ENPH 353
Final Report

PIDestrian Hit and Run
Holden Jones 45771052
Mykal Bakker-Westeinde 96213699

Table of Contents

Overview

This report presents our solution to the ENPH 353 Parking Competition. The goal was to develop an autonomous agent that drives through the environment while obeying traffic laws and returns license plates and associated parking IDs (text quoted from the competition outline).

To summarize our approach, we implemented both a PID following and an imitation learning agent to navigate the course and used the PID follower in the competition. We identified pedestrians, the inner loop vehicle and license plates with classical colour thresholding. We created a convolutional neural network (CNN) very similar to the one used in Lab 5 and trained it on ~2500 license plates gathered from the competition environment.

1. Software Architecture

Our project structure consisted of three packages: 2020T1_competition, controller (both in the ~/ros_ws/src/ directory and CNNS, in the home directory.

2020T1_competition was the pre-made competition package. It contained the gazebo launch files, the world, robot, vehicle and pedestrian description and the score tracker app. The only thing we changed in this package was to adjust the real time factor of the simulation. When running the simulation over zoom our real time factor decreased to about 0.60. To increase it we increased the maximum step size of the simulation.

The controller package includes our PID and imitation learning controllers, as well as our plate reader. PID control is run on the pid_follower.py file. This file executes the majority of our robot's actions and logic. It subscribes to the /clock and /R1/pi_camera/image_raw topics and publishes to the /R1/cmd_vel and /license_plate topics. In it we initialize the timer, execute our line following algorithm and look for pedestrians and the inner vehicle. Imitation learning is run on the follower.py file. Its functions are identical to pid_follower.py for initializing the timer, publishing plates, avoiding pedestrians and the inner car and entering the inner loop.

To follow the competition track, *follower.py* sends the image feed to the *driver.py* file in the Driver folder with a boolean that represents whether or not our robot is in the inside loop. *driver.py* loads both the *driver_inner* and *driver_outer* models which have been trained and written to these files in CNNS. The models predict from the image feed what angular velocity commands to return to the follower.py file.

To identify and read plates, both follower and pid_follower send the entire image feed to the *reader.py* file in the Plate_reader folder. This program detects license plates, separates each plate into characters and then reads each character by the model trained by our convolutional neural network (CNN) in CNNS. *reader.py* loads the model from the *classifierbest* file.

Our CNNs both to drive and detect plates were created and trained in the CNNS package.

2. Driving

Before we started to code we needed to make a decision on how we wanted to control our robot. After using reinforcement learning in our lab we decided that it would be too complicated to implement the controller with it. Although PID (Proportional integral derivative control) following seemed to be the most straightforward approach, we wanted to get our hands dirty with machine learning. We decided to implement a PID controller that would most likely be used in the competition and an imitation learning controller for educational purposes.

2.1 PID following

Our PID controller originally started with a hardcoded turn based on timers to locate our robot in the proper position on the track. After many test runs we noticed that every once in a while the timers would activate for slightly longer or shorter than they were supposed to, leading to erratic behavior at the start of our robot's run. We thus removed timers from our hardcoded turn and had our robot drive straight until it saw the first line and then turn left until it was properly oriented.

To drive, we first applied a mask to our video feed to isolate the white lines of the track. Our original algorithm used a horizontal slice of the image, and compared the average values of the left half and the right half to determine what angular adjustments needed to be made. Our linear speed was kept constant. We figured that the operation used by this first PID follower was too ambiguous/complex for the CNN to replicate exactly. To simplify the task, we made a follower that used similar states to lab 6. We separated the right half of the horizontal slice into 12 equal vertical parts and determined which part had the most of the right line in it. We then adjusted the angular speeds to keep the white line in the same state at all times. We ended up only needing a proportional error adjustment, so our PID control was really just P control. This was a much more robust line following algorithm.

Fig 1: Masked horizontal slice showing the different states the right line can be in. The states are represented by blue lines which have been added for demonstration purposes.

After counting 6 plates, our robot switched to following the left line in a similar way to enter the inner loop. To cross the first opening in the inner loop, once our robot detected the 7th car directly to its right it would drive straight for 2 seconds before following the left line once more. The car was detected using a pixel threshold on a masked image that isolated only parked cars. For the second opening of the inner loop, once 4 seconds had passed from the time it crossed the first opening our robot would start following the right line before switching back to the left line (from another timer) to complete the course.

