

Gadalin:

Autonomous Parking Enforcement Robot

Prepared for

Miti Isbasescu

ENPH 353: Engineering Physics Project Design

Prepared by Group 14

Simon Ghyselincks

Submitted on April 22nd, 2023

Department of Engineering Physics

The University of British Columbia

Vancouver, BC

Table of Contents

1.0	INTRODUCTION.....	1
2.0	SUMMARY	1
3.0	STRATEGY.....	1
4.0	SOFTWARE ARCHITECTURE	2
5.0	PLATE DETECTION	2
5.1	DETECTION PROCESS	2
5.2	DATA GATHERING STRATEGIES	3
5.3	DATA PREPROCESSING	4
5.4	CNN MODEL AND TRAINING	5
5.4.1	Architecture.....	5
5.4.2	Training Parameters	5
5.4.2	Validation and Performance	6
5.5	MODEL TESTING AND ERROR ANALYSIS	6
6.0	CONTROLLER AGENTS.....	6
6.1	OUTER TRACK MOTION CONTROL.....	6
6.1.1	Skyscanner.....	7
6.1.2	Fixate Tuning and Optimization	8
6.1.3	Hardcode Segments and Plate Readings.....	8
6.2	PEDESTRIAN DETECTION	8
6.3	TRANSITION TO INNER LOOP AND TRUCK DETECTION.....	8
6.4	INNER LOOP AGENT	9
7.0	CONCLUSIONS	9

List of Figures

Figure 1- Competition Environment (ENPH 353 Course Page)	1
Figure 2- Placard Identified and Highlighted with Slicing Points	3
Figure 3- Army of Robots.....	3
Figure 4- Data Processing Options (Rightmost Selected).....	4
Figure 5 - (a) Parking ID Data, (b) Initial Character Data, (c) Trimmed Character Data and Distribution..	4
Figure 6- Plate Characters CNN	5
Figure 7- Prototyping SkyScanner.....	7
Figure 8- Fixation Marking on Competition Image.....	7
Figure 9-Pedestrian Crossing Development	8
Figure 10- Truck Collision Prevention	9

1.0 INTRODUCTION

Gadalin is a self-driving robot that autonomously navigates through a simulated competition environment in ROS Gazebo while seeking, identifying, and reporting license plates on parked cars. The robot was built and developed to compete against other teams as part of the ENPH 353 Engineering Design Course at The University of British Columbia (UBC). The goal of the competition is to score points by correctly reading parking plates and IDs, and to avoid losing points through traffic infractions or collisions. The project aims to refine and develop machine vision, machine learning, and engineering design practices which are emphasized throughout the course.

2.0 SUMMARY



Figure 1- Competition Environment (ENPH 353 Course Page)

The competition layout has 8 parked cars distributed through two connected traffic rings. There are two active pedestrian crossings, a moving truck, and an offroad section with shadows to navigate through. A full loop around the outside track nets extra points. In the event of more than one team scoring full points, the time duration of completion is used as a tie-breaker.

As part of the competition rules, the robot does not have access to any simulation data aside from a virtual camera feed. Driving commands to control the robot are sent via an autonomous controller agent developed as part of the project, while data from placards on the parked cars needs to be identified and reported to a score tracker. There is no feedback to the robot on the correctness of the plate character predictions, and the robot is free to navigate and make predictions until either it stops the competition timer, or the 4-minute time limit is reached. At the end of the round, the last prediction for each plate is tallied against the correct actual values to determine a score.

The goal is to drive a full loop of the outer track, visit all 8 parked cars, and make 8 correct parking number and plate identifications, while avoiding any collisions or exiting the painted road lines with more than two wheels.

3.0 STRATEGY

The strategy for Gadalin was divided into two branches: driving and plate identification. For plate identification, an early development goal was to find a technique to make a large set of labeled training data for training a Convolutional Neural Network (CNN). Although there are ways of generating synthetic data, my goal

ENPH 353 Gadalín Robot Design

was to find ways to harvest data directly from simulation in an automated fashion so that the training data matches the competition data as close as possible.

There were many options available to investigate for driving, including road-following PID, reinforcement learning, and imitation learning. The robot must be fast and reliable to complete the course in a competitive time without losing any points from road excursions or collisions. Ultimately, after some early driving tests with a manual XBOX 360 controller and work completing the plate detection, an interesting classical machine vision and PID strategy that uses environmental cues to realign with the outer loop road sections was discovered, see Section 6.0.

4.0 SOFTWARE ARCHITECTURE

The software structure and repository is divided into four areas: the robot controller, competition environment, plate data generation, and CNN training areas. Links are provided below:

Software Source Code	GitHub Repository
Main Repository	https://github.com/chipnbits/gadalín
Robot Controller	GitHub
Competition Environment	GitHub
Plate Data Generation	GitHub
CNN Training Area	GitHub

An overview of the directories and file structure can be found in Markdown within the GitHub readme, as well as the graphical diagram shown in Appendix A.

The robot controller is run as a ROS Python node, which is launched in parallel with the plate tracking and reporting node. By having them as separate nodes, they run as parallel processes for improved efficiency. The two nodes have a kill-switch command topic between them to synchronize their shutdown. The plate reading node loads CNN models and weights that are hosted in the CNN Training area where they were developed. In addition, there is an inner loop controller agent that is instantiated as an object in the robot controller for navigating the inner track segment. The `vision_processing.py` file holds a library of machine vision functions that are used by all aspects of the robot controller and plate reader.

5.0 PLATE DETECTION

The plate detection and readings are accomplished by first identifying parking placards, extracting characters from the normalized image, and then processing them through two CNNs to form a prediction for the plate. See Appendix B for an overview of the data flow.

