ENPH 353 Robot Competition 2025

Rachel Miner, Alfred Gunawan

January 2025 to April 2025

Team 2: The Smithies
ENPH 353
Engineering Physics
The University of British Columbia
April 7, 2025

Contents

1 Introduction

Background

This project uses convolutional neural networks (CNNs), PID controllers, and various image processing techniques to drive a robot in a simulated ROS environment using Gazebo, enabling it to navigate and read clueboards.

ROS and Gazebo

ROS (Robot Operating System) is an open-source robotics framework that provides tools and libraries for building robot applications. It facilitates communication between different components of a robot system using a publish-subscribe architecture, making it easier to modularize tasks like perception, planning, and control. Gazebo is a 3D robotics simulator that integrates with ROS to simulate complex environments and physics. It allows developers to test and visualize robotic systems in a virtual world before deploying them to real hardware, significantly accelerating the development process.

CNNs

CNNs stand for convolutional neural networks and are a type of neural network that is particularly well suited to working with images or other grid like matrix datasets. Similar to conventional neural network architecture such as multi layer perceptrons (MLPs), CNNs are also comprised of multiple layers. There are various types of CNN layers that do various things such as convolutional layers that apply filters, pooling layers which reduce spatial dimensions, dense layers that perform regression, etc. In the context of this project, a CNN takes an image as an input and outputs a prediction on what type of alphanumeric character is found in the image.

PID Controls

A PID controller–short for Proportional, Integral, Derivative-is used to keep the robot centered on the road by minimizing the error between the road center and the camera center. The proportional term responds to the current error, while the derivative term dampens oscillations by accounting for how fast the error is changing. The integral term, which addresses accumulated past errors, was not used in this project. Only the P and D components were implemented for smoother control.

Image Processing

This project used a range of image processing techniques, including RGB filters and HSV masks. HSV (Hue, Saturation, Value) makes it easier to isolate specific colors like fuchsia compared to standard grayscale. Binarization helped simplify visual input by removing background details and highlighting key features like the road and clue boards. Cropping focused analysis on expected regions of interest. Perspective transforms were used to make isolating characters from analysed clue boards easier by reducing the impacts of rotation/skew in the raw image and displaying the board as a well defined rectangle.

Project Objective

The objective of this project is to use ML, particularly CNNs, and PID controls to make a robot navigate a simulated environment with moving obstacles and read clueboards to solve a murder mystery.

Contributions

The contributions were split in two main categories: driving and clue reading. Rachel was in charge of driving. This involved line following, navigating the different biomes in the environment, avoiding obstacles such as the pedestrian, and optimizing camera view of the clueboards (including adding two extra cameras to Robbie's URDF). Rachel was also in charge of starting/stopping the timer. Alfred was in charge of reading clueboards and publishing read clues. This involved recognizing when a clueboard is visible and developing a CNN to read the clues written on it as well as handling publishing guesses.

Overarching Software Architecture

The driving software and clue-reading software were kept largely in separate scripts, though there were overlaps that were eventually discarded. They were activated together through a launch file.
Driving:
The driving software is somewhat of a monster due to the hundreds of lines of code involved, but is fairly straightforward in terms of logic. Line following was done using PID, and image processing functions were based on which area of the environment the robot was located in. Recognizing the movement of the obstacles was done through binarizing images to filter out everything (or most things) except the obstacle in question and noting whether the number of contours found indicated that an obstacle was seen. The trip up the mountain was hard-coded entirely, but all other sections other than some transitions between zones were done via line-following.
Clue Reading
Clue reading essentially takes frames from the two extra cameras and outputs a clue to be publuished. This process is comprised of two main steps: data preprocessing, and CNN analysis.

- Data preprocessing involved isolating the largest clueboard in the frame, cropping the frame to just this clueboard, doing a perspective transform to a rectangle to mitigate rotational variance, and then using contours to isolate individual characters.
- After individual characters were attained, they are passed into a CNN that takes 108x108 images and returns a prediction on what type of alphanumeric character it is.