Although our PID control wasn't as sleek as an imitation controller and used a lot of code, in this controlled competition environment our PID follower was very robust and could run extremely fast. We could operate the robot up to 0.5 linear speed, although we ran it at 0.4 in the competition as plate detection was more reliable at slower speeds.

2.2 Imitation Learning

After getting a PID follower to reliably navigate the course, we decided to train a CNN to imitate the PID follower's movements. We saved every frame from the video feed, and labelled each image with the command made by the PID controller at the given frame. Data collection was quick because of the automatic labelling, and after 2 laps of the outer loop we gathered around 2000 images.The CNN is similar to the one used for character recognition.

With this data, we were able to train a model to follow the outer loop (nearly) as well as our PID follower.

Once we had a follower for the inner loop, we tried to use it to train a second imitation learner. The follower we made relied on timers, and it drove differently depending on the current state/time of the robot. This information of state couldn't be conveyed through a single frame, so identical images were sometimes labelled differently, and the CNN didn't learn very well. To get a better dataset we needed a simpler follower, so we made a follower to follow the road's center of mass. Training a CNN using the data from this follower was much better, and we were able to follow well around the inner loop, although at a slow speed (0.15 vs 0.3 linear speed).

After we were done training both networks for the outer and inner loop, we compared their performance to our PID followers', and found that the imitation learning was not as fast as the "expert" that trained it, and decided to keep using a PID controller. The main advantage we saw to using an imitation learner was the small amount of code required.

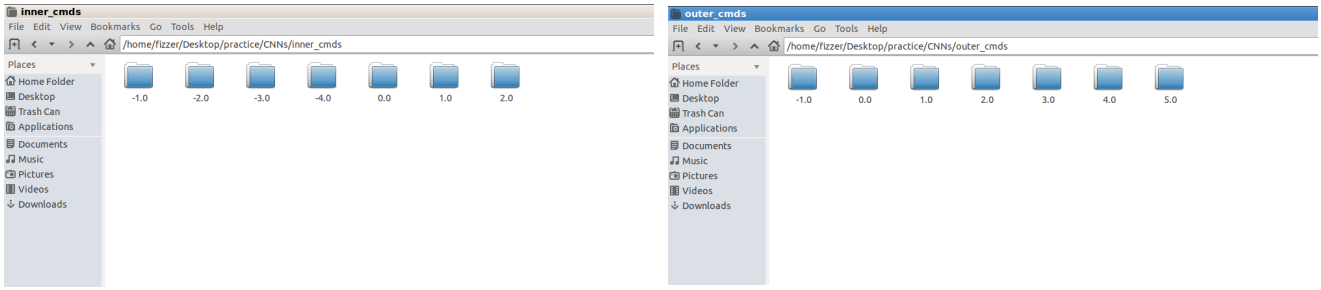

Fig 2: File system used to store training data for imitation learning. Each folder contains up to 2000 frames from the agents POV, in a state that corresponds to the folder name. For example, the folder "0.0" contains images of zero-error states, which are straight-line movements. Notice how the motion of our agent can be reduced to the classification of seven unique states, which is a simple task for a neural network. We found that increasing the number of states beyond this point did not improve the agent's driving.

2.3 Pedestrian Avoidance

To avoid casualties, we needed a quick way of detecting the locations of pedestrians. A crosswalk is an obvious cue to look for pedestrians, and because of it's red colour it was easier to detect. We constantly looked for the crosswalk, and when we saw it we began looking for the blue pants of the pedestrian. If we couldn't detect a pedestrian and the crosswalk in the same frame, we decided it was safe to cross. We sped up our agent for the crossing to minimize the chance of hitting a pedestrian.

We had to hard code the PID follower to move straight during the crossing, so it doesn't get thrown off by the white road lines. In contrast, the imitation learner was trained to move straight through the crosswalk, and it required no extra code.