5.1 DETECTION PROCESS

The first step is to reliably find, detect, and perspective transform the parking placards which contain a parking ID and license plate information at the bottom. Detection is via an HSV filtering technique to get the white contour areas of the robot camera image. The road lines are reduced using erosion with a linear kernel. The white contours are further filtered to search for shapes that can be modeled as four-sided polygons and that do not have too narrow of an aspect ratio. The shape must also have parked car paint coloring framing it on the left and right sides, which eliminates partial plate images and other erroneous readings. See Figure 2 below.

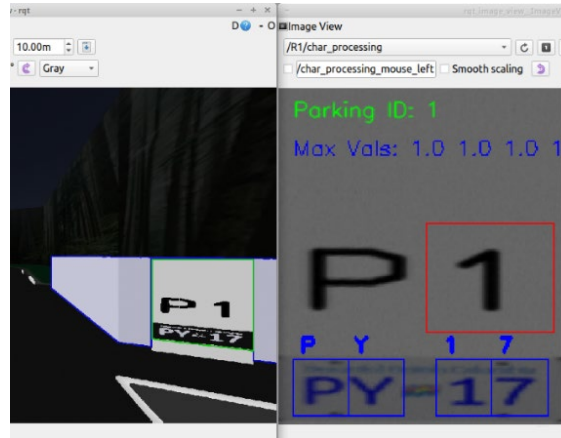


Figure 2- Placard Identified and Highlighted with Slicing Points

The upper placard is then extended downward by a fixed percentage so that it also includes the plate area shown in black, see Figure 2 above. Finally, the perspective transform provides a uniform rectangular and corrected head-on view of the plate that is then sliced into a parking ID and four plate characters that can be directed to the CNNs to form a prediction. The results are stored in a dictionary for each parking ID and the majority consensus reading is sent to the license server when a new plate reading is added.

5.2 DATA GATHERING STRATEGIES

The parking ID data was easy to acquire by implementing a button on an XBOX 360 controller to save images of the perspective transformed plates, navigating the robot under manual control. There are only 8 digits to interpret, and they are always static in position and lighting, so 1000s of images were taken by streaming photos saves while driving the robot in all angles in front of each plate.

One early strategy was to modify the competition files to spawn 8 copies of the robot, one at each plate, with their paint color modified to black to avoid distracting the placard finding algorithm. The correct spawn positions were found using the odometry topic. A bash script running overnight resets the simulation with new plates repeatedly and each robot saves a single perspective warped image of the placard it spawned at. The labeling of the image is pulled from the competition .csv file that holds the randomly generated plate characters for each parking ID.



Figure 3- Army of Robots

In earlier models the license plate generating file was also altered to generate specific examples that were causing confusion such as 'O' and 'Q'. The overnight data was not positionally diverse enough, so it was further augmented by manually capturing 52 plate types using the XBOX 360 controller. Each letter of the

alphabet is forced into each one of the two letter positions at all 8 parking locations in the plate generation script.

The final data set was captured using the fully developed autonomous robot controller to drive the course. This offered a huge advantage, because they are the exact same images expected in competition and the robot takes a very deterministic path each time. The process was automated by using the parking ID CNN output to pull a label from the .csv plate data file, along with a bash script to iterate many runs in simulation. Since full runs were down to ~30 seconds, at this point it only took a few hours of unsupervised time to gather 1000s of labeled placards. The parking IDs in thumbnails of the images were scanned by eye manually to check for misclassified data, indicated by a mismatch between labeled and actual parking ID, which could be done in only 15 minutes. The earlier confusion samples ‘O’ and ‘Q’ along with manually captured forced letter images were also added as an augmentation set, to avoid over constraining the model.

5.3 DATA PREPROCESSING

The saved placards were processed in a Jupyter notebook for each CNN, by slicing out the characters from the placard first and then resizing to a standard size. For the parking IDs the image is conditioned by converting to grayscale. The license plate characters use an HSV mask on the blue character color, which is then applied to the image to slice out the character regions, the masked portion is kept in grayscale to take full advantage of the input range on the CNN, see Figure 4, far right, below. An image processing laboratory section in Jupyter was set up to try and compare the most effective image conditioning techniques side by side.

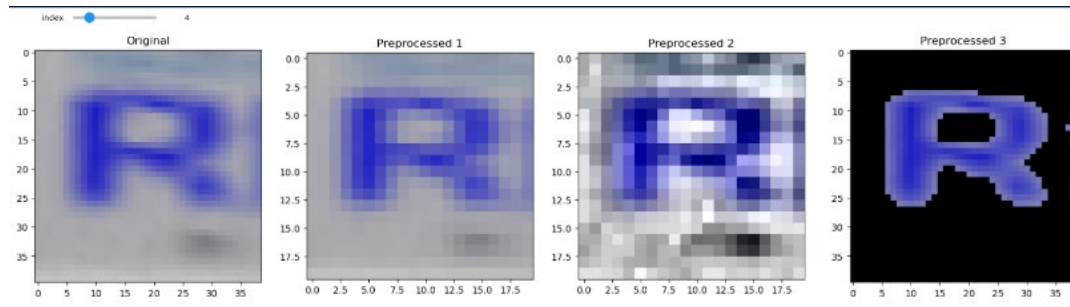


Figure 4- Data Processing Options (Rightmost Selected)

In both CNN training methods, the character samples are shuffled, and their labels are converted to a one-hot vector form which assigns a basis vector to each label type, for a total of 8 parking ID and 36 license character labels. The parking ID data distribution was not rebalanced as is standard practice, because the accuracy was found to be 100% even without rebalancing. However, the license plate recognition is a more complex task and data rebalancing was used from the start, especially since the digits are represented 2.6 times as frequently as letters. The final training set uses partially rebalanced data, where the digits are truncated, but some variety is allowed in the letter samples to preserve the extra weighting on ‘O’ and ‘Q’ samples. A bash