Guesses are input into a csv database and guesses with a high enough frequency are published.

2 Driving Module

Basic Overview

Driving was a difficult task to approach due to the number of things to take into account. To break down this problem, the entire simulated environment was separated into ”zones” which indicated what actions the robot should take. The main ”Driver” class contained a large portion of the code, particularly in its image_callback function. Within the image callback, the robot would check to see if he had to stop (using functions called from another script called road_processing which performed image processing based on zone and return True or False). If he didn't have to stop, he would line follow with a binarized image and set speed, both also based on zone. If the stopping function returned True, he would take a series of actions based on whichever zone he was in.

Stopping Points and Zones

To see the visual breakdown of zones, see Appendix B.

There were many stopping points defined, which all made the robot switch zones after passing them. The stopping points, and all other areas where any hard-coding was necessary, were as follows:

1. *Pedestrian crossing*

   In zone 0 (at the very beginning of the competition start), the robot checks to see if there is a red line in the correct positioning to be a ”stopping point”. An HSV mask is used to filter the red line, and the function called only returns true when the line is below a certain height on the camera image (ie the robot is close enough) and the line is roughly horizontal (ie the robot is oriented correctly).

   After stopping, the robot continued to analyze red-masked images. My motion detector function, in this zone, returned True when the number of red contours exceeded 1 (i.e. the red line was broken by the walking pedestrian). The robot waited to see movement, then waited till the movement stopped so he could move forward until he passed the crosswalk (hard-coded). (This is the entirety of zone 1).

2. *Truck Loop Entry*

   To find the entry to the truck loop is an extremely illogical function (in human logic, but it works great in robot logic). I had planned to, after binarizing the road, check the size of the road contours in the upper left and upper right sides if the image, however, during testing to see how large to make it, I accidentally measured len(contours) rather than the area. Doing this, I was able to notice that the number of contours increases to ¿150 right before the intersection, then returns down to 30 where I wanted it to stop and turn into the loop, so I made the stopping point using that. Following the stop, the robot waits to see when the white contours of the truck appear then disappear, then take a hard-coded left turn into the loop, where it continues to line follow.

3. *Truck Loop Exit*
   The exit of the truck loop is very similar to the entrance in that it is looking at len(contours). It skips reading the first 100 frames after starting to line follow, then finds a frame with 100 ¡ len(contours) ¡ 150. He then takes another hard-coded left turn to exit the loop (he is now in zone 4).

4. *Fuchsia Line Entry to Grassland Biome*
   The entry to the grassland biome is pretty much the same as finding the pedestrian, just using a fuchsia mask for hsv rather than red. When entering the grassland biome, the image processing changes, using hsv and erosion to deal with the very messy texture of the biome. Instead of line following the road, the robot line follows the edge of the road in a slower pace (0.5x normal road speed). See

5. *First Large View of Pond*
   When the robot is facing the pond, once he gets close enough, there is a stopping point. There is another hsv mask which only includes the pond (mostly), but due to the texture of the pond, there are a ton of contours here as well. When the number of contours reaches 15000, the robot stops, turns a little towards clueboard 5, then returns to the road, incrementing zones.
   This was necessary due to the clue reading being very dependent on picture quality and distance. Otherwise, this stopping point would not have been implemented.

6. *Beginning of Pond Straits*
   Once the robot is facing the strait between the two ponds (also found via contours), the robot crosses the road to avoid colliding with clueboard 6 (which would lie in the path of the robot otherwise).

7. *Fuchsia Line by Yoda-land*
   The robot stops at the fuchsia line, and there is a dark brown mask which only allows Yoda's cloak through. Once the robot sees Yoda, he teleports to the last fuchsia line right by the tunnel. There, he backs up to read the sign, then is hard-coded to go up the mountain until he reached the sign and is positioned such that his right camera views the sign clearly.