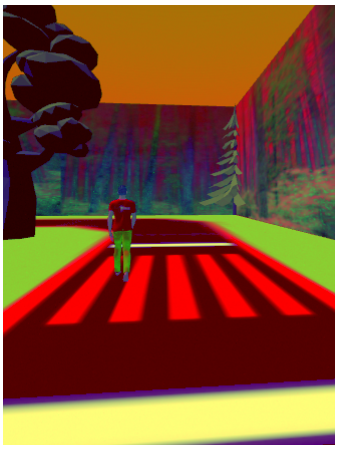

Fig 3a : HSV pedestrian image

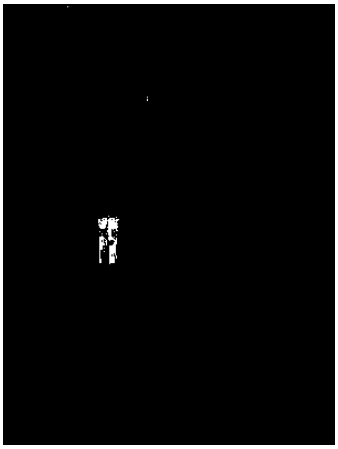

Fig 3b : thresholded pedestrian image

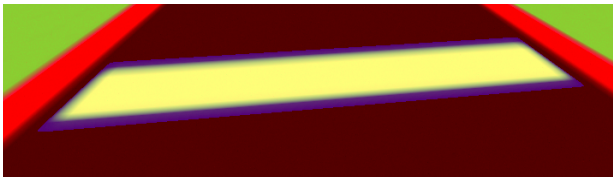

Fig 4a: HSV crosswalk image

Fig 4b: thresholded crosswalk image

2.4 Inner Loop Vehicle Avoidance

To avoid the truck in the inner loop we used a very similar technique to the one we used in pedestrian avoidance. When driving in the inner loop we masked our video feed to detect the grey of the truck. Whenever the amount of pixels in the mask directly in front of our robot reached a certain threshold, it would stop for three seconds before resuming.

Having to stop for the truck in the inner loop would increase our run time significantly. After doing some tests, we realized that we could avoid stopping for the truck at all. By timing the start to coincide with when the truck was on the upper left side of the inner track we were able to avoid meeting the truck 100% of the time, allowing our run times to consistently clock under 45 seconds.

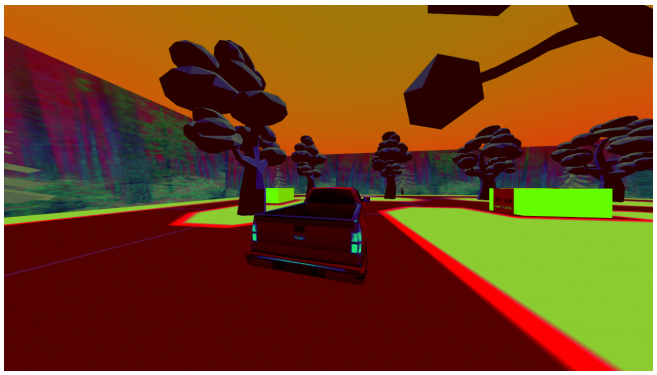

Fig 5a: HSV truck image

Fig 5b: Thresholded truck image

3. License Plate Reading

Successful plate recognition counted for 52 of 57 possible points available for the competition. We trained a CNN to classify letters and digits, using mainly images collected from the competition environment. Plate classification is simple with quality data, and the most challenging task was to reliably detect/identify license plates, before reading their content.

3.1 Plate Identifier

After implementing SIFT object detection in a lab, we were ready to try detecting license plates using a similar approach. We drove around manually, screenshotted license plates and reused lab code to get a homography on the license plates. Even in the static environment of Google Colaboratory, it was difficult to get a complete homography on the license plates from different angles, and we decided to try a simpler method.

Our simple approach to plate detection was to threshold to the blue colour of the plates, and draw a box around everything blue in the frame. The car and sky are also blue, but by only looking for blue objects that fit in a box with pre-set dimensions, we were able to eliminate that ambiguity. To further refine the detection, we only looked for plates near the bottom corners of the frame; this window changed when the robot entered the inner loop. With these strict conditions, the plate detection was near perfect and the inputs to our classifier were clear and well framed.

```
64
65          for cnt in contours:
66              (x,y,w,h) = cv2.boundingRect(cnt)
67              if h in range(20,60) and w in range(w1,170) and x in range(x1,x2) and y in range(y1,y2):
68
```

Fig 6: Code snippet for plate detection. We only looked for images of a certain width and height (w,h), in a specific location of the image (x,y).