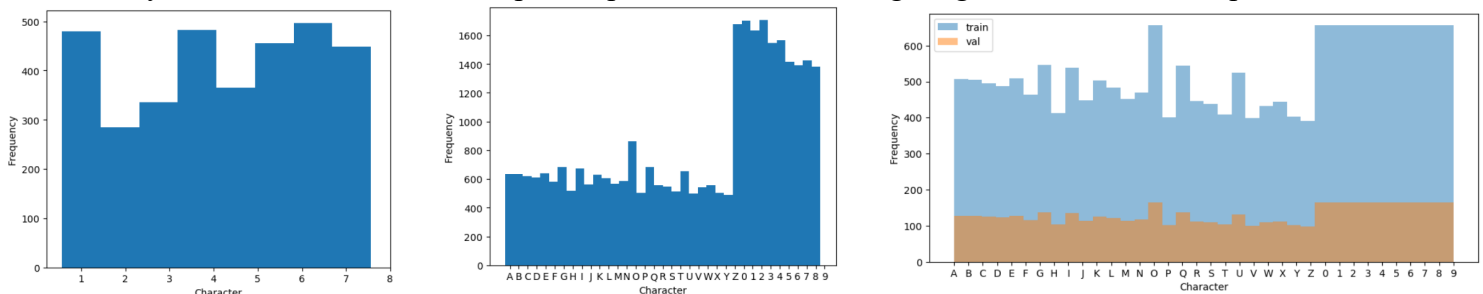


Figure 5 - (a) Parking ID Data, (b) Initial Character Data, (c) Trimmed Character Data and Distribution

script was also employed to pre-balance the data between all 8 parking locations using the filename labels in the training data directory.

A 80/20 split was used for the training and validation set, and it was decided to use the live simulation as a test set while working on robot controls. In addition, data distortion purposefully introduces Gaussian blur and random image translation to augment the training set for license plates. More emphasis is placed on horizontal translation than vertical to mimic the slicing inaccuracies seen in the competition environment.

5.4 CNN MODEL AND TRAINING

5.4.1 Architecture

The architecture for the parking IDs was simple, with only 55,000 trained parameters. The likely explanation for the success of relatively small number of parameters is that it is somewhat possible to detect the parking ID not by digit recognition, but simply given the lighting on the placard itself which varies in each parking spot but is static across all instances of Gazebo. The CNN architectures are shown in detail, see Appendix C.

Both models use sections of conv 2D followed by max-pooling which extracts low-level, then mid-level features, with consolidation and translation invariance from the max pooling. The data is flattened followed by a dropout section in preparation for dense neural network sections. The dropout randomly sets half of the parameters to zero helping to prevent overfitting, the dense layers then produce a set of 128 final indicators. In the case of license plate detection, positional information is also appended to the 128 parameters because it is very valuable information in prediction making. The first two positions are always letters and the last two are always digits. This is done by concatenating the positional 1-hot vector into the final layer before applying SoftMax to reach a prediction.

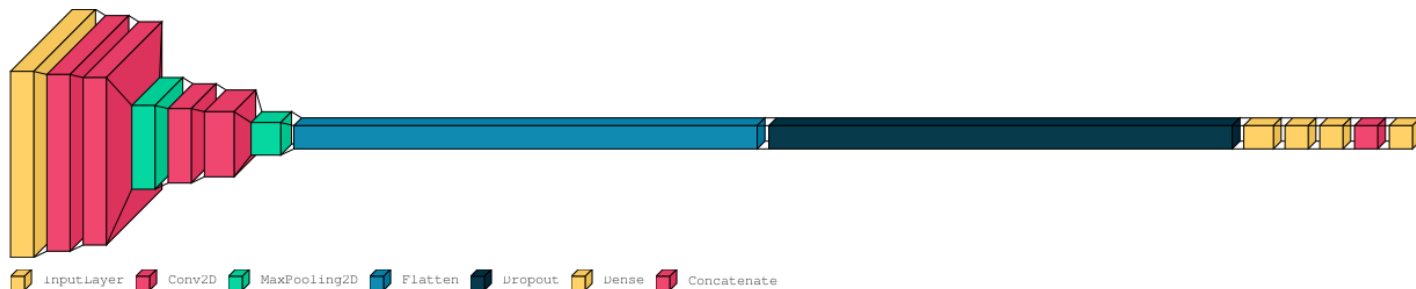


Figure 6- Plate Characters CNN

5.4.2 Training Parameters

The training parameters were kept similar to some of the earlier lab work studied in the course. The data set for the plate detection is very large, about 30,000 characters, so a slower learning rate was attempted, and gives good results. Wandb was used to track the training and the individual epoch weights were saved during training, allowing for intentional and obvious over-training. The optimal epoch of weights to use is pinpointed after training based on the data, and they are fetched from the weights folder to replace final weightings.

Model: "Parking ID CNN"	Model: "Plate Detection CNN"
"learning_rate": 1e-4,	"learning_rate": 1e-5,
"optimizer": "RMSprop",	"optimizer": "RMSprop",
"loss_function":	"loss_function":
"categorical_crossentropy",	"categorical_crossentropy",
"metrics": ["accuracy"],	"metrics": ["accuracy"],
"epochs": 140,	"epochs": 40,
"batch_size": 32,	"batch_size": 32,

5.4.2 Validation and Performance

The CNN model training performance is shown in Appendix C, for a more detailed comparison of training models explored during development, please view the Wandb report at the links below:

Parking ID Model	Plate Character Model
https://api.wandb.ai/links/gadalin/r3hhngu7	https://api.wandb.ai/links/gadalin/e78ql2gi

An epoch representing the highest accuracy in the validation set before reaching the stages of over-training is used. For the plate character recognition this corresponds to the range from 30 to 40.