To see images showing the image processing of each stopping point, see Appendix A.

3 Clue Reading Module

Basic Overview

The clue reading problem essentially boils down to how do we get the robot to recognise a clue board in the image feed, read the correct clue from it, as well as publish the correct clue. Two

nodes were created for this, a left and right clue reader which read images from the left and right cameras respectively. The primary reason for creating two nodes was to exploit the fact that the 1st, 3rd, 5th, and 7th signs would almost always be to the left of our robot and vice versa. As stated in the overarching software architecture section, once an image was attained, clue reading occurred in 2 main phases: data preprocessing, and character reading via CNN. I'll first go over the data processing and then cover the neural network.

Data Preprocessing

Once an image is obtained from the camera feed, the image callback function begins by binarizing the frame into grayscale and finding contours. The largest contour is nearly guaranteed to be the clue board at any distance where clues are readable so we find the bounding box of the clue board in the image. Once this is found, we perform a perspective transform into a rectangle to account for any rotation or skew and we get a clearer rectangle that is easier to work with. Finally, to extract characters to pass into the CNN, we convert the board into hsv and threshold to focus only on the blues of the image (H value of 90-130) which leaves only the characters. Contours are then found and we get the bounding boxes, then return the cropped characters sorted left to right so we can assemble the clue later.

Character Reading

This section is relatively straightforward, our CNN takes in inputs of 108x108 images and outputs a prediction on what alphanumeric character the image represents. Since we've already attained images from the previous step, we evaluate using the model and then add the prediction of the clue to a database to be used for publishing. Some further details:

- Training data was gathered through loading the competition environment and driving around manually while taking screenshots of the camera feed to simulate frames that the robot would see in the actual competition. These screenshots were preprocessed as any frame would and the resulting character images comprised the training/validation dataset.
- The CNN is comprised of 8 total layers. The first 4 are 2 pairs of convolutional and max-pooling layers. The first and second convolutional layers were comprised of 16 and 32 filters respectively while both maxpooling layers used 2x2 kernels. The fifth layer is the flattening layer, the sixth is a droput layer set at 0.4, and the last two are dense layers comprised of 128 and 36 neutrons respectively using reLu activation functions.
- The CNN was trained on tensorflow 2.12.1 and keras 2.12.0 with a learning rate of 1e-4, a batch size of 16, and 50 epochs. There were 353 total datapoints (purely coincidence) in the training + validation dataset. A histogram of the data used to train the model, an accuracy vs epoch plot, and a confusion matrix can be found in Appendix C.

Clue Submission
Character reading was conducted on both the top and bottom half of the board to get the clue type and value respectively. For each frame, the read clue type and value were saved to a csv along with the frequency of each guess. Sometimes, the full clue type wouldn't be read correctly so to account for this, the nodes would determine clue type based on the first letter read of the clue's type. For example, if a guess's clue type's first letter was 'S', then the node would publish 1 to the score tracker for 'Size', a 'B' would correspond to 8 for 'Bandit', etc. Once a certain guess hit a threshold frequency (Set at the low value of 2 to ensure the robot wouldn't be sitting on a correct guess without publishing), the node would go through the database and gather all the clues with the same first letter of the clue type and then publish the longest clue. The choice of publishing the longest clue was done to avoid faulty guesses since, in practice runs, wrong guesses were always shorter than the correct guess.

4 Integration of Both Modules

As stated in the integration, the final method for integrating the two portions of the project together were simply by including both main scripts in a launch file. While the driving code started the timer, the clue reading code sent the read clues to the score tracker.

5 Conclusions

Robot Performance

Our final score during the competition was 18. The driving worked as expected, however a bug found in the clue reading code only right before the competition led the robot to publish only the first longest clue read rather than the one which was published most often. Therefore, many of the clues which had been correctly read were incorrectly published. Had this issue been fixed, the score would have been 36 rather than only 18.

