

# Autonomous Clue Detection Robot

December 16th, 2024

ENPH 353

Authors: Amjad Yaghi and Richard Helper

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background of the Report . . . . .	1
1.2	Contribution Split . . . . .	1
1.3	Software Architecture . . . . .	1
<b>2</b>	<b>Discussion</b>	<b>3</b>
2.1	Robot Driving Method . . . . .	3
2.1.1	Contour Detection and Basic PID . . . . .	3
2.1.2	Staging . . . . .	3
2.1.3	NPC Avoidance . . . . .	4
2.1.4	Offroading . . . . .	4
2.1.5	Emergency Controller . . . . .	5
2.2	Clue Plate Recognition Module (CNN) . . . . .	5
2.2.1	Data Acquisition . . . . .	5
2.2.2	Image Processing . . . . .	6
2.2.3	Model Architecture . . . . .	7
2.2.4	Training on Images . . . . .	7
2.2.5	Failure Cases . . . . .	9
<b>3</b>	<b>Conclusion</b>	<b>10</b>
3.1	Performance During Competition . . . . .	10
3.2	Unadopted Other Attempts . . . . .	10
3.3	Improvements for Future . . . . .	10
<b>A</b>	<b>Referenced Material</b>	<b>11</b>
<b>B</b>	<b>Notable People</b>	<b>11</b>
<b>C</b>	<b>ChatGPT Usage</b>	<b>11</b>

# 1 Introduction

## 1.1 Background of the Report

This report summarizes the development and implementation of a control system and Convolutional Neural Network (CNN) for a line-following robot that can detect and read clue plates. The robot is part of a competition that requires the navigation of a predefined track, obstacle avoidance, and accurate processing of text in the environment. Throughout the semester, we gained insights and practical experience through labs and coursework, which directly contributed to improving the robot's design and functionality.

[You can view our best run here!](#)

## 1.2 Contribution Split

Richard Helper: PID-based line following and obstacle avoidance.

Amjad Yaghi: CNN-based clue detection, visual processing of clue boards, UI development, emergency teleportation strategy for competition.

## 1.3 Software Architecture

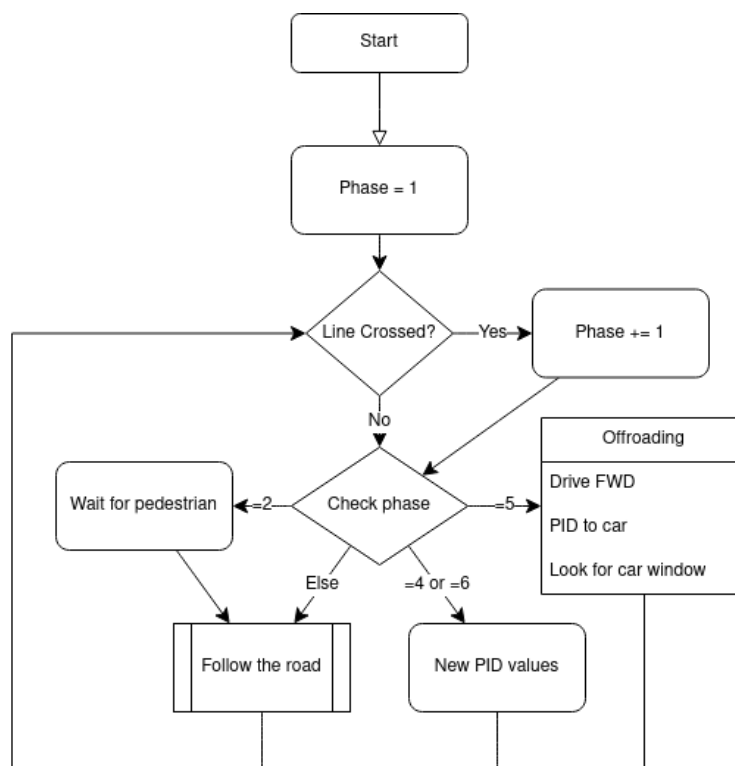


Figure 1: [Driving software architecture](#)