5.5 MODEL TESTING AND ERROR ANALYSIS

Both the models at epoch 30 and 40 were live tested and found to be similar enough that the maximum validation accuracy of the epoch 40 is used. In actual testing it reads at over 98% accuracy based on a large set of testing runs. The only two samples of character confusion were ‘K’ and ‘X’ at P1 and ‘B’ and ‘8’ also at P1. The robot is slowed down a bit to gather more samples in this section as a solution.

The accuracy of the model is so high that it did not make sense to adapt a confusion matrix from one of the earlier labs to use on a test set. It would require generating a test set of data which would likely look almost identical to the validation set given the deterministic nature of the robot path over the course. There are many factors not captured by a confusion matrix such as the exact position and angle of a misread, these are also important discoveries to make in tuning the control model. Simple pencil and paper were used to jot down confusion items as they occurred with a description, which was enough to expose patterns. Closer examination is possible with the visual debugging tools that were built for this purpose. See Figure 2 for example.

6.0 CONTROLLER AGENTS

To facilitate moving the robot around the track, a variety of control techniques including PID and a custom machine vision algorithm are combined, with the principal division being between the outer track and inner track portion. Although the outer track has a section with faint lines and camouflage colors, the control treatment is the same as the other three sides of the perimeter track. In addition, pedestrian and truck detection systems avoid collisions while minimizing the amount of time lost waiting for track clearance.

6.1 OUTER TRACK MOTION CONTROL

The very first development of the controller agent began shortly before time-trials and was focused on the outer track, along with meeting the minimum motion requirements for passing the trial—simply moving the robot one road segment and initiating a start and a stop to the timer. I first made a tool box of abstracted basic robot commands to avoid tuning Twist() messages by hand for the ROS /cmd_vel topic.

To get the most accuracy with hard-coded sequences, the internal simulation clock topic is used as a timing mechanism, so that turn radius and distance can be controlled by velocity and time. However, the timing is never perfect and there is a set tick rate in the simulation itself which means that hard-coded motion sequences for the robot have compounding errors.

The work done with the license plates helped to develop an eye for features that could be used for realignment in turns. The robot was positioned in one of the corners to search for these features:

- The road lines and forest/ground intersections
- The parked cars
- The skyline

In addition, the texture at the end of every road segment is the same, but this forest texture repeats 8 times within the map and not just at the corners, and running image recognition like SIFT is computationally expensive making it non-ideal for rapid controls feedback.

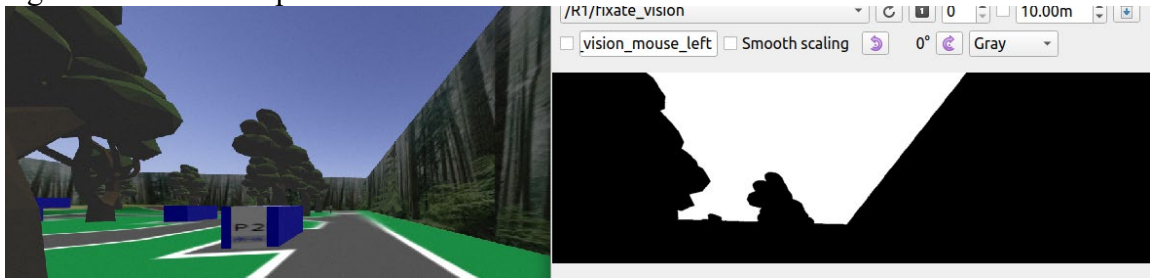


Figure 7- Prototyping SkyScanner

6.1.1 Skyscanner

After analyzing the options to explore, the sky seemed the most promising, so an HSV thresholding test was done to check the image clarity. The most distinct feature is the vertex where the two ‘forest’ panels meet in the corner. If this corner could be found then the robot could be realigned to the center of the repeated texture adjacent to it, without worrying about false matches with the texture in other locations.

The first approach to the problem using a Hough Transform with image pre-conditioning was not as successful as desired. It was hard to select the exact line, and often the line was not extending all the way down to the desired vertex. The Hough Transform is also surprisingly computationally intensive which is problematic when it comes to precision and speed. The solution was to make a custom line interpolation algorithm for this context since the line is always in the same position and the contour is not blocked by anything to the right.

The first white pixel at two predetermined heights is found by entering the image from the right. The two pixels are then interpolated into a line with a slope and the slope is followed down and to the left of the bottom point to extend the line until reaching a termination condition, which is when there are no white pixels within a threshold of the line extension. The endpoint is then marked with blue for debugging and then a fixed distance from the termination vertex is applied to get a center of road marker. The size of the green dot gives a good indication of the accuracy in pixels that is required to stay on the road, see Figure 8 below.

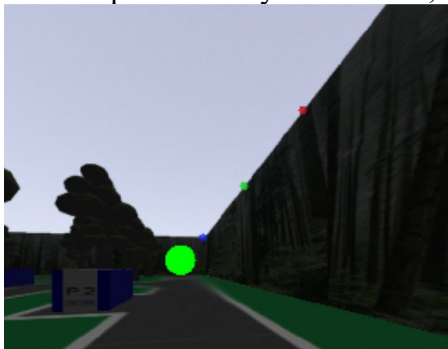


Figure 8- Fixation Marking on Competition Image

It is only necessary to align once, since the road is straight, and the robot steering runs true. This also allows for running the segments at high speed and accelerating to speeds that would cause the robot to normally flip using sequential linear velocity commands.

6.1.2 Fixate Tuning and Optimization

Initially a basic while loop was used for prototyping realignment but required turning at a slow and precise rate. The solution to optimize is with PID tuning of the controls response where the robot will turn quickly to center the green dot until it is close, then slow down. With PID the robot does not need to come out of the turns well aligned to avoid time losses. The `fixate()` function is also designed so that there is a setpoint that can be tuned, allowing the robot to fixate to the left or to the right of the center of road.