Abandoned Methods

Driving-Clue Reading Integration
Originally, the driving module relied heavily on the clue reading as some of the zone changes relied on knowing how many clues had been read, however, it was explained on Tuesday (two days before the competition) that this was going to be impossible based on how clue reading was to be implemented, so last minute large changes needed to be made to the driving code to figure out how to know when to start looking for the truck. This caused the initial integration of having the Driver class reference the ClueReader class to be changed to having a launch file which included

both. These were all then moved from the ”node” folder to the ”scripts” folder.

Driving

In the driving implementation, there were many, many iterations to much of the code. I will list only the larger changes from the initial method. Originally, the code finding the stopping point would increment the zone in the Driver class, however this ended up being inconvenient due to the zone being incremented indefinitely every time the robot stopped. Another change was motion detection. At first, the motion detection looked at a previous image as well as the current image and looked for changes, however due to lighting changes, there was always a false positive, and tuning proved very annoying and finnicky. Hence, contours were used instead.

Clue Reading

Implementing the CNN was relatively straightforward with the only big change from original iterations being the capability to read numbers as well as letters due to not realising numbers could also be part of the clues. What was changed a lot was the clue board processing. Clue board isolation was initially done with a combined blue hsv and contour mask but was switched to just a blue hsv mask plus cropping out the top 40% of pixels. Character isolation proved to be challenging initially. I experimented with cropping the clue board into equal segments of 12 but rotational and translational variance made this challenging to execute. In the end, I settled on using contours with erosion to isolate the characters which provided decent results.

Future Improvements

Driving Improvements

The biggest improvement I would make to driving would be to use Imitation Learning. This is not only to make driving a little smoother (something I noticed from the teams I saw use this), it would also have made driving in the grasslands much easier rather than spending hours upon hours working on image processing. It would also have been a greater learning experience, and perhaps faster than the amount of tuning required for my method (given that the driving implementation, including optimization, took around 50 hours). Another change would be to follow Yoda to the tunnel rather than teleport. This was the original plan, however there was too much time taken optimizing camera view to be able to fully attempt this.

Other flaws include: first, its large size and many functions per script made it hard to read and modifyclearer structure and separation would have made it easier for Alfred to test independently. Second, excessive hard-coding made the system fragile to changes. Lastly, heavy computation in the main image callback slowed performance; running driving and clue reading together initially caused failures, forcing us to lower the Real Time Factor to 0.5.

Clue Reading Improvements

1. *CNN Input Size:* From discussion with classmates, the CNN taking inputs of 108x108 is unusual and on the much larger side. I've heard of sizes from 28x28 to 16x16. Regardless, the large input size resulted in a high computation time that may have had an impact on our robot's performance as handling both the driving and clue reading nodes was demanding. From previous labwork in the course, I chose the large size to increase accuracy of the CNN. To some extent, this was effective as the main factor that dictated correct clue submissions was good character isolation and not the CNN itself but it's evident that for the competition, a lower computation time would've been more beneficial.

2. *Double Submissions:* With the way I coded clue submission logic, more than one guess of a unique clue type may be published over the course of a run. To avoid overwriting a correct clue guess, I made it so that our robot published the longest clue of that type in the database of detected clues. This only worked due to the observation that wrong guesses were always shorter due to the way character isolation was coded. A better way to address this problem would have been to pivot to only publishing once for each clue type. The robot taking the single frame for each clue type where the board was largest and clearest and analyzing just that would leverage the high accuracy of our CNN and cut down massively on computation time as well as completely avoid potentially wrong double submissions.

3. *Not Submitting Correct Clues:* After looking through the databases of collected clues of multiple test runs as well as the competition run itself, I found that the robot may read the correct clue but would simply not publish it. This is most likely caused by me not adding a delay between each clue submission or the high computation time. Out of all the issues, this was perhaps the most frustrating as if this issue had been fixed by competition, our team would've gained an additional 18 points which would've doubled our score.