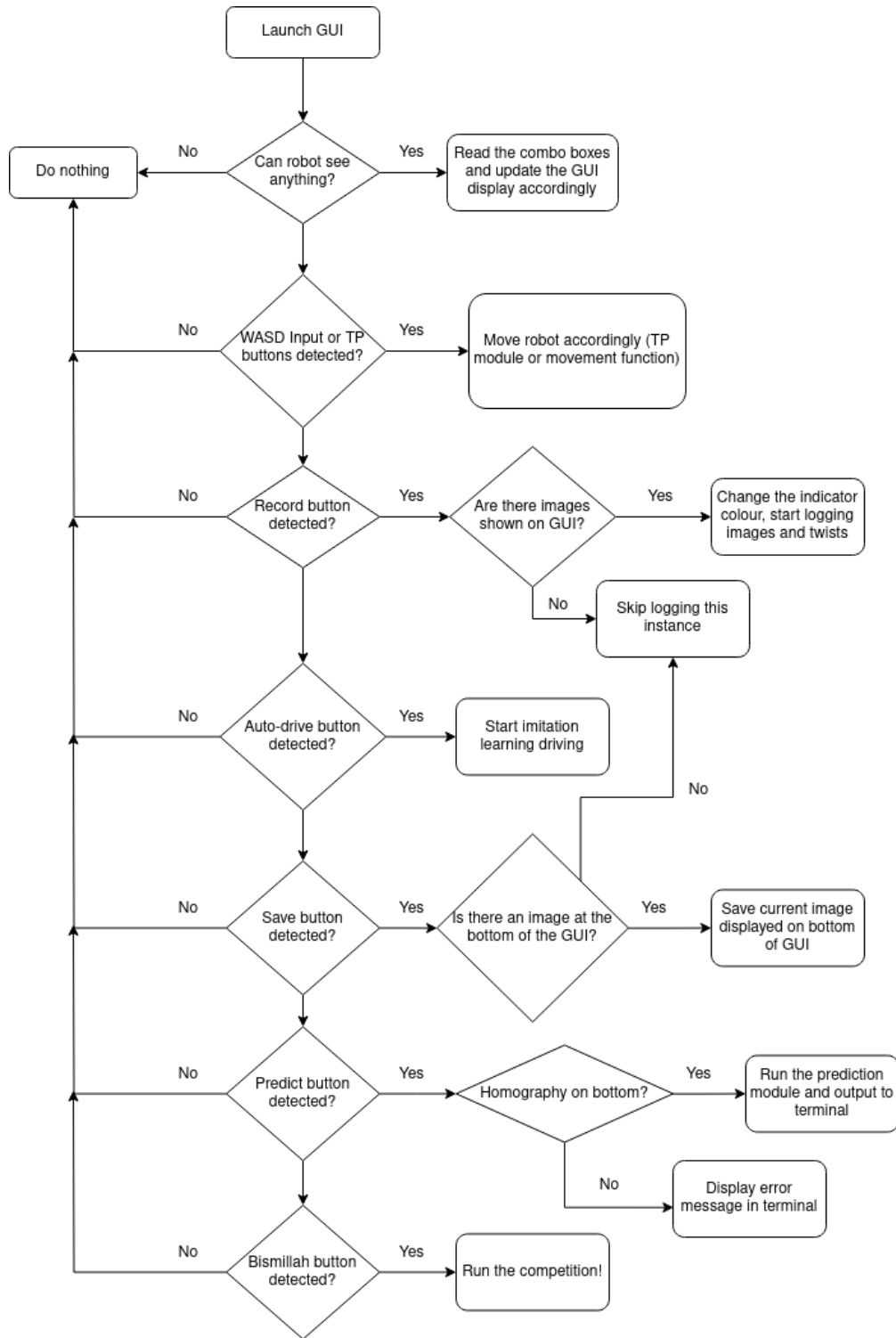


Figure 2: Software architecture used on competition day

## 2 Discussion

### 2.1 Robot Driving Method

The planned driving module used PID, a state machine, and several specific behaviours to make its way through the obstacle course. Unfortunately, we were not able to integrate it with the clue detection in time, and so were forced to go through with our emergency controller detailed at the end of the section.

#### 2.1.1 Contour Detection and Basic PID

Due to the roundabout in the middle of the on-road section, the robot needed to be biased toward turning left so as not to collide head on with the truck. There was also the issue of noise in the middle of the road during the dirt phase. To counteract these issues, the PID was based off the centre of the leftmost contour above a certain perimeter threshold, using an adaptive offset towards the opposite end of the screen that grew smaller as the marker was placed closer to the horizon. See the figure below for an example.