The low message rate on the `cmd_vel` topic requires that consecutive aligned readings are found before releasing the fixate algorithm. To avoid erratic behavior, a response limit is also added to the PID response. The combination of the two ensures that the green dot is centered quickly, and that the PID control is not released until it is also motionless, meaning that the robot is not oscillating or rotating. A rotating robot released from PID control can misalign itself by the time the stop command reaches `cmd_vel`.

6.1.3 Hardcode Segments and Plate Readings

The outer loop is programmed with camera sweeps near the plates. The goal is to maintain high speed while keeping within the level of precision required to avoid collisions. A challenge is that the only control feedback and correction points are at the four corners. A collision or misalignment is an event that cannot be recovered from. In retrospect this is one of the most unappealing aspects of the motion strategy, and it was also quite tedious and time consuming to derive and test all the precise sequences for the outer lap.

6.2 PEDESTRIAN DETECTION

The pedestrian crossing signals an interruption in the controller routine through image callback and increments the robot into the next state, pedestrian crossing. A simple HSV color threshold with the area of the largest red contour is enough to specify a trigger that only happens within a short distance of encountering a pedestrian crossing.

The pedestrian detection uses an HSV filter on the distinct color of the pants. The contour of the thresholded signal is greatly amplified by applying a small erosion for noise and then 30 pixels of dilation (`filterPedPants()` in [vision_processing](#)). The result is a distinct white blob moving with periodic motion.



Figure 9-Pedestrian Crossing Development

A naïve first attempt was to determine the two extrema of the periodic motion and then cross when one of them is revisited. After several iterations and improvements, the final strategy employed uses motion detection instead of centroid detection. The pedestrian is completely motionless the pedestrian is motionless, allowing for the minimal *reliable* crossing delay. The motion is read by performing a bitwise XOR between two consecutive threshold frames, then counting the number of XOR pixels to characterize degree of change.

6.3 TRANSITION TO INNER LOOP AND TRUCK DETECTION

After the fourth turn, the robot moves a predetermined distance before turning left into the inner track. Control is handed over to a truck collision avoidance system. The truck can potentially cause a 20% increase in the run time because the robot is so fast and optimized, the detection strategy advances two goals:

ENPH 353 Gadalín Robot Design

- Entering the loop while cutting off the truck, the truck is ‘at fault’ for any collision.
- Waiting for the truck to move far forward to allow for fast driving in the inner loop.

A similar strategy to that used for the pedestrian was adopted by thresholding on the black windows and tires of the truck and then quantifying the amount of thresholded pixels in the frame. To eliminate false flags in situations where it is possible to cut-off the truck, the upper right side of the frame is discarded from the signal. The threshold for advancement is set to a very low number of truck pixels in frame so that the robot waits until the truck has moved far away to the back left corner. This strategy allows for speeds that are much faster than the truck in the inner loop while avoiding collision.

Shown below are the truck, the threshold, and the blackout region of the frame that permits a ‘No Go’ signal until the back left corner.

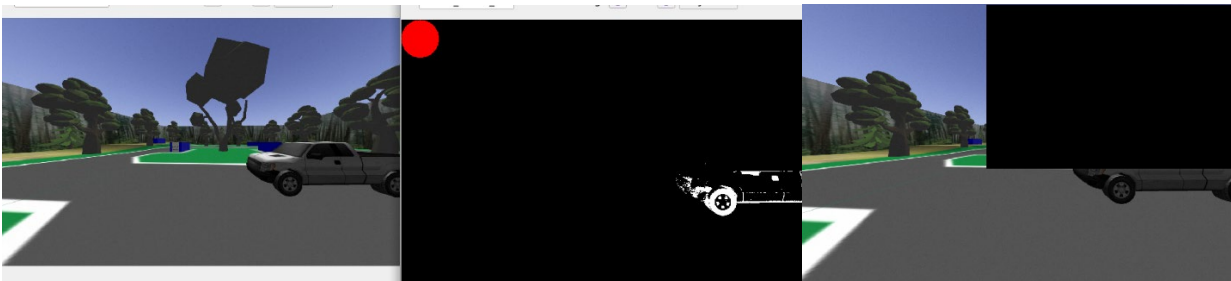


Figure 10- Truck Collision Prevention

The truck model was tested extensively by spawning the robot facing the inner loop at different positions and angles and confirming an optimal window of restriction for entry.

6.4 INNER LOOP AGENT

Once the green-light for entry is given, the inner loop agent takes over, which is instantiated as a separate class from within the robot controller. This software structure makes it much easier to debug and edit the code. An initial approach of filtering for the white border lines and parked cars did not yield a promising PID signal, so the strategy was changed to filter for the road surface. The x-position of the centroid of the largest contour is used as a numerical input into the simple-pid object which can be tuned with a setpoint, proportional, integral, and derivative parameters: <https://pypi.org/project/simple-pid/>.

Testing for viability of this signal was done using a joystick controller with the error output streaming to console. The path of least error was followed manually to get an idea of the signal conditioning, and although the inner loop exits create a deviation from the center of the inner loop, it is not enough to trigger an exit. In addition, there is some bias in the setpoint so that the controller maintains the centroid slightly to the right which adds stability. The entire raw image is thresholded and used in this manner to get good results.

The forward speed of the robot is set as constant with the PID handling the turning inputs. The tested behavior is consistent enough to hardcode a timer for reaching the final plate to force a good vantage point for plate reading. A final turn is executed, and the robot controller runs a kill switch command to the inner loop agent, followed by a stop-timer signal to the scoring system. After a pause a victory spin is encoded.

7.0 CONCLUSIONS

If I could go back and develop things differently, it would have been better to use overlapping slices for the character recognition to have better translational variance tolerance. The functional model for the plate recognition could be changed to also accept the output from the Parking ID CNN as an input in the last layer alongside the positional information. I believe concatenating this data would enhance the performance even

ENPH 353 Gadalin Robot Design

more. It would have also been more interesting develop self-training model for the outer loop, although finding and capitalizing on an exploitative feature in the simulation was a fun challenge, as it was a completely different strategy than any other student used last term.