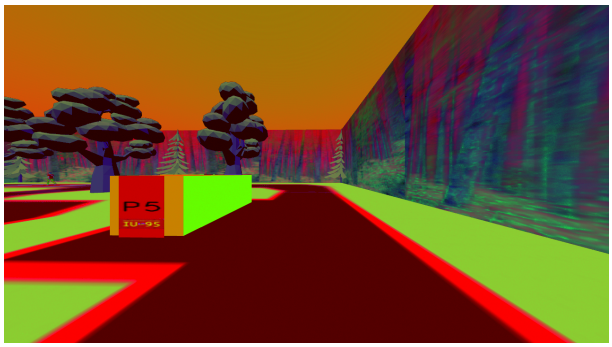

Fig 7: Hsv image of upcoming plate.

Fig 8: Thresholded image of upcoming plate.

3.2 Plate Classifier

After isolating the plates, we trained a network almost identical to the one used in Lab 5 to read the characters. To generate data we originally adapted the license plate generation script to produce plates with the same font as the ones in the competition. We then applied random gaussian blurring and shifting to these plates and with this data trained our first network. With this first CNN we were able to read ~50% of license plates successfully. If a letter was detected where a number should be, we hardcoded a change from letters to similar numbers and vice versa. For example if an 0 was read as the first letter we changed it to O, or 1 to I. If the misplaced letter/number did not have a similar number, we used the network's second best prediction.

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 28, 32)	2528
max_pooling1d (MaxPooling1D)	(None, 14, 32)	0
conv1d_1 (Conv1D)	(None, 12, 128)	12416
max_pooling1d_1 (MaxPooling1	(None, 6, 128)	0
conv1d_2 (Conv1D)	(None, 4, 1024)	394240
max_pooling1d_2 (MaxPooling1	(None, 2, 1024)	0
flatten (Flatten)	(None, 2048)	0
dropout (Dropout)	(None, 2048)	0
dense (Dense)	(None, 512)	1049088
dense_1 (Dense)	(None, 36)	18468

Total params: 1,476,740
Trainable params: 1,476,740
Non-trainable params: 0

Fig 9: CNN architecture. Our training script first separated all plates into separate characters, then assigned each character a one-hot encoded value (0 for A, 25 for Z, 26 for 0, 35 for 10, etc.). Our neural network was sequential and had two convolutional and max pooling layers followed by two dense layers. The last 1x36 layer outputted the one-hot encoded prediction.

By saving each plate image whenever it was detected, every time we ran our controller we gathered more plates directly from the competition environment. After some time we collected ~500 plates and trained a new network with exclusively these images. With this CNN we were

able to read ~80% of license plates. The more we ran our controller, the more data we generated and the better our networks performed. Our final network was trained on ~1500 plates from the competition environment. We also applied a random gaussian blur to 1000 of the plates to bring our total training set to ~2500 plates. With this we were able to read almost all plates that weren't cut off perfectly.

