simpler terms, there are at most two sensors to detect the line deflect right or level, then we will apply six levels of line deviation: if number 1 is the value while identifying a line, the more number 1 the more left or right deflection except the case in the straight line.

Utilizing the PID algorithm to supply the microcontroller with a PWM control value will be a suitable technique to reduce the delay in robot movement. Set the sensor values to the corresponding values:

Sensor Values	Corresponding Values	Robot Positions
1 0 0 0 1 1 0 0 / 0 1 0 0	2 1	Deflect Right
1 1 1 1 0 1 1 0	0	On Line
0 0 1 1 / 0 0 1 0 0 0 0 1	-1 -2	Deflect Left
0 0 0 0	-3	Out Line

Figure 24: Sensor Values and Corresponding Values

Also based on the frame code below to build code in Ohstem with basic values of **Kp**, **Ki**, and **Kd** (0.6, 0.2, 0.2 respectively) to implement this project:

```
def calculate_pid()
{
    P = error;
    I = I + error;
    D = error - previous_error;

    # Initializing Simple Constant Values: Kp = 0.6, Ki = 0.2, Kd = 0.2
    PID_value = (Kp * P) + (Ki * I) + (Kd * D);
```

```
    previous_error = error;
}
```

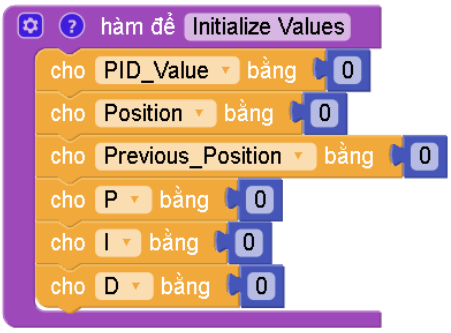

Figure 25: Initializing Variables in Ohstem

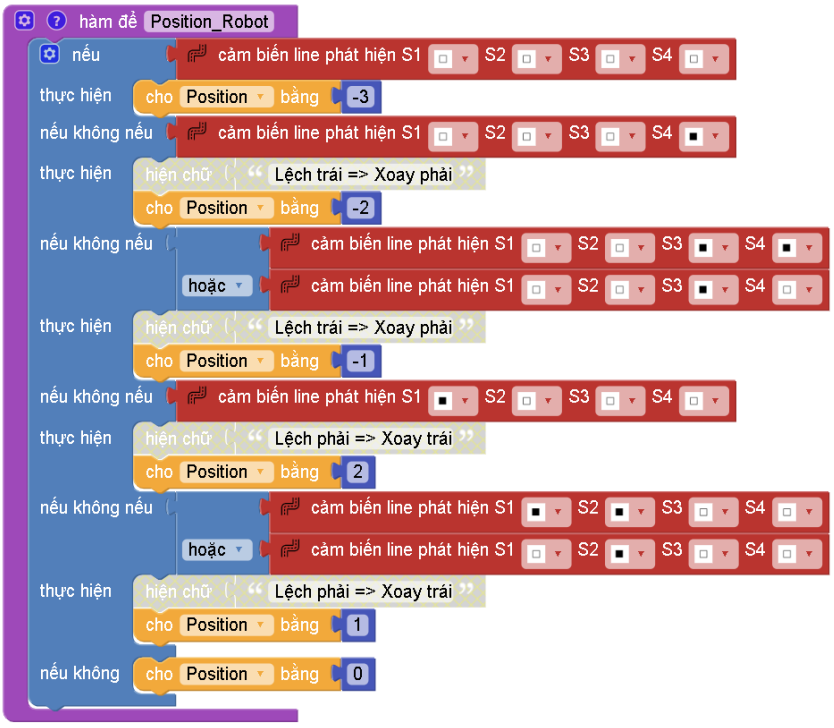

Figure 26: Changing Sensor Values to Corresponding Values

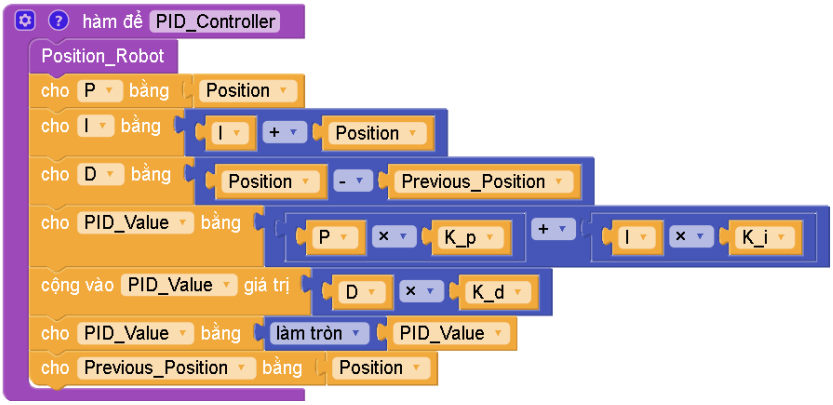

Figure 27: PID function

3.5 Connecting Data from Adafruit Server to Ohstem

First of all, connecting wifi plays a crucial role in accessing wireless successfully. Make sure that the wifi connecting in Ohstem is the same as the wifi used when getting the CamESP32 IP Address.

Figure 28: Connecting Wifi in Ohstem

3.6 Generating Line Functions in Ohstem

Line Forward function: Applying PID Control for precision line following.

In the context of controlling the Robot Mecanum's movement along a line, the application of a PID (Proportional-Integral-Derivative) controller is crucial for minimizing errors and ensuring optimal and accurate movement. To be more specific, the function integrates PID control to precisely adjust the robot's movement based on real-time feedback from line sensors, encompassing the following key elements:

- **Integration of PID Controller:** The PID value gets calculated using the **PID_Controller** function and serves as a movement correction factor for the robot. It incorporates proportional error (P), integral error (I), and derivative error (D) components, collectively minimizing deviations from the desired line path.

- **Position Sensing:** The accurate determination of the robot's position concerning the line is achieved through the **Position_Robot** function. Line sensors embedded in the robot assess the surroundings, categorizing the robot's position as Left, Right, Center, or Off the line. This real-time sensing forms the foundation for precise line-following behavior.

- **Motor Adjustment based on the value of Position:** The Mecanum wheel motor speeds are dynamically adjusted based on the computed PID value and the robot's position. This meticulous modification ensures the robot stays precisely on course, enhancing its ability to handle diverse line-following scenarios.