The robot did not perform as expected in the competition due to a hardware issue. On the competition day a portable Amazon external GaN charger may have caused hardware malfunction issues with Asus drivers in the Linux environment. The laptop was completely frozen before starting the attempted first run which had never been seen before in earlier tests at home. The computer also had to be inverted in a document projector for competition display, which may have caused the internal accelerometer reading to react poorly with Linux. The computer was performing very irregular and failed to load the CNN model for plate detection within the first 5 seconds of launching the robot so that the first plate P1 was missed (no prediction made).

Post-competition, doing a hard shutdown on a frozen Linux boot and then immediately launching back into this untested configuration (inverted) caused the computer to again malfunction and not load the CNN models. Immediately before the computer froze at competition, several test runs as a 'warm-up' were at full score. A longer start-delay in the software would have prevented the issue, but the hardware itself was in an untested and deviated configuration at the time of competition. A better safety factor could have been made by gathering data on how long it takes to load the CNN models, then doubling that time for a start delay.

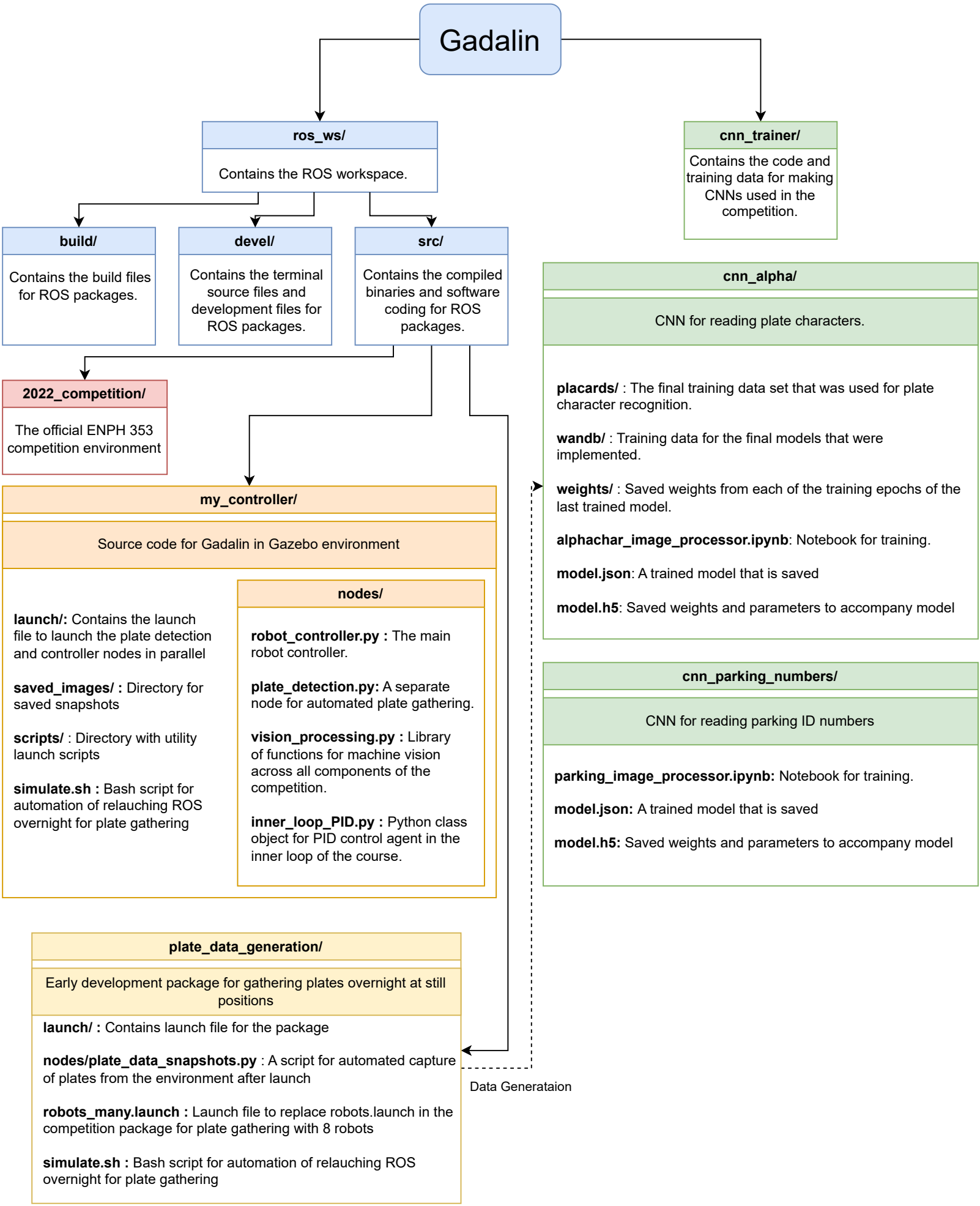
The missed P1 reading meant that only 7/8 plates were read correctly. The time of completion for traversing the course was 31 seconds which ranked third for time in the competition. It is worth noting that in past iterations of the competition the fastest time record was 42 seconds, this year was anomalous, with two teams running robots in the low-20s. The final model was able to produce 35 perfect runs in a row at an average of ~29 seconds in pre-competition testing. 5 consecutive perfect runs can be viewed at <https://www.youtube.com/watch?v=xnkjWQLoff8> A more aggressive but less tested model capable of 24 seconds was held in reserve in the speedrun branch on GitHub, but it would not have been able to beat the 23 seconds fastest time before Gadalin's heat, so the reliable model (~29 sec) was selected at competition.