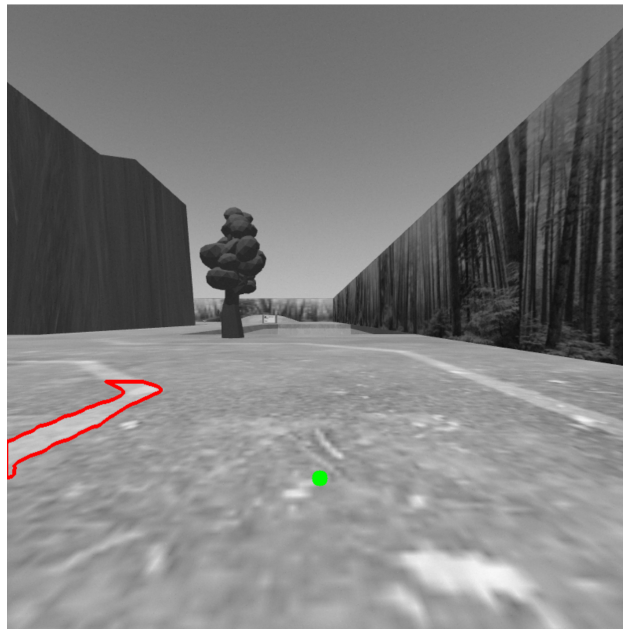


Figure 3: Contour detection and PID marker

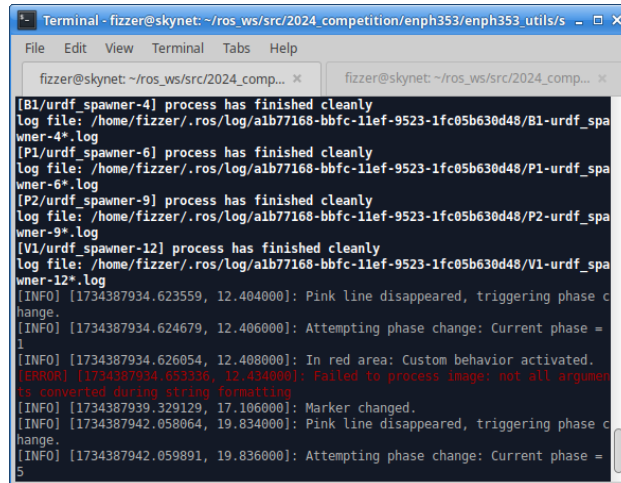
#### 2.1.2 Staging

As the robot drives, it encounters different scenarios marked by lines across the road. There is a red bar that marks the beginning of the pedestrian section and pink ones that mark the transition between the paved road, dirt road, offroad, and hill sections. The drive module uses a state machine to account for this. Whenever one of these lines is detected - that is, the bottom 10 percent of the screen has a certain number of pixels that satisfy the red-pink masking - and then disappears, the next state is initialized. This usually means swapping image processing and control algorithms, as is detailed later in this section.

### 2.1.3 NPC Avoidance

Of the three NPCs, the pedestrian was the only one that the robot would ever hit. The truck acted as a contour in such a way that the PID automatically avoided it, and Baby Yoda was navigated around entirely.

To dodge the pedestrian, the staging system was put to simple use. Upon crossing the red line, the robot swaps to phase 2, and waits for the marker to shift dramatically. As the pedestrian crosses the left side of the road, it would disrupt the contour and cause this shift. The robot would then resume its normal behaviour.

A terminal window titled "Terminal - fizzer@skynet: ~/ros\_ws/src/2024\_competition/enph353/enph353\_utils/s" displays ROS log messages. The logs show the completion of several urdf\_spawner processes (B1, P1, P2, V1) and subsequent phase change attempts triggered by a pink line disappearing. A red error message is visible: "[ERROR] [1734387934.653335, 12.434000]: Failed to process image: not all arguments converted during string formatting".

```
[B1/urdf_spawner-4] process has finished cleanly
log file: /home/fizzer/.ros/log/alb77168-bbfc-11ef-9523-1fc05b630d48/B1-urdf_spawner-4*.log
[P1/urdf_spawner-6] process has finished cleanly
log file: /home/fizzer/.ros/log/alb77168-bbfc-11ef-9523-1fc05b630d48/P1-urdf_spawner-6*.log
[P2/urdf_spawner-9] process has finished cleanly
log file: /home/fizzer/.ros/log/alb77168-bbfc-11ef-9523-1fc05b630d48/P2-urdf_spawner-9*.log
[V1/urdf_spawner-12] process has finished cleanly
log file: /home/fizzer/.ros/log/alb77168-bbfc-11ef-9523-1fc05b630d48/V1-urdf_spawner-12*.log
[INFO] [1734387934.623559, 12.404000]: Pink line disappeared, triggering phase change.
[INFO] [1734387934.624679, 12.406000]: Attempting phase change: Current phase = 1
[INFO] [1734387934.626054, 12.408000]: In red area: Custom behavior activated.
[ERROR] [1734387934.653335, 12.434000]: Failed to process image: not all arguments converted during string formatting
[INFO] [1734387939.329129, 17.106000]: Marker changed.
[INFO] [1734387942.058064, 19.834000]: Pink line disappeared, triggering phase change.
[INFO] [1734387942.059891, 19.836000]: Attempting phase change: Current phase = 5
```