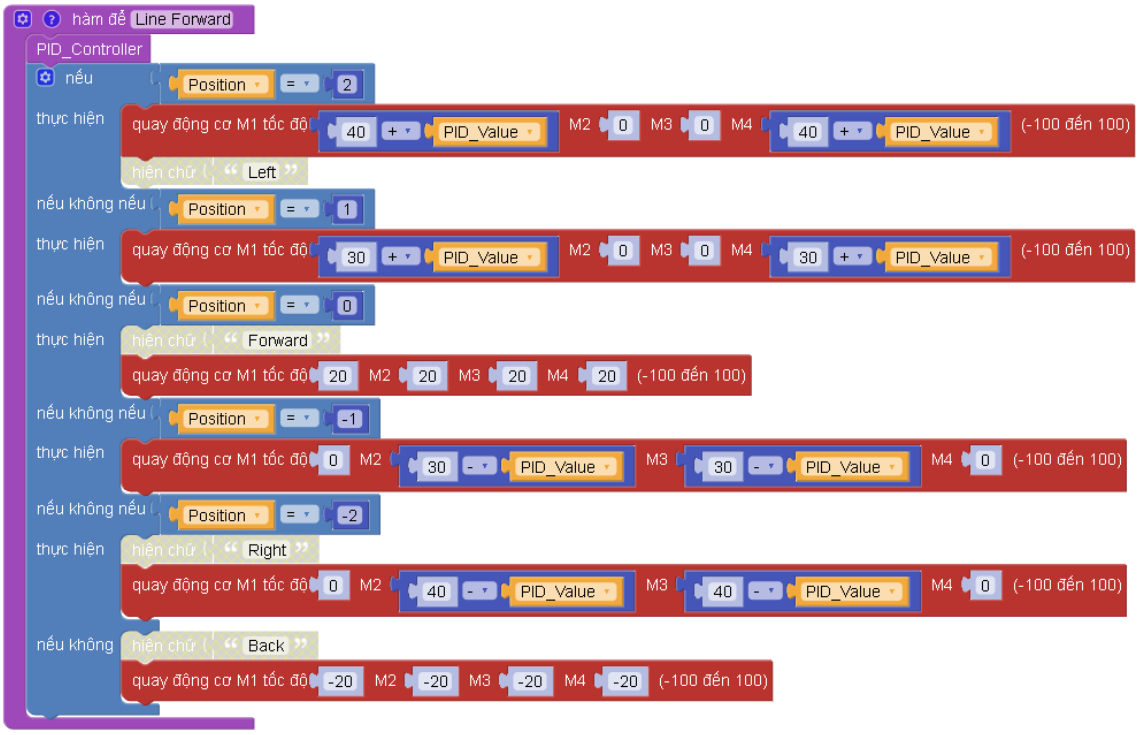

Figure 29: Line Forward Function

Therefore, the **Line_Forward** function is an autonomously moving robot that adeptly follows the designated line. The meticulous integration of PID control, real-time position sensing, and dynamic motor adjustments collectively results in robust and precise line-following behavior.

Forward function: Autonomous Line Following with Mecanum Wheels

The mentioned **Forward** function serves as the linchpin of the robot's autonomous navigation system, orchestrating its movement along a predetermined path with the precision required for effective line following. Primary concepts of this function include:

- Within the function, the robot continuously reads data from strategically positioned line sensors designed to detect the path on the ground. The conditions embedded within the loops ensure the robot's adept response to diverse line configurations, distinguishing between being directly on the line, slightly deviated, or at a junction point.

- When identifying the aforementioned conditions, the robot adjusts its motor speeds using the PID controller. This controller fine-tunes the movement, allowing the robot to smoothly navigate the path. Additionally, there are mechanisms to halt the robot's motion when required, ensuring it responds effectively to varying environmental cues.

- The integration of conditions within the loops enhances the adaptability of the robot, enabling it to handle intricate line-following scenarios. This detailed decision-making process is crucial for the robot to interpret and respond to changes in the detected line configuration, showcasing the effectiveness of the implemented algorithm.

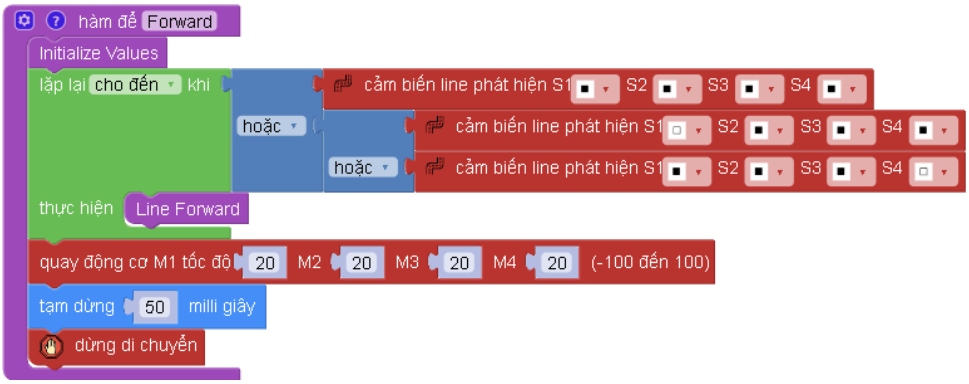

Figure 30: Forward Function

In essence, the **Forward** function is a key component of the robot's autonomous navigation system, demonstrating a sophisticated interplay of sensor readings, PID control, and motor adjustments to achieve precise line-following capabilities.

Line Left and Right function: Accurate Line Following Correction

Designed as a strategic complement to the primary *Line_Forward()* function, the **Line Left and Right Function** function assists Robot Mecanum in maintaining the robot's alignment with the line. Its primary objective is to fine-tune the robot's position by deploying specific **"rotate right"** and **"rotate left"** commands, intentionally activating sensors 2 and 3. This intentional activation significantly increases the probability of these sensors being triggered, enhancing the robot's ability to detect and precisely follow the line, particularly in scenarios where deviations may occur.

The fundamental purpose is to ensure that the robot consistently stays centered on the line during its traversal, which is accomplished by carefully orchestrating the rotation commands to adjust the robot's orientation based on the readings from sensors 2 and 3. The integration of these commands within the function's structure facilitates a nuanced approach to line following, allowing for adaptability to various environmental challenges.