With more development time, a buffer for plate data could have been used to remove the plate sweeping sequences and process plates while moving, along with other optimization strategies to further reduce the completion time. I believe that the sky scanning technique holds an advantage of not experiencing the same computational bottleneck that imitation-learning poses, and not requiring as frequent `cmd_vel` inputs. It is feasible that the completion time could be reduced to run in the low-20s with this technique.

Overall, this year's competition had several of the fastest robots ever, with a variety of technical solutions to the project challenges. It was a challenging field, and I take pride in the level of performance that the robot achieved, along with some of its innovative approaches. The course was successful in developing practical skills for applying computer vision and machine learning tools in an engineering design context, as well as managing the complexities of developing and interfacing various software tools in a Linux environment.

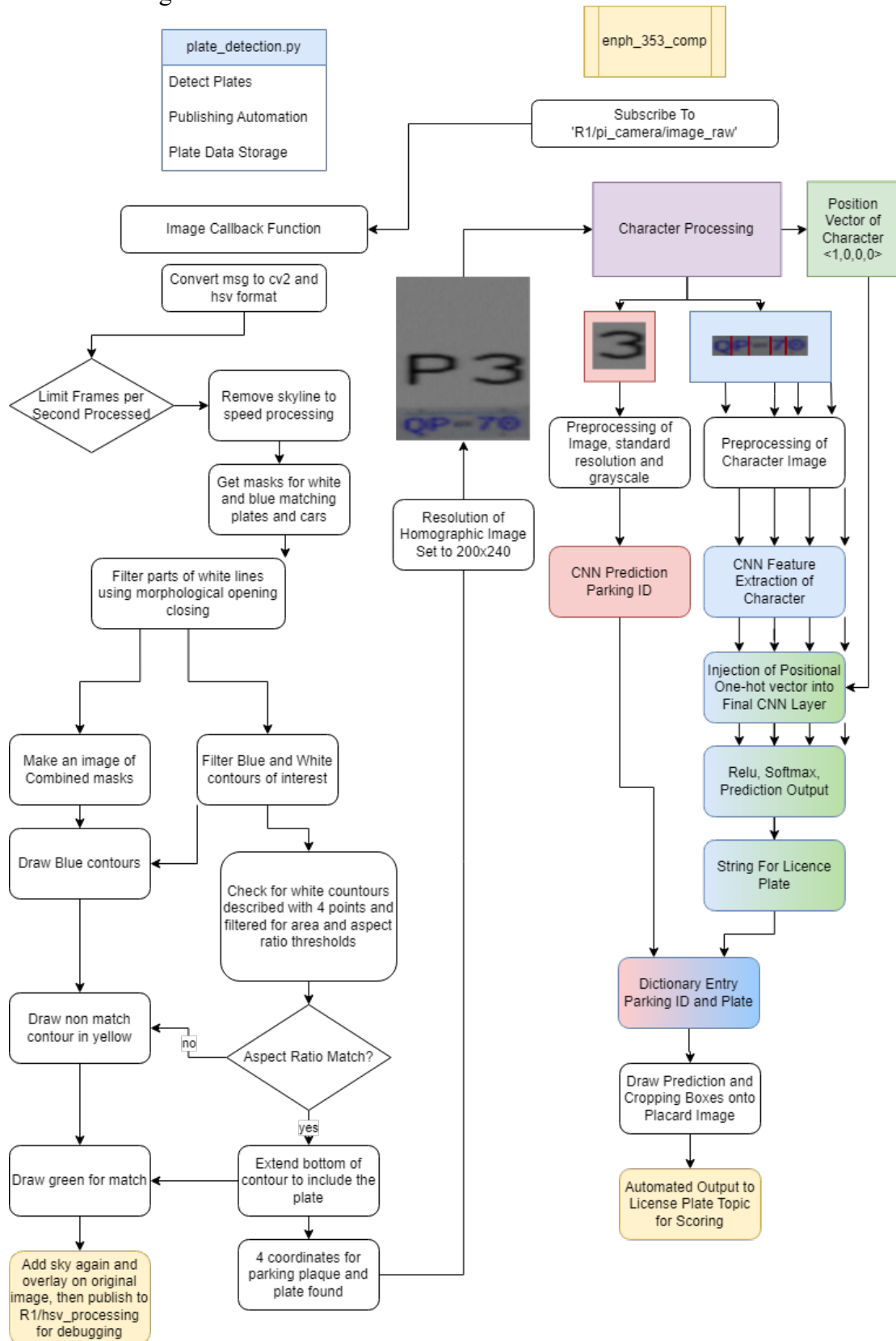
APPENDIX A

Software Structure



APPENDIX B

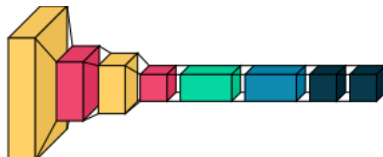





Data Flow in Plate Recognition and Reading



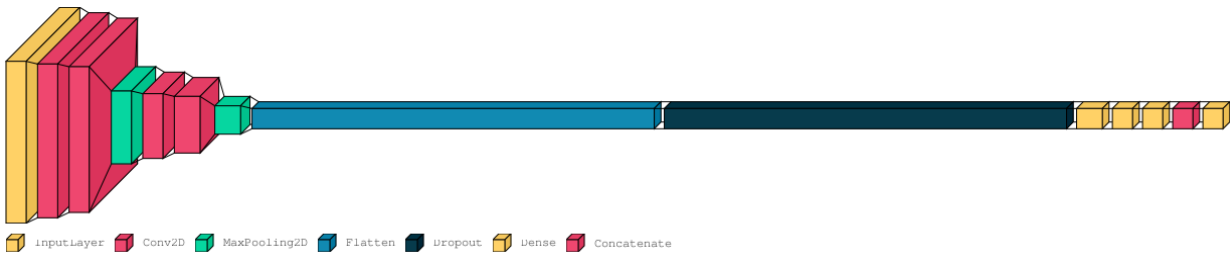
APPENDIX C

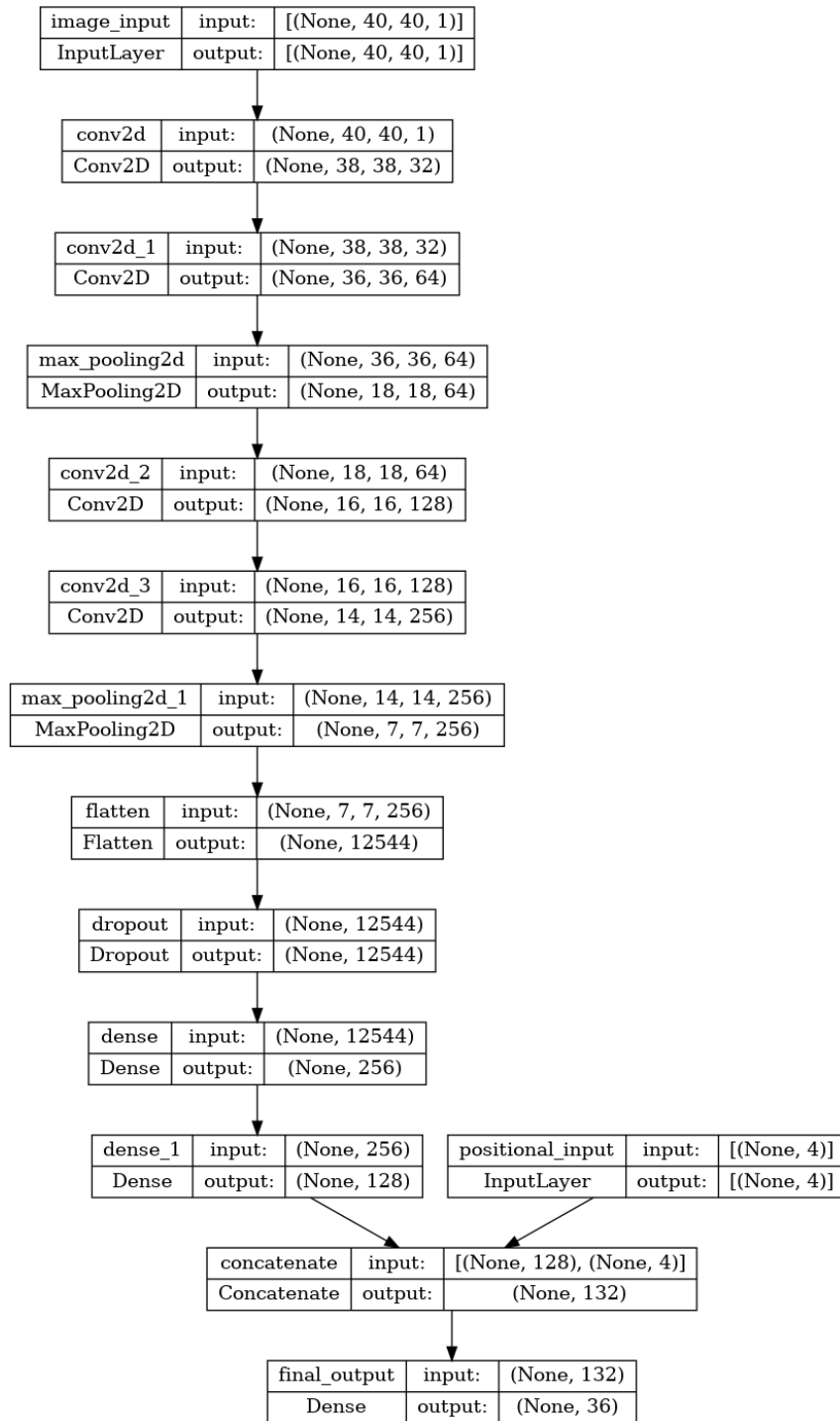
Convolutional Neural Network Models and Training