Figure 4: Readout of pedestrian detection

### 2.1.4 Offroading

The offroad section consists of five simple phases. First, the robot drives forwards at an angle towards the hill for a set period of sim time. It's not particularly sensitive as to where the robot ends up, so this worked fine. Then the robot would turn until it detected the windows of the car through the blue mask shown in the attached figure. These windows are a very particular shade of blue, so it doesn't get confused by the clue boards. It then PIDs directly toward the centroid of these pixels until they make up a certain percentage of the screen. Finally, it turns right until the pink line is centred in its view, thus catching a view of the clue, and drives towards the tunnel.

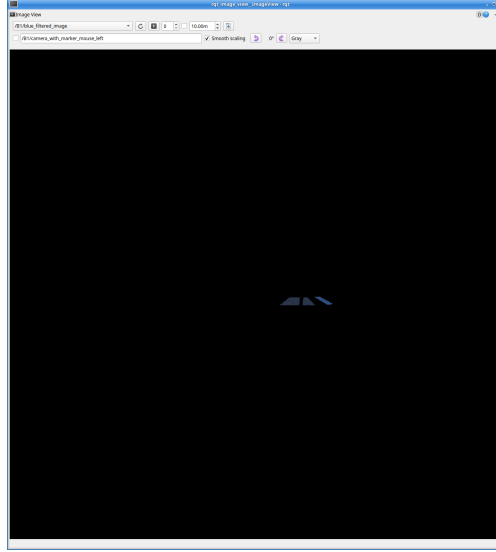


Figure 5: Filtering for car windows

### 2.1.5 Emergency Controller

The controller used in the competition does the following:

Drive for a predetermined time in predetermined steps, wait for the CNN to report a board, and then continue its circuit. This circuit includes teleportation to each staging line. [Here is a link to the python file which contains this sequence.](#)

## 2.2 Clue Plate Recognition Module (CNN)

### 2.2.1 Data Acquisition

All data was acquired manually by going into the simulation after changing the clue boards. Amjad would drive around and screenshot them using the GUI. Here's a demo of what that looked like.