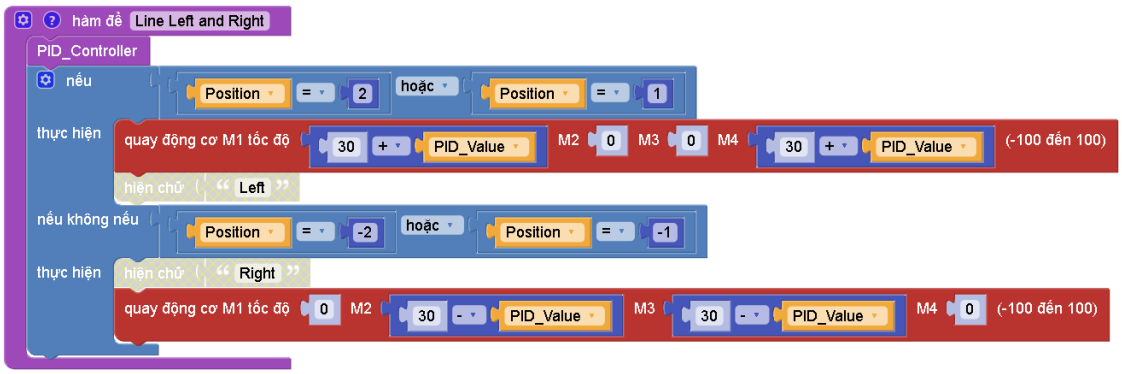

Figure 31: Line Left and Right Function

As a miniature function nested within the broader line-following strategy, the **Line Left and Right** function contributes to the overall efficiency and accuracy of the robot's movement. Its objective in manipulating the robot's position through rotation commands showcases the complexities involved in fine-tuning the line-following behavior. This function becomes particularly valuable in scenarios where precise alignment is essential, such as navigating through complex paths where deviations from the line may occur.

Left function: Precision control in left turns.

The function orchestrates left turns, employing a sophisticated interplay of values initialization, interaction with line sensors in loops, and sequential adjustment of motor speeds. The following key concepts delineate the intricacies of this function:

- Upon initiation, essential variables are set, establishing the foundational framework for subsequent motor control. Specific motor commands are executed to initiate a left turn, laying the groundwork for the turning process.
- After initialization, a timed sleep interval is introduced, ensuring a smooth and meticulous left turn. This interval facilitates controlled rotation by adjusting the speeds of designated motors, promoting gradual and stable movement.
- The distinctive feature of the **Left** function is its dynamic interaction with line sensors within loops. These loops continually monitor the robot's position about the line, providing real-time feedback for adaptive adjustments. The primary loop condition scrutinizes specific line sensor configurations, determining deviations from the desired path. Emphasis is placed on readings from line sensors, particularly sensors 1, 2, and 3, to interpret spatial information effectively.
- Within the loop, the function dynamically engages with line sensors to detect changes in the robot's position. Then, it meticulously crafted loop conditions check for the absence of specific line sensor readings, indicating departures from the ideal path. After that, sequential adjustments to motor speeds occur as the robot progresses through the turn, aligning with the curvature of the line.
- Continuous evaluation of inputs from line sensors enables the function to fine-tune motor speeds. This ensures the robot's alignment with the desired path, preventing overshooting or undershooting during the turn.

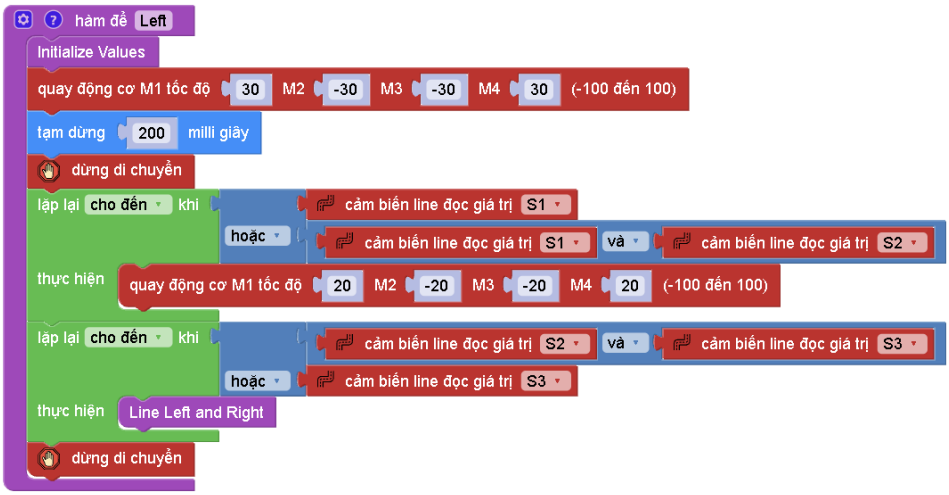

Figure 32: Left Function

Therefore, the function's effectiveness in executing left turns is intricately linked to the meticulous conditions within its loops. The specific concentrate on line sensor readings, namely sensors 1, 2, and 3, demonstrate a purposeful approach to interpreting spatial information, which empowers the robot with real-time adaptability, fostering controlled and accurate left turns through precise motor adjustments based on the nuances of line sensor data.

Right function: Precision control in right turns.

The **Right** function, similar to its counterpart *Left* function, is integral to the robot's line-following strategy. It initiates and executes right turns with a structured approach, leveraging line sensor data and motor adjustments. Noteworthy points of distinction from the *Left* function include:

- Essential variables are initially set; however, specific motor movements command a smooth initiation of the right turn, akin to the *Left* function.
- In the loop conditions where sensor readings are evaluated, the choice to first check sensors 3 and 4 before assessing sensors 2 and 3 is a deliberate strategy. This sequence allows the function to initially focus on the right side of the robot. By prioritizing sensors 3 and 4, the function checks for deviations or alignments to the right of the desired line path. This sequential evaluation helps the robot make nuanced adjustments, particularly when approaching curves or turns to the right.
- Following this initial assessment, the subsequent loop, which checks sensors 2 and 3, provides a complementary analysis of the left side. This sequential process ensures that the function considers both sides of the robot independently, allowing for differential responses based on the specific conditions detected. The effectiveness lies in its ability to adapt to the nuances of the line-following scenario, enhancing the robot's precision and responsiveness in navigating complex paths.