General Improvements

Decreasing the RTF further would have made processing time less of an issue, therefore giving the clue reading more reliability and potentially having less issues. Also, increasing the quality of the side cameras would have also accounted for flaws in the clue reading, so that the driving wouldn't have to result in such precise distances and angles from clueboards.

6 References

Ron Kitainik - Referenced by Rachel

While working in the penthouse, Ron K and Rachel discussed driving tips quite often (not showing each others' code, but simply discussing ideas). One of these discussions was regarding how to image process the grassland, and many tips that Ron suggested were useful as they brainstormed together.

Krista Traboulay - Referenced by Alfred

Alfred discussed with Krista methods for processing data to attain inputs for clue recognition CNNs as well as potential architectures for CNNs.

Ron Nelson - Referenced by Alfred

Ron N and Alfred discussed the potential to leverage the fact that clue board generation was deterministic to better isolate characters for CNN input. Although this method was never implemented, the discussion led to advancements on the final data preprocessing such as the perspective transform.

Miti Isbasescu - Referenced by Rachel and Alfred

Miti is listed here for obvious reasons, but most notably for recommending against RL for driving.

Ada Li and Connor Fransoo - Referenced by Alfred and Rachel

For general help on many issues. In particular, discussion with Ada was what led to the algorithm to prevent double submission of clues and discussion with Connor leading to choose PID over Imitation Learning.

ChatGPT - Referenced by Rachel and Alfred

ChatGPT was particularly helpful for de-bugging, especially small typo-type mistakes (such as constantly forgetting self. when calling functions or variables or calling the wrong one since their names were all quite similar) and syntax errors.

Ethan Predinchuk - Referenced by Rachel

Recommended to use erosion to help with image processing for the grasslands.