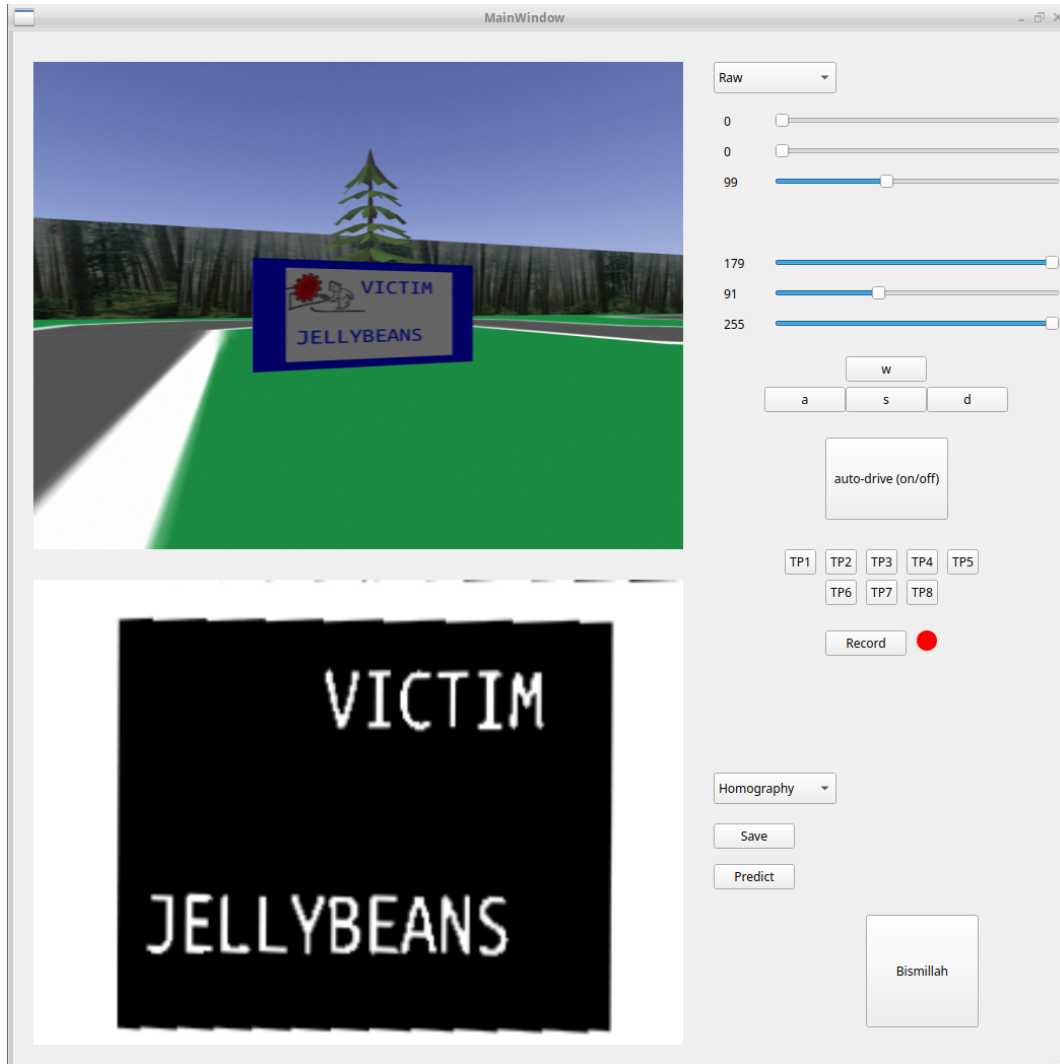


Figure 6: GUI showing homography allowing for easy screenshots using the Save button

This was done tens of times - and although time consuming - gave us very transferrable (as in: relevant to what we would see on competition day) training data. The GUI has drop down menus that allow you to control what you want to see in the display labels. This was very useful for testing how much to threshold, erode, dilute and process images in general.

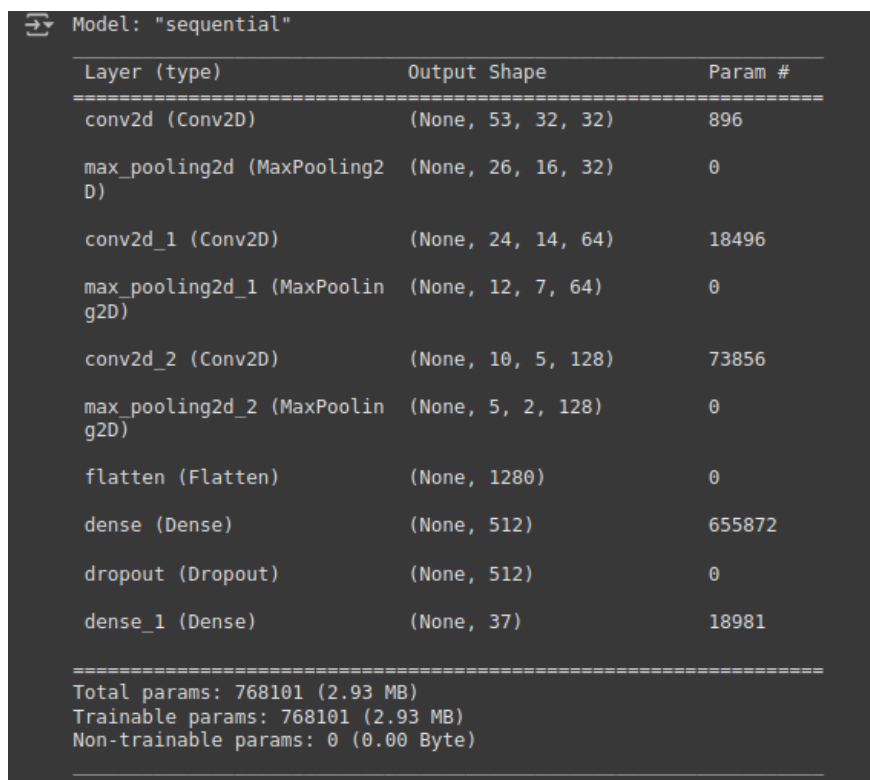
### 2.2.2 Image Processing

Images were then meticulously broken down into contours and stretched to get clear pictures of each letter for the CNN to train on / predict. You can read the exact steps [here, in the prediction module of the repository](#). In short, images were cut in half and a contour box was forcefully drawn around the individual letters. If there was overlap, contours were limited to a certain size and would forcefully split into the best number of boxes in order to separate the letters. [There is also a more digestible version on Google Colab which I will link here as well](#). You can play around and test it for yourself.