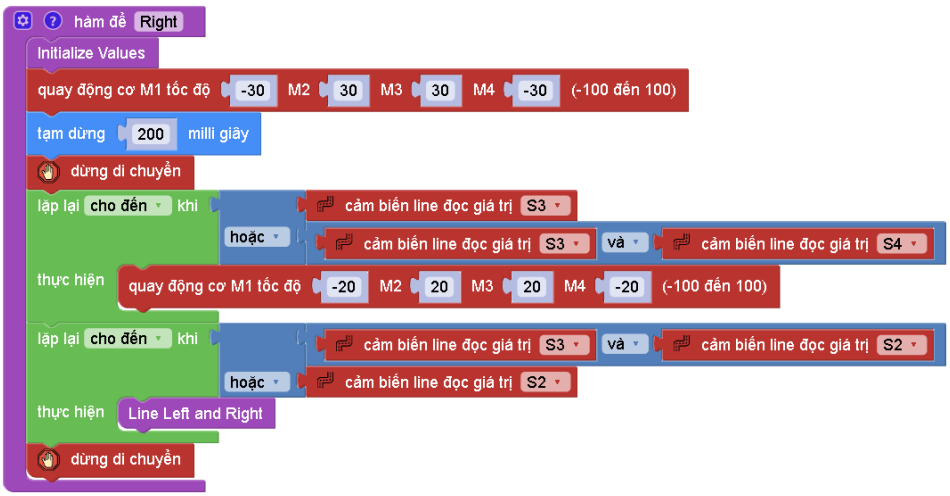

Figure 33: Right Function

To conclude, the **Right** function aligns with the structure of the *Left* function, with key distinctions lying in the intentional activation of specific sensors during right turns, optimizing line-following accuracy.

Stop function: Halting autonomous motion

The **Stop** function is invoked to bring the robot to a complete stop, effectively ceasing its movement along a predefined path. Important features of the **Stop** function consist of:

- It is set to all four wheels' speeds to zero in order to halt the movement, which is the main objective of the function. By bringing each wheel to a standstill, the function ensures that the robot comes to a complete stop.
- As a user-friendly feature, the function incorporates a visual indicator, which is to show a positive image **YES**, serving as a visual confirmation that the robot has successfully come to a stop. This visual cue enhances the user's understanding of the robot's state on Yolobit.

Figure 34: Stop function in Ohstem

3.7 Generating AI Detection Command

Operation Mechanism: Base on the Button Operation in Micro-Controller & Micro-Processor Subject, generating the main 3 values:

- **KeyReg0**: Current State Image.
- **KeyReg1**: Image State from the previous iteration.
- **KeyReg2**: Image State from the two previous iterations.

The initial values of those are **NO_SIGN** which means the robot will be in the stop state. Then, in order to decide on what next direction of processing, compare the three image states. When a new Traffic Sign is updated from the server, the robot will retrieve data from the server, leading to the fact that the **2 Previous State Images** and the **Current State Image** differ from each other, then immediately set the Flag, and perform the function of that Traffic Sign. If the received Traffic Sign remains, every 5 seconds the robot will perform the function of that Traffic Sign again. (1 piece of music is played if it is an old Traffic Sign and should be implemented again. If there is a new Traffic Sign, 2 pieces of music will be played consecutively)

Define AI Values function: Generate all necessary values to serve the Operation Mechanism above.

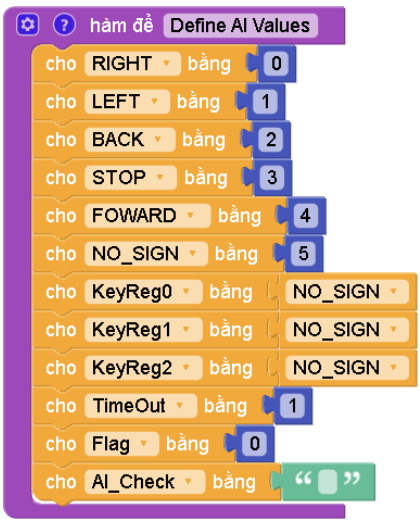

Figure 35: Define AI Values function in Ohstem

Check AI Detection function: For each value of **AI_Check** value received from the Adafruit server, the **KeyReg0** will be set the corresponding value.

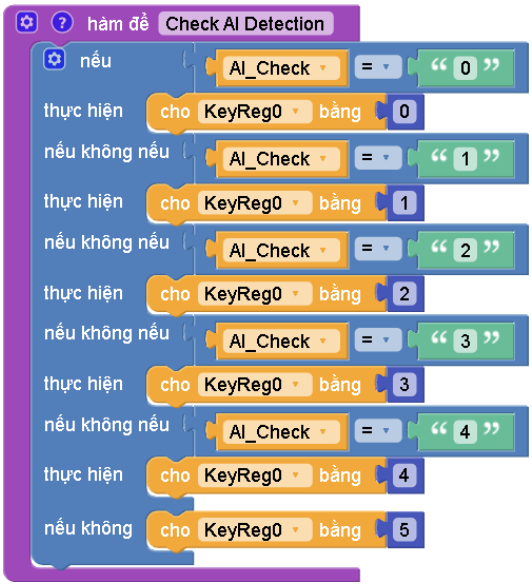

Figure 36: Check AI Detection function in Ohstem

AI Detection function: When receiving an update from the server, the robot will promptly acquire the necessary data. So the **Current State Image** is different from the **2 Previous State Images**. Then set the Flag, and execute the traffic sign's function. The robot will repeat the function of the received traffic sign every five seconds if the Traffic Sign is still unchanged.

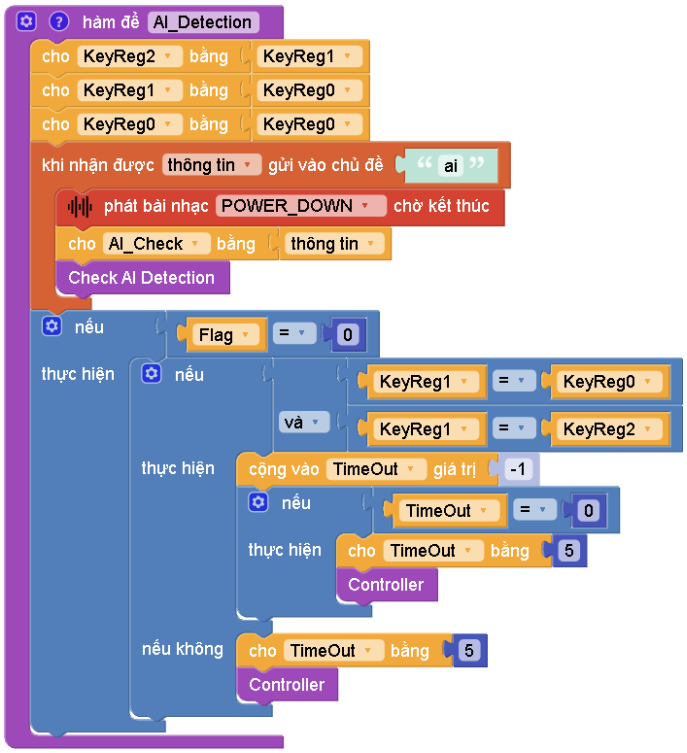

Figure 37: AI Detection function in Ohstem

Controller function: Implementing the function of the Traffic Sign is detected. Moreover, if it belongs to the 5 types of Traffic Signs used to train the AI model, Yolobit will display YES. Else it is NO_SIGN, the Yolobit will demonstrate NO (to distinguish it from the Stop Sign) and do nothing.