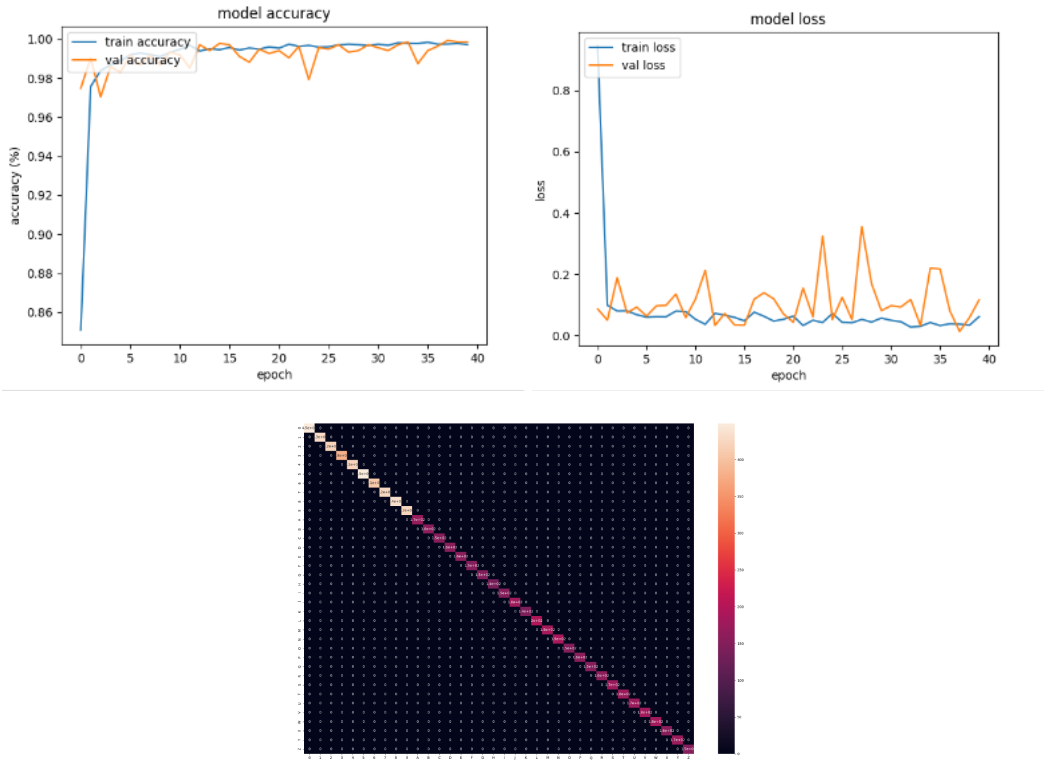

Fig 10: Model accuracy, model loss and confusion matrix plotting the predicted characters vs. actual characters of our input data. We didn't use a test set, rather tested it out by driving. Our model was trained over 40 epochs on 2500 plates. Due to there being only 10 numbers and 26 letters, note how much more training data was collected for the numbers. Our original CNNs trained on a smaller data set were almost perfect at number identification but much worse at letter identification. By training the CNN on more data this was fixed.

Our plate identifier was able to return 2-8 images of each plate to the classifier. Some were harder to read than others, and some had letters cut off entirely. Our agent would read each plate and then publish the one it had most confidence in. This confidence was sometimes misplaced though and the neural network would predict both the correct and incorrect plates with 100% confidence. We noticed that most of the incorrect answers were from plates that had been cut off or taken from far away. We thus manually lowered the confidence of such plates.

To return license plates associated with the right parking IDs, we considered creating another detector and reader for the stalls. However since the stalls always were in the same order, we simplified the task using a counter and incremented the plate number for every plate we

published. This strategy would fail if any of the plates were missed but through over 200 test runs without this occurring we were confident our robot would not fail to report a plate.

Fig 11a: A generated plate

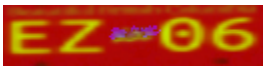

Fig 11b: A real, collected plate

4. Testing

After time trials once our robot was capable of completing the entire course, we started testing and documenting its performance to iron out any bugs. We tested 250 runs and made 15 major changes. By the last change, our robot completed 39 runs with 95% perfect runs and an average time of 43 seconds. Our test runs are documented in the appendix below.

5. Appendix

Run #	Completed(Y/N)	Error	Score	Time (blank if not completed)	Notes	Zoom Share(Y/N)
1	Y		57	45		N
2	Y		57	54		N
3	Y		49	60	Read 6 instead of 5 on last plate	N
Change: Only starting robot when car is on other side of loop						N
4	N	Read first plate as plate #3, all plates after wrong and entered loop early				N
Change: Increased the time between plates to 0.85 at 0.4 speed						N
5	Y		57	49		N
6	N	Ran into first car				N
Change: Stopped robot turning hard left in first 3 seconds						N
7	Y		51	44	Min uncertainty changed a good plate to a bad one	N
Change: If the plate is cut off (x = 0) made min_uncertainty 0.5						N
8	Y		57	39		N
9	Y		57	38		N
10	Y		57	37		N
11	N	Start messed up				N
12	Y		57	42		N
13	N	Start messed up				N
14	Y		57	39		N
Change: Removed timer from the start						N
15	Y		57	43		N
16	Y		57	40		N
17	Y		57	39		N
18	N	Read first plate twice				N
Change: Increased plate_time gap to 1						N