### 2.2.3 Model Architecture

The architecture we used for training was inspired by previous work we did in our labs in class, [which you can read more about here](#). The following is an image of the model summary.



The image shows a terminal window displaying the summary of a sequential model. The table lists 11 layers: conv2d, max\_pooling2d, conv2d\_1, max\_pooling2d\_1, conv2d\_2, max\_pooling2d\_2, flatten, dense, dropout, and dense\_1. It provides the output shape and the number of parameters for each layer. At the bottom, it summarizes the total, trainable, and non-trainable parameters.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 53, 32, 32)	896
max_pooling2d (MaxPooling2D)	(None, 26, 16, 32)	0
conv2d_1 (Conv2D)	(None, 24, 14, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 12, 7, 64)	0
conv2d_2 (Conv2D)	(None, 10, 5, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 5, 2, 128)	0
flatten (Flatten)	(None, 1280)	0
dense (Dense)	(None, 512)	655872
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 37)	18981
=====		
Total params: 768101 (2.93 MB)		
Trainable params: 768101 (2.93 MB)		
Non-trainable params: 0 (0.00 Byte)		

Figure 7: Model architecture summary

### 2.2.4 Training on Images

Since we did not do any data augmentation (no artificially generated data based on what we had already), we needed as much of our data set used for training as possible. As such, the strategy was to monitor training using a 80-20 training-validation split, and once it was satisfactory, to remove the validation set entirely and train on the complete set of images.

```

Training the Model

1 # Load the model if you want to continue training
2 # model_save_path = '/content/drive/My Drive/ENPH353_FIZZCOMP/character_recognition_model.keras'
3 # model = load_model(model_save_path)
4
5 # Compile the model
6 model.compile(
7     optimizer='adam',          # Optimizer for training
8     loss='categorical_crossentropy', # Loss function for multi-class classification
9     metrics=['accuracy']      # Metric to evaluate during training
10 )
11
12 # Split the data into training and validation sets
13 from sklearn.model_selection import train_test_split
14
15 # 80% for training and 20% for validation
16 X_train, X_val, y_train, y_val = train_test_split(
17     data, labels, test_size=0.20, random_state=42, shuffle=True
18 )
19
20 # Print the shapes to verify split
21 print(f"Training data shape: {X_train.shape}")
22 print(f"Validation data shape: {X_val.shape}")
23
24 # Train the model
25 history = model.fit(
26     data,
27     labels,
28     #X_train,
29     #y_train,
30     epochs=10,          # Set to a low number for quick testing
31     batch_size=32,      # Batch size
32     #validation_data=(X_val, y_val) # Validation data
33 )
34

```

Figure 8: Fully using our dataset

Notice in the figure above, `x_train` and `y_train` are commented out, this is because we were using all letters to train our final competition model. Below is a sample of what our accuracy, loss and confusion matrix looked like when we actually had a split:

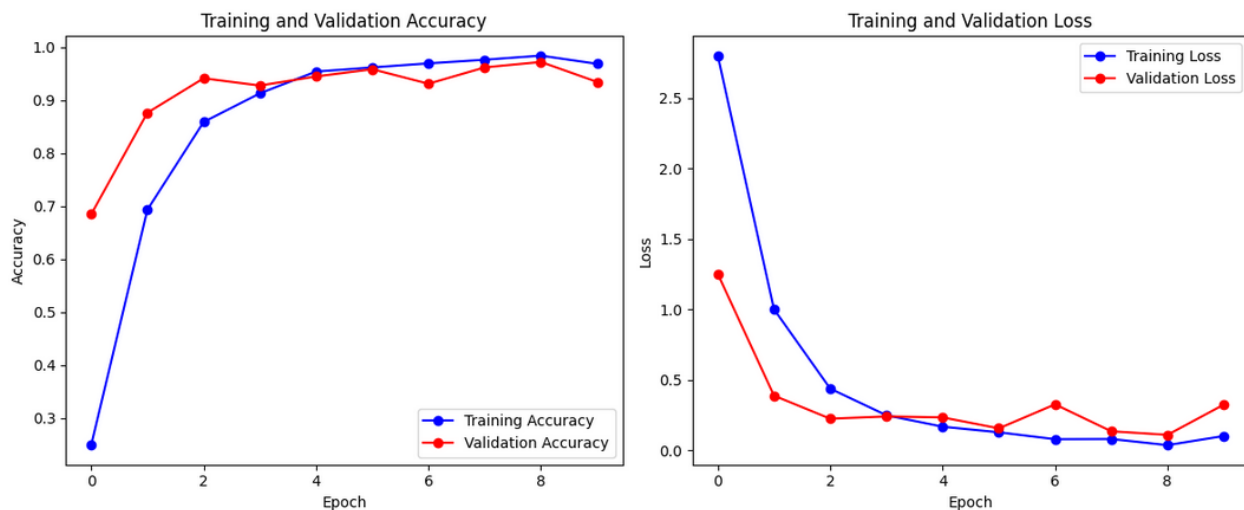


Figure 9: Training and validation graphs when validation set is 20 % of data

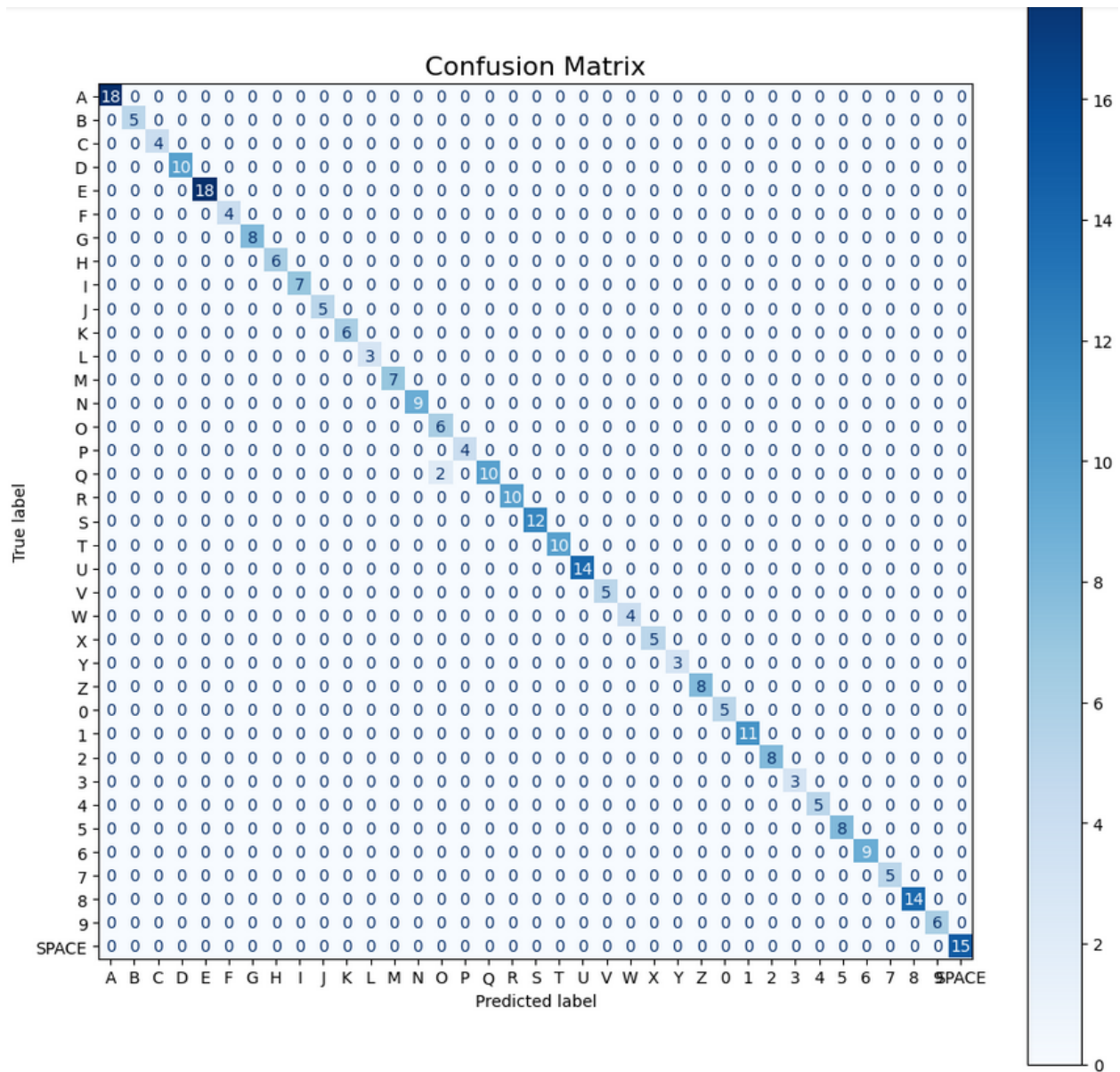


Figure 10: Beautiful confusion matrix

## 2.2.5 Failure Cases

Occasionally, when segmenting the letters, there is enough overlap to cause a failed prediction. There was an attempt to solve this by intentionally training on overlapped images, however not enough was manually captured. In Section 3.1, we discuss our performance during competition. Unluckily, one of our predictions failed as a result of this overlap; however, this is a rare event. [You can view the individual letters that were used to train the model here](#). You may notice some of them have a lot of overlap (by design).