Keras Output for Model Architectures

Model: "Parking ID CNN"					
Layer (type)	Output Shape	Param #			
=====					
conv2d (Conv2D)	(None, 18, 23, 16)	160			
max_pooling2d (MaxPooling2D)	(None, 9, 11, 16)	0			
conv2d_1 (Conv2D)	(None, 7, 9, 32)	4640			
max_pooling2d_1 (MaxPooling2D)	(None, 3, 4, 32)	0			
flatten (Flatten)	(None, 384)	0			
dropout (Dropout)	(None, 384)	0			
dense (Dense)	(None, 128)	49280			
dense_1 (Dense)	(None, 8)	1032			
=====					
Total params: 55,112					
Trainable params: 55,112					
					
	Conv2D		MaxPooling2D		Flatten
	Dropout		Dense		

Model: "Plate Detection CNN"		
Layer (type)	Output Shape	Param #
=====		
image_input (InputLayer)	[(None, 40, 40, 1)]	0
conv2d_1 (Conv2D)	(None, 38, 38, 32)	320
conv2d_2 (Conv2D)	(None, 36, 36, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 18, 18, 64)	0
conv2d_3 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_4 (Conv2D)	(None, 14, 14, 256)	295168
max_pooling2d_2 (MaxPooling2D)	(None, 7, 7, 256)	0
flatten_1 (Flatten)	(None, 12544)	0
dropout_1 (Dropout)	(None, 12544)	0
dense_1 (Dense)	(None, 256)	3211520
dense_2 (Dense)	(None, 128)	32896
positional_input (InputLayer)	[(None, 4)]	0
concatenate_1 (Concatenate)	(None, 132)	0
final_output (Dense)	(None, 36)	4788
concatenate_1[0][0]		
=====		
Total params: 3,629,044		
Trainable params: 3,629,044		



Data flow in Functional Model for Plate Characters

Training Validation and Performance