Figure 38: Controller function in Ohstem

3.8 Integrating CamESP32 into Robot Mecanum and Practical Operation

The last stage is to integrate the camera into the robot and use AI models to recognize the Traffic Signs. In general, after successfully integrating the devices and loading the code, the **Robot Mecanum** runs fairly well when receiving signals from the server after data is uploaded, processing commands correctly, moving on the line, and not straying from the line.

Figure 39: Integrating All Functions

Besides, we observe that it takes a lengthy time for the robot to receive new data from the server (about 1 second per time), leading to the robot's processing sensitivity not being optimal.

The following link below shows the demonstration result about the process of running and executing Traffic Sign commands of Robot Mecanum using CamESP32 and Teachable Machine model:

- Video Demonstration of AI Intergration and Robot Mecanum

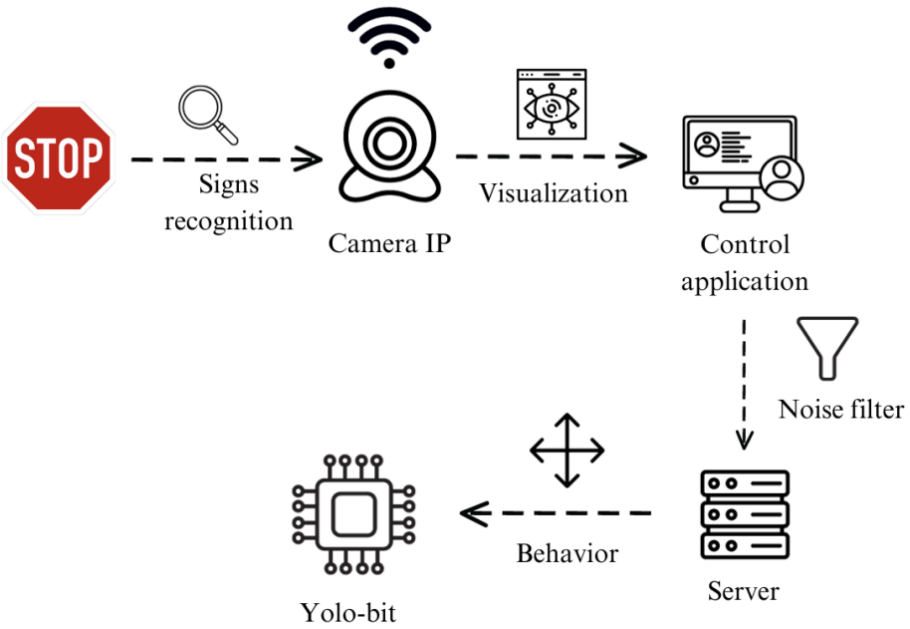

Figure 40: Sign Recognition mode flow

3.9 Designing User Interface (UI)

First of all, using **Figma** to design a basic template.

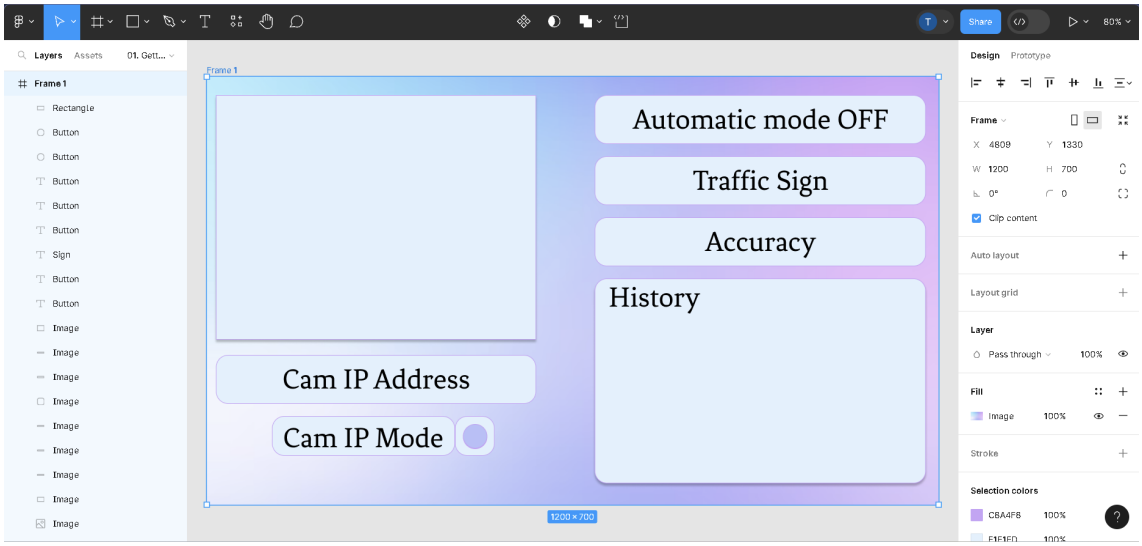

Figure 41: Figma App

After the design is finished, translate it to **PyCharm** and use library **Tkinter** to continue coding UI. Below are some main functions in the UI code.