## 3 Conclusion

### 3.1 Performance During Competition

The robot performed almost as well as designed during competition. It correctly reported four different clues, teleported three times, and crossed the line once to a total of 18 points. Due to one misspelled clue, we were not able to maximize our points, but there was no catastrophic malfunction.

### 3.2 Unadopted Other Attempts

Notice, in Figure 6 which shows the GUI, that there is a button labelled "Record" with a red light next to it. This was because due to conflicts integrating driving and clue detection together, Amjad tried to quickly create an imitation learning model to navigate the robot. The record button would take screenshots every 100ms and create a csv file which mapped the current linear and angular velocity of the robot to the image. Images were then fed into [another CNN](#) with very similar architecture to the one used for training on letters. Surprisingly this method had accurately navigated the first segment of the road up to the second clue board. However, it became quickly apparent that we did not have the necessary compute (shoddy laptop :/) in order to use an imitation learning model in time for the competition, thus it was scrapped and the emergency teleportation method was used instead.

### 3.3 Improvements for Future

The primary area in need of improvement is the integration of clue detection and driving. We would have used a very wide angled camera so that the clues could be seen without stopping, and altered the homography such that it could undo the resulting distortion. This would also include getting the robot to reach the top of the hill, and ironing out the CNN overlap issue discussed in Section 2.2.5. Secondly, there is a lot of potential to optimize the usage of nodes to separate actions which needed to occur simultaneously on the robot.

# References

## A Referenced Material

All referenced material can be found on the ENPH 353 website. There, you will find access to eight labs which helped us in various ways for this project.

[Here's a link.](#) It will likely update as months pass.

## B Notable People

We would be remiss not to mention Daniel Song, Michael Khoo and Ebrahim Hussain who gave input regarding their previous experience in this course.

Lastly, many discussions were had with Ella Yan (a colleague taking the course) particularly regarding image processing and how to implement an imitation learning model. We would like to give our flowers to her and her teammate Nora Shao for managing to implement their imitation learning model with severe time constraints.

## C ChatGPT Usage

ChatGPT was used to help implement ideas that we already had, and for debugging. Here is an example of how ChatGPT was used to help us troubleshoot making predictions.

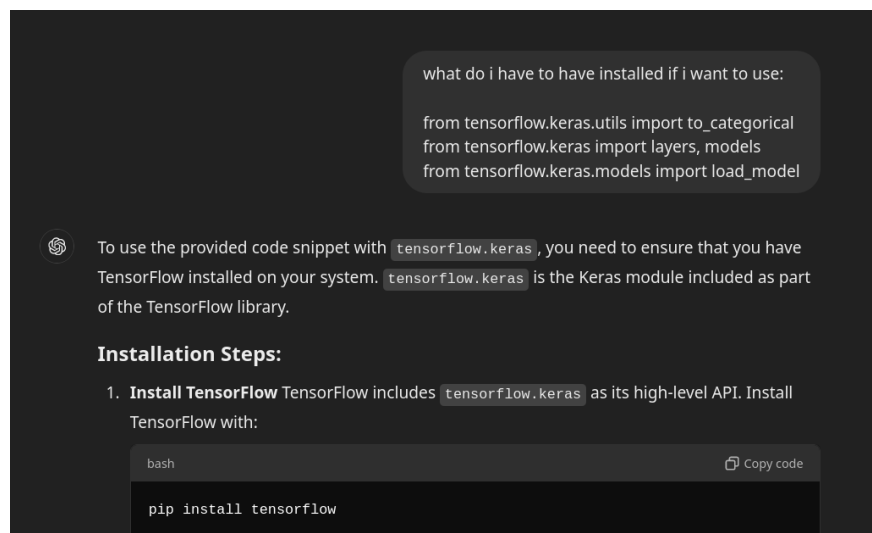


Figure 11: Troubleshooting with ChatGPT

In fact, this was a notable conversation with ChatGPT, as it helped to reveal that there was a mismatch between the version of the model in Google Colab, and the one used locally! (Took a while to figure out what was wrong.)