19	Y		57	56	Didn't start at right time	N
20	Y		49	43	Read last plate wrong, plate cut off	Y
21	Y		57	42		Y
22	Y		57	42		Y
23	Y		57	40		Y
24	Y		57	39		Y
25	Y		57	40		Y
26	Y		57	42		Y
Change: Increased speed to 0.45						Y
27	Y		51	37	Read 6 instead of 5	Y
28	Y		57	37		Y
29	Y		37	45	Missed two plates, sticking to 0.4 speed from now on	Y
Speed back to 0.4						
30	Y		57	41		Y
31	Y		57	44		N
32	Y		57	39		N
33	Y		57	43		N
34	Y		57	43		N
35	Y		45	39	Read K instead of X and 1 instead of 5	N
36	Y		49	39	Last plate cut off	N
37	Y		57	39		N
38	Y		57	40		N
39	Y		57	40		N
40	Y		57	45		N
41	Y		57	39		N
42	Y		57	43		N
43	Y		57	41		N
44	Y		57	41		N
45	Y		51	40	Read JR15 as JR11	N
46	Y		57	45		N
47	Y		51	39	Plate cut off	N

48	Y		57	42		N
49	Y		57	49		N
50	Y		57	43		N
51	Y		51	40	VT90 instead of VT98	N
52	Y		57	42		N
53	Y		51	39	PZ55 istead of OZ55	N
Change: New classifier (classifier3)						N
54	Y		57	42		N
55	Y		57	39		N
56	Y		57	39		N
57	Y		57	43		N
58	Y		57	42		N
59	Y		51	43	PF42 instead of BF42	N
60	Y		49	39	BM41 instead of BH41	N
61	Y		57	43		N
62	Y		57	39		N
63	Y		57	43		N
64	Y		57	45		N
65	Y		57	39		N
66	Y		57	41		N
67	Y		57	40		N
68	Y		57	39		N
69	Y		57	41		N
70	Y		57	44		N
71	Y		57	44		N
72	Y		57	42		N
73	Y		57	42		N
74	Y		57	40		N
75	Y		57	39		N
76	Y		51	39	MS95 instead of HS95	N
77	Y		57	44		N
Change: New classifier (classifier4)						N
78	Y		57	42		N
79	Y		57	39		N
80	N	Inner loop cut left too soon				N

Made right following time from 5.5 to 6						N
81	Y		51	39	QB58 instead of 56	N
82	Y		51	41	YV88 instead of 86	N
83	Y		51	42	MO38 instead of 36	N
Change: New classifier (classifiernew)						N
84	Y		57	39		N
85	Y		57	41		N
86	Y		57	41		N
87	Y		57	44		N
88	Y		57	38		N
89	Y		57	39		N
90	Y		49	39	OT93 instead of DT93	N
91	Y		57	40		N
92	Y		57	40		N
93	Y		57	42		N
94	Y		57	39		N
95	Y		57	42		N
96	Y		51	40	Y instead of V	N
97	Y		51	48	Y instead of V	N
98	Y		57	39		N
99	Y		57	39		N
100	Y		57	58	Stuck behing car	N
101	Y		57	40		N
102	N	Did not turn at start	0			N
103	Y		57	46		N
104	Y		51	45	L instead of I	N
105	Y		51	40	GG19 instead of FI59	N
106	Y		57	42		N
107	Y		51	40	R instead of A	N
108	Y		45	40	Y instead of N, R instead of A	N
109	Y		51	40	Y instead of V	N
110	Y		57	51		N
111	Y		57	39		N
112	Y		57	42		N

113	Y		57	41		N
114	Y		57	40		N
115	Y		57	46	sketchy crosswalk (veered a bit to left)	N
116	Y		57	40		N
117	Y		57	40		N
118	Y		57	44		N
119	Y		49	40	QK33 instead of 31	N
120	Y		57	66	COmputer almost dead so super laggy	N
121	Y		57	39		N
122	Y		57	42		N
123	N	Messed up start			Changed start to 0.2 speed	N
124	Y		57	39		N
125	Y		57	41		N
126	Y		57	53		N
127	Y		51	41	Y2 instead of QY	N
128	Y		57	42		N
129	Y		57	45		N
130	Y		57	40		N
131	Y		57	41		N
132	Y		51	40	K instead of X	N
133	Y		57	45		N
134	Y		57	45		N
135	Y		57	45		N
136	Y		57	45		N
137	Y		57	45		N
138	Y		57	40		N
139	Y		57	40		Y
140	Y		45	40	K for X, 3 for 7	Y
141	Y		57	55		Y
142	Y		51	45	K instead of E	Y
143	Y		57	40		Y
144	Y		57	53		Y