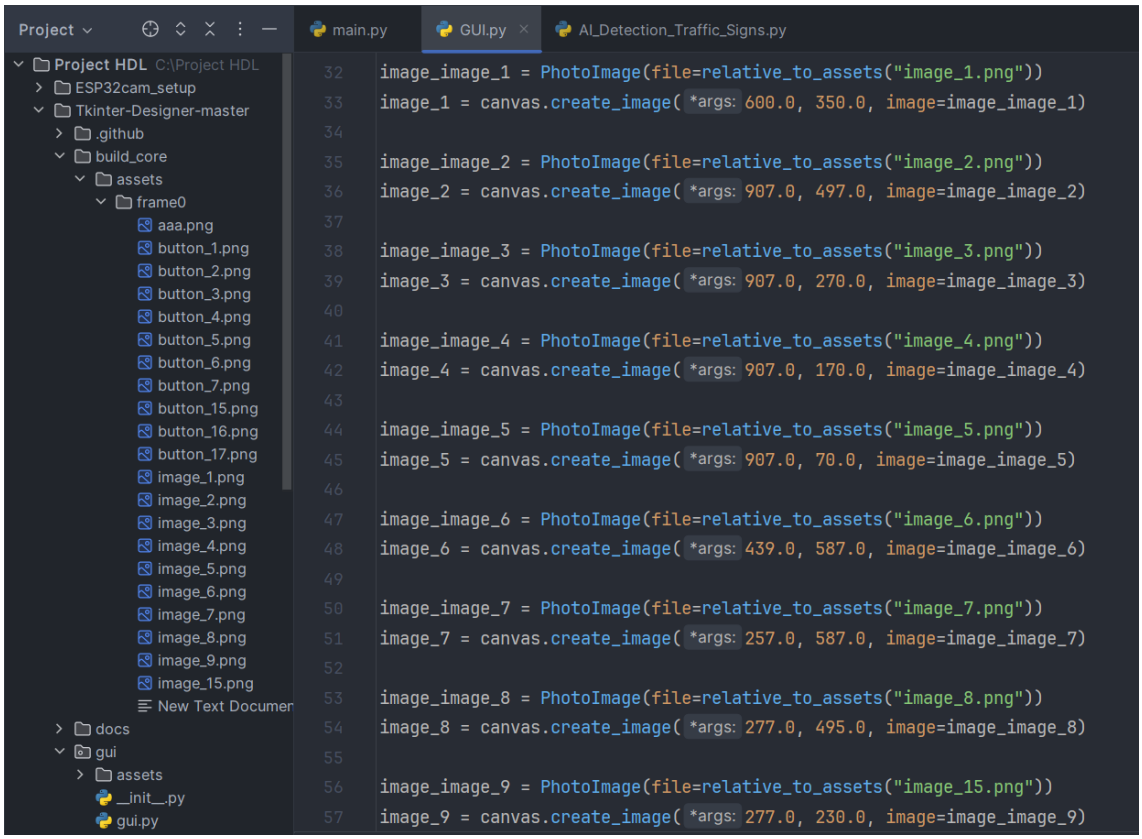

Figure 42: Create frames in Pycharm

Set up background and icon for UI.

```
OUTPUT_PATH = Path(__file__).parent
ASSETS_PATH = OUTPUT_PATH / Path(r"UI Images Path")

def relative_to_assets(path: str) -> Path:
```

```
    return ASSETS_PATH / Path(path)

window = Tk()
window.title("Project HDL - AI Detection UI")
window.iconbitmap('CSE Icon Path')

window.geometry("1200x700")
window.configure(bg="#FFFFFF")

canvas = Canvas(window, bg="#FFFFFF", height=700, width=1200, bd=0,
    highlightthickness=0, relief="ridge")
```

Update time, date, and the type of Traffic Sign. In addition, when a unit of time such as a minute or second unit, is less than 10, the string 0 will be concat before them. (For instance, Date: 15/6/24 Time: 15:6:3 will be Date: 15/06/24 Time: 15:06:03)

```
# History
custom_font1 = Font(family="Times", size=26, weight="normal")
# slant="italic", underline=0, overstrike=0
history1 = ""
history2 = ""
history3 = ""
history4 = ""
history5 = ""
history6 = ""

def fix_time(current_time):
    day = str(current_time.day)
    month = str(current_time.month)
    hour = str(current_time.hour)
    minute = str(current_time.minute)
    second = str(current_time.second)
    if int(day) < 10:
        day = "0" + day
    if int(month) < 10:
        month = "0" + month
    if int(hour) < 10:
        hour = "0" + hour
    if int(minute) < 10:
        minute = "0" + minute
    if int(second) < 10:
        second = "0" + second
    return day, month, hour, minute, second

def change_history(new_value):
    global history1, history2, history3, history4, history5, history6
    current_time = datetime.now()
    day, month, hour, minute, second = fix_time(current_time)
```

```

    if (history1 != "" and history2 != "" and history3 != ""
        and history4 != "" and history5 != "" and history6 != ""):
        history1 = history2
        history2 = history3
        history3 = history4
        history4 = history5
        history5 = history6
        history6 = day + "/" + month + "/" + "24" + " " + hour + ":" +
    minute + ":" + second + " " + new_value
    elif history1 == "":
        history1 = day + "/" + month + "/" + "24" + " " + hour + ":" +
    minute + ":" + second + " " + new_value
    elif history1 != "" and history2 == "":
        history2 = day + "/" + month + "/" + "24" + " " + hour + ":" +
    minute + ":" + second + " " + new_value
    elif history1 != "" and history2 != "" and history3 == "":
        history3 = day + "/" + month + "/" + "24" + " " + hour + ":" +
    minute + ":" + second + " " + new_value
    elif history1 != "" and history2 != "" and history3 != "" and
    history4 == "":
        history4 = day + "/" + month + "/" + "24" + " " + hour + ":" +
    minute + ":" + second + " " + new_value
    elif history1 != "" and history2 != "" and history3 != "" and
    history4 != "" and history5 == "":
        history5 = day + "/" + month + "/" + "24" + " " + hour + ":" +
    minute + ":" + second + " " + new_value
    elif history1 != "" and history2 != "" and history3 != "" and
    history4 != "" and history5 != "" and history6 == "":
        history6 = day + "/" + month + "/" + "24" + " " + hour + ":" +
    minute + ":" + second + " " + new_value

    history.config(text=history1+"\n"+history2+"\n"+history3+"\n"+
    history4+"\n"+history5+"\n"+history6,)

history = Label(font=custom_font1, text=history1+"\n"+history2+"\n"+
    history3+"\n"+history4+"\n"+history5+"\n"+history6,
                borderwidth=0, anchor="w", justify=LEFT,
    highlightthickness=0, bg="#E1F1FD")

history.place(x=652.0, y=368.0, width=510.0, height=290.0)
```