A Appendix 1: Image Processing for Driving

Motion Detection

Pedestrian Detection
Truck Detection

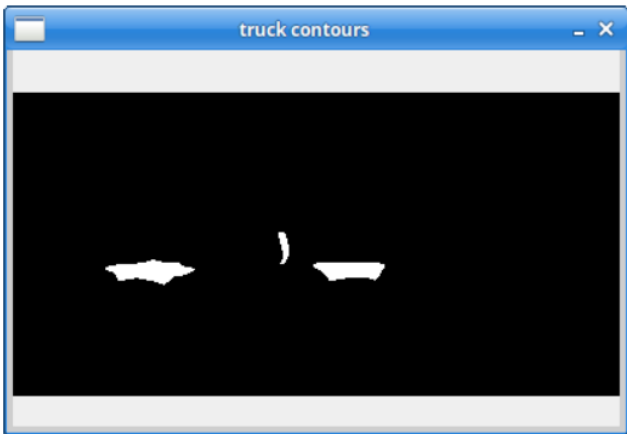

Figure 1: Processed Truck Image

Yoda Detection

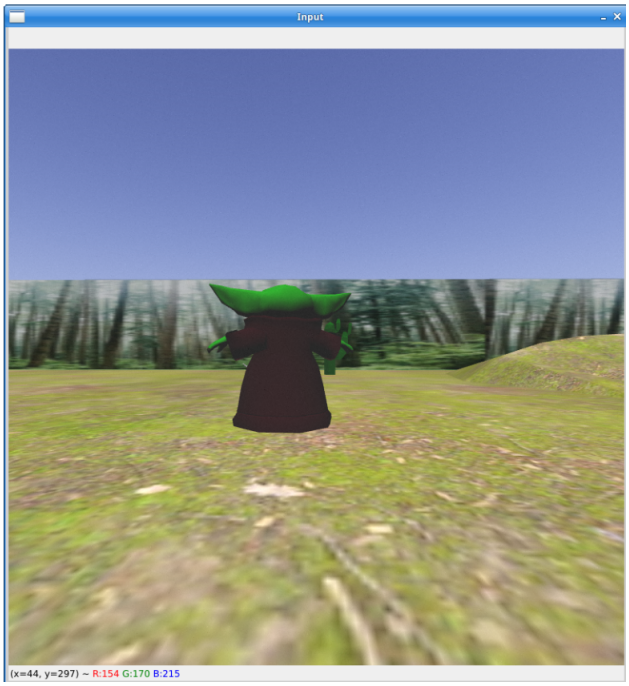

Figure 2: Raw Yoda Image

B Appendix 2: Driving Zones

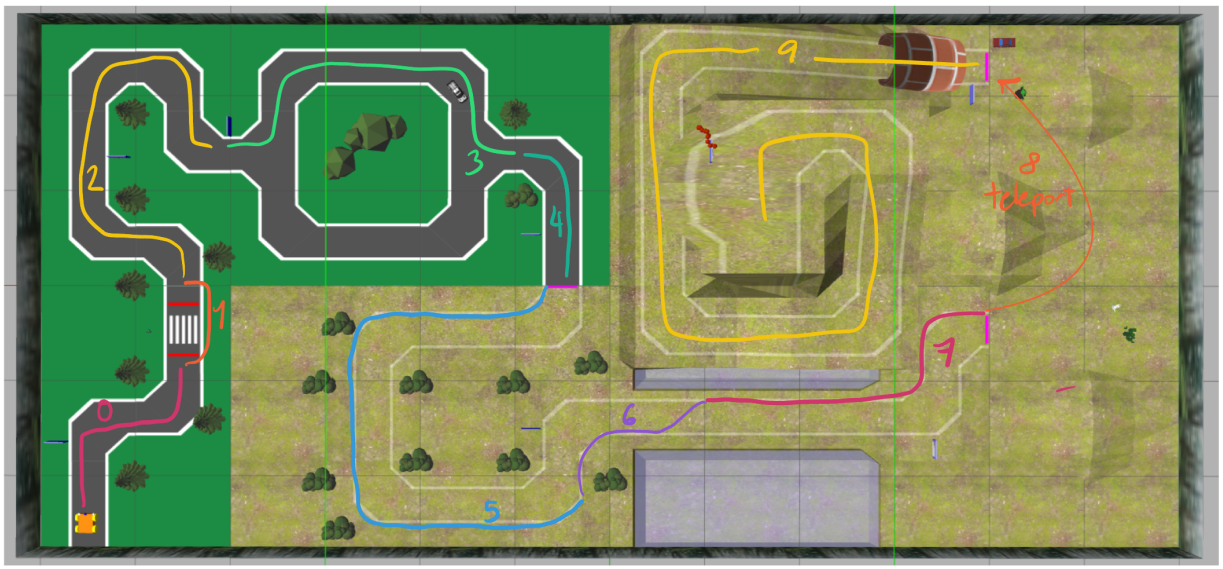

Figure 3: Map of zone changes

C Appendix 3: CNN Training Results and Parameters

Training Data Histogram

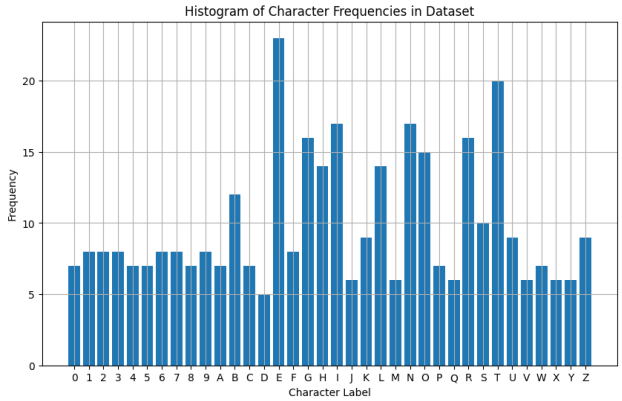