145	Y		57	43		Y
146	Y		57	55		Y
147	Y		57	40		Y
148	Y		57	38	Fastest possible run with 0.45 speed up	Y
149	Y		51	53	8 instead of 6	Y
150	Y		57	43		Y
151	Y		57	43		Y
152	Y		51	51	DBD9 instead of MD49	Y
153	Y		57	41		Y
154	Y		57	43		Y
155	Y		51	43	RJH1 instead QR34	Y
Change: New classifier with blurred images, and min uncertainty of 0.95 for blurred images						
156	Y		57	44		N
157	Y		57	43		N
158	Y		57	41		N
159	Y		45	44	MN61 instead of PA61 (wrong), SNH9 instead of QG70 (cut off)	N
160	N				saw vehicle coming into loop, went right	
Change: Classifier5 and removed inner_start+3						
161	Y		57	41		N
162	Y		57	43		N
163	Y		57	43		N
164	Y		57	62	testing vehicle avoidance in inner loop	N
165	Y		51	62	Plate cut off	N
166	Y		57	60		N
167	Y		57	43		N
168	Y		51	43	Plate cut off	N
169	Y		49	38	Plate cut off	N
170	Y		57	42		N
171	Y		57	42		N

172	Y		57	46		N
173	Y		57	39		N
174	Y		57	59	stuck behind car	N
175	Y		57	42		N
176	N				Hit pedestrian I think? I wasn't looking	N
177	Y		57	41		N
178	Y		57	42		N
179	Y		57	42		N
180	Y		57	42		N
181	Y		57	42		N
182	Y		57	46		N
183	Y		57	42		N
184	Y		57	42		N
185	Y		57	63		N
186	Y		57	42		N
187	Y		51	40	Plate cut off	N
188	Y		57	41		N
189	Y		57	42		N
190	Y		51	45	VD03 instead of VO03	N
191	Y		57	40		N
192	Y		57	43		N
193	Y		57	43		N
194	Y		57	42		N
195	Y		49	63	Stuck behind car	N
196	Y		57	44		N
197	Y		57	39		N
198	Y		57	39		N
199	Y		57	41		N
200	Y		51	40	R instead of W	N
201	Y		51	40	O instead of U	N
202	Y		57	39		N
203	Y		57	43		N
204	N	Did not detect crosswalk in time and veered off.				N

205	Y		57	43		N
206	Y		51	40		N
207	Y		49	70	Stopped behind car right at the license plate, and didnt even detect the plate.	N
208	Y		57	49		N
209	Y		57	42		N
210	Y		57	43		N
211	Y		57	42		N
Change: Reversed last change, changed outer_plates so plate 1 is being read many times						
212	Y		57	45		N
213	Y		57	42		N
214	Y		57	43		N
215	Y		57	44		N
216	Y		57	41		N
217	Y		57	43		N
218	Y		57	42		N
219	Y		57	40		N
220	Y		57	40		N
221	Y		57	43		N
222	Y		57	45		N
223	Y		57	43		N
224	Y		57	44		N
225	Y		57	40		N
226	Y		57	42		N
227	Y		57	44		N
228	Y		57	41		N
229	Y		57	40		N
230	Y		57	41		N
231	Y		57	42		N
232	Y		57	41		N
233	Y		57	41		N
234	Y		57	39		N
235	Y		57	43		N

236	Y		57	41		N
237	Y		57	39		N
238	Y		57	41		N
239	Y		57	43		N
240	Y		57	41		N
241	Y		57	45		N
242	Y		51	44	Plate cut off	N
243	Y		57	71		N
244	Y		57			Y
245	Y		57			Y
246	Y		57			Y
247	Y		57			Y
248	Y		57			Y
249	Y		51		Last plate wrong	Y
250	Y		57			Y
TOTAL RUNS	250					
TOTAL COMPLETE	239					
TOTAL PERFECT	194					
SINCE LAST UPDATE						
TOTAL RUNS	39					
TOTAL COMPLETE	39					
INCOMPLETE	0					
TOTAL PERFECT	37					
AVERAGE TIME	43					
PERCENTAGE PERFECT	94.87 %					