Update data to Adafruit server and User Interface.

```
Flag_UI = 0
def Generate_UI():
    global counter_ai, ai_res, prev_res, accuracy, prev_accuracy
    if Flag_UI == 1:
```

```
        counter_ai = counter_ai - 1
        if counter_ai <= 0:
            counter_ai = 1
            prev_accuracy = accuracy
            if int(prev_accuracy) >= 75:
                prev_res = ai_res
            new_state, ai_res, image, accuracy = image_detector()
            print("AI Output: ", ai_res.strip() + " " + accuracy + "%")
            if prev_res != ai_res and int(accuracy) >= 75:
                # Publish data to Adafruit Server
                client.publish("p.ai", ai_res)
                client.publish("p.image", image)
                client.publish("p.tile", accuracy + "%")
                client.publish("ai", new_state)
                change_history(ai_res.strip())
                change_accuracy(accuracy)
                change_traffic_sign(ai_res.strip())

            window.after(1000, Generate_UI)
```

Generate a button to control Automatic Mode State.

```
# Automatic_On_Off
Flag = 0
def Automatic_On_Off():
    global button_image_4, button_4, Flag, Flag_UI, button_image_6, button_6, Flag1, button_image_7, button_7
    if Flag == 0:
        button_image_4 = PhotoImage(file=relative_to_assets("button_15.png"))
        button_image_6 = PhotoImage(file=relative_to_assets("button_17.png"))
        Flag = 1
        Flag_UI = 1
        Generate_UI()
    elif Flag == 1:
        button_image_4 = PhotoImage(file=relative_to_assets("button_4.png"))
        button_image_6 = PhotoImage(file=relative_to_assets("button_6.png"))
        Flag = 0
        Flag_UI = 0
        if Flag1 == 1:
            button_image_7 = PhotoImage(file=relative_to_assets("button_7.png"))
            Flag1 = 0
            toggle_camera()
            button_7.config(image=button_image_7)
```

```
21
22      button_4.config(image=button_image_4)
23      button_6.config(image=button_image_6)
24
25
26  # Cam_IP
27  button_image_6 = PhotoImage(file=relative_to_assets("button_6.png"))
28  button_6 = Button(image=button_image_6, borderwidth=0,
        highlightthickness=0, bg="#E1F1FD", command=lambda: Automatic_On_Off
        (), relief="flat")
29  button_6.place(x=124.0, y=464.0, width=308.0, height=62.0)
```

Implement a function to resize images of the camera to match it with the frame and also a button to switch ON, switch OFF the camera. (Update images continuously to make it look like a video)

```
1  # Frame Cam
2  camera_active = True
3  def toggle_camera():
4      global camera_active
5      camera_active = not camera_active
6
7  def update_frame():
8      if camera_active:
9          frame = get_data()
10         label = Label(width=519, height=394)
11         label.place(x=16.0, y=31.0)
12
13         frame = cv2.resize(frame, (519, 394)) # resize the frame to 200
       x200
14         img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
15         img = Image.fromarray(img)
16         imgtk = ImageTk.PhotoImage(image=img)
17         label.imgtk = imgtk
18         label.configure(image=imgtk)
19         window.after(500, update_frame)
20
21
22 # Cam_On_Off
23 Flag1 = 0
24 def Cam_On_Off():
25     global button_image_7, button_7, Flag1, camera_active, Flag_UI
26     if Flag_UI == 1:
27         if Flag1 == 0:
28             button_image_7 = PhotoImage(file=relative_to_assets("
       button_16.png"))
29             Flag1 = 1
30             camera_active = True
31             update_frame()
```

```
        elif Flag1 == 1:
            button_image_7 = PhotoImage(file=relative_to_assets("
    button_7.png"))
            Flag1 = 0
            toggle_camera()
     else:
        button_image_7 = PhotoImage(file=relative_to_assets("button_7.
    png"))
        Flag1 = 0
        camera_active = False

     button_7.config(image=button_image_7)

button_image_7 = PhotoImage(file=relative_to_assets("button_7.png"))
button_7 = Button(image=button_image_7, borderwidth=0,
    highlightthickness=0, bg="#E1F1FD", command=lambda: Cam_On_Off(),
    relief="flat")
button_7.place(x=420.0, y=568.0, width=40.0, height=40.0)
```

Mechanism of UI: The **Cam IP Address** connects with the Automatic Mode State, which means when showing the IP Address it will turn ON the Automatic mode. Otherwise, the Automatic mode is OFF. Furthermore, the **Cam IP Mode** button implements the function of switching ON or OFF the camera. Notice that only when the Automatic mode is ON, the button to control the state ON and OFF of the camera is allowed. Owning to the time the Automatic mode is OFF, the time the camera stops to identify **Traffic Signs**. In **History** data, we update the 6 newest values in order to check the change of the **Traffic Signs**.

The following link below shows the demonstration result about the process of running and operating the User Interface:

- Video Demonstration of User Interface

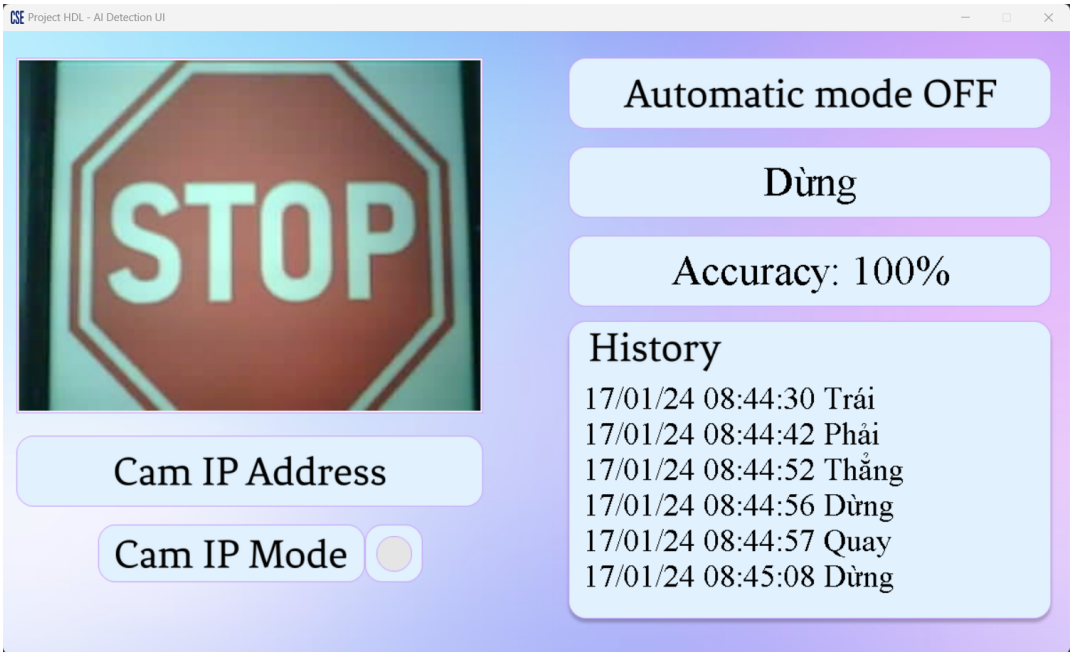

Figure 43: UI detects and sends data to the Dashboard in Adafruit Server

4 Conclusion

On the whole, throughout the development **AI-Integrated Image Detection for Autonomous Mecanum Robot Control** project with our all-out effort, we stand at the juncture of innovation and implementation, reflecting on the journey that unfolded and the accomplishments achieved. The project's foremost achievement lies in the successful integration of artificial intelligence and robotics, forging a symbiotic relationship between image detection and autonomous control. The developed system demonstrates a robust capability to recognize and respond to traffic signs: **Left, Right, Straight, Stop, U-turn** propelling the Mecanum robot into the realm of context-aware and adaptive navigation.

The foundational step involved harnessing the power of Teachable Machine to train a sophisticated AI model, proficient in recognizing an extensive array of traffic signs. This AI prowess is fluidly integrated with the CamESP32, forging a robust channel connecting real-time image capture and data transfer to the Adafruit server. The amalgamation of these technologies laid a concrete foundation for the robot's visual perception capabilities. In parallel, the development of a nuanced line-following algorithm granted the Mecanum robot the ability to navigate predefined paths with adaptability and responsiveness. On the user interface front, a thoughtfully designed platform emerged, offering manual control options and the real-time visualization of captured images, fostering enhanced situational awareness for both users and the autonomous system. The final accomplishment was realized in the integration of AI image detection with autonomous control, empowering the robot to make context-aware decisions based on visual cues. This orchestration of AI, robotics, and user interface design culminated in a project that not only surpassed its individual components but synergistically showcased the potential of integrated technologies in the realm of autonomous robotics.

The project's trajectory was marked not only by triumphs but also by challenges that became stepping stones for valuable lessons. First and foremost, encountering hardware malfunctions on the Mecanum Robot provided us with precious experience and substantial knowledge in troubleshooting and system management from wiring the Mecanum wheels to soldering power circuits. Resolving hardware issues was not just a technical challenge but also an opportunity to enhance skills in assembly and repair. By addressing these malfunctions, we deepened our understanding of working with fundamental components of the robot, establishing a solid foundation for building and maintaining autonomous systems in the future. What is more, iterative fine-tuning of the line-following algorithm proved instrumental in achieving optimal performance, requiring a delicate balance between responsiveness and stability. The integration of AI image detection with real-time robot control brought forth complexities in synchronizing data flow, challenges that were methodically navigated through meticulous system design. As the project unfolded, the paramount lesson emerged in the form of adaptive control strategies, underscoring the necessity for continuous adjustments to accommodate the dynamic variations inherent in diverse line-following scenarios. Simultaneously, the optimization of the user interface revealed itself as an ongoing process, highlighting the critical importance of a clear and intuitive design for fostering effective human-robot interaction. In overcoming these challenges, the project not only evolved technologically but also enriched its fabric with the wisdom gleaned from each hurdle surmounted.

About the future work, the project has established the framework for it, which is intended to make the Robot Mecanum be utilized and applied directly in transforming agriculture. It is plainly evident that farming techniques should be revolutionized by the autonomous Mecanum robot, which would use the AI-driven vision to supplement its well-established line-following habit. Future endeavors include deploying the robot to autonomously navigate crop fields, employing its AI-trained model to assess plant health, monitor water levels, and promptly respond to anomalies such as wilting leaves or potential roadblocks. In addition to this, incorporating dynamic path-planning algorithms will enhance the robot's ability to navigate through unpredictable environments with greater agility. The integration of such

technology promises to bring efficiency, precision, and a new level of automation to agricultural processes, contributing to the sustainable farming practices and addressing the new evolving challenges in the agriculture sector.

To conclude, the project is truly a noteworthy advancement in the amalgamation of robotics and artificial intelligence, offering a model for the forthcoming autonomous systems. With all of its successes and setbacks, this is a starting point for more research and development in the field of intelligent robots. As we look ahead, the project stands as a testament to the boundless possibilities when technology, creativity, and determination converge in the pursuit of groundbreaking solutions in transforming agriculture.

Last but not least, we would like to extend our sincere gratitude to Dr. Le Trong Nhan for your unwavering support, insightful guidance, and encouragement have been instrumental throughout every phase of this undertaking. The scholarly value of this study has been enhanced by your knowledge and dedication, which has greatly advanced our future career and personal development. We are profoundly thankful for the invaluable lessons, constructive feedback, and inspiration you provided throughout the whole semester. If you have any further questions, please contact us via this email:
dat.nguyenitbk0506@hcmut.edu.vn or thien.nguyenquang1506@hcmut.edu.vn

References

[1] Fayaz Hassan, Line Follower Robot Using PID Algorithm, https://www.electronicsforu.com/electronics-projects/hardware-diy/line-follower-robot-using-pid-algorithm

[2] Ioan Doroftei, Omnidirectional Mobile Robot – Design and Implementation, Technical University Gheorghe Asachi Iasi, 2007

[3] Jung Won Kang & Bong Sung Kim, Development of omni-directional mobile robots with mecanum wheels assisting the disabled in a factory environment, IEEE, 2008

[4] Majed Youns, Position Control Of Robot Arm Using Genetic Algorithm Based PID Controller, University of Mosul, Iraq, 2013

[5] Mrs J Computing, Use Figma to create a drag and drop User Interface for Python using Tkinter, https://www.youtube.com/watch?v=7RXBLeOOiYQ

[6] Sheikh Farhan Jibrail, PID Control for Line Followers, National Institute of Technology, Rourkela, 2013

[7] Synopsys, What is an Autonomous Car, https://www.synopsys.com/automotive/what-is-autonomous-car.html

[8] Tesla, Autopilot and Full Self-Driving Capability, https://www.tesla.com/support/autopilot

[9] TurbineThree, Make Tkinter Python Applications Look Modern In 10 Minutes, https://www.youtube.com/watch?v=PIaccbMT6fo

[10] Yubo Feng & Myung Jin Chung, Integrating Mecanum wheeled omni-directional mobile robots in ROS, IEEE, 2016