Figure 4: Histogram of Training Data

Accuracy vs Epoch During Training

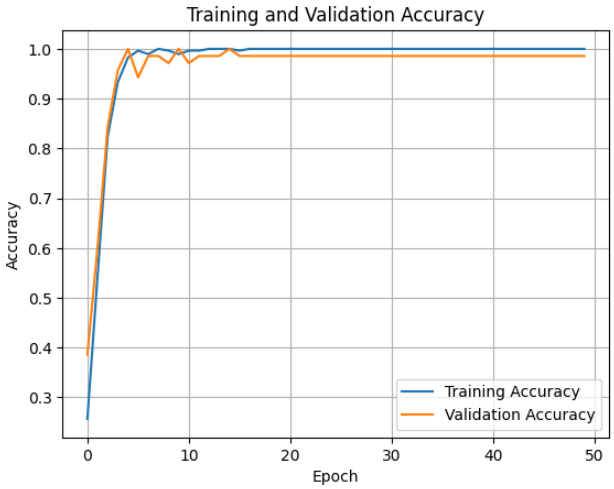

Figure 5: Accuracy vs Epoch

Confusion Matrix

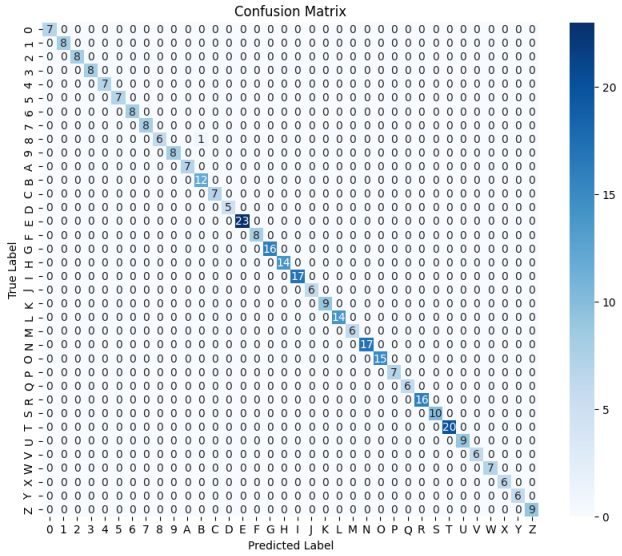

Figure 6: Confusion Matrix

D Appendix 4: Data Preprocessing for the CNN

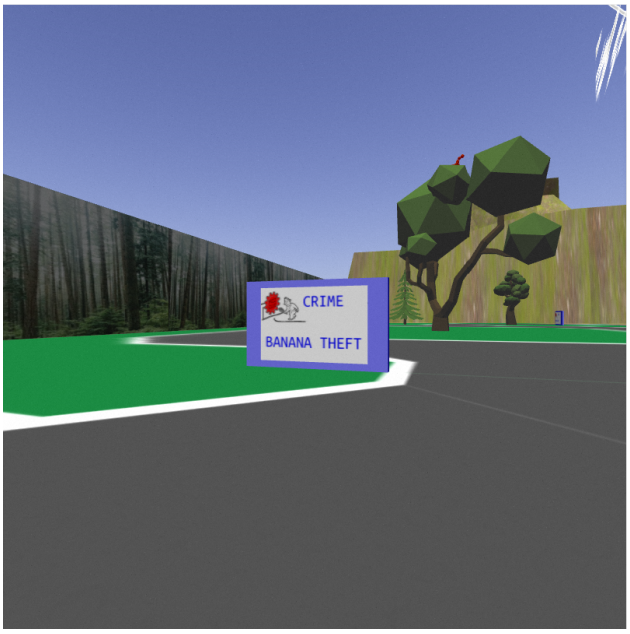

Figure 7: Example of a Raw Training Image

Figure 8: An Example of a Cropped Board

Figure 9: An Example of an Isolated Character

Figure 10: Another Example of an Isolated Character