THE CANADIAN SCIENCE FAIR JOURNAL

ARTICLE

Development of a Level 2 Autonomous Vehicle Using Convolutional Neural Networks and Reinforcement Learning

By Brendon Matusch

Age 14 | Sudbury, Ontario

$2000 Scholarship to EU Contest for Young Scientists | Youth Can Innovate Award - Intermediate | Challenge Award - Intermediate | Excellence Award - Intermediate Gold Medal | $4000 Western University Scholarship Canada-Wide Science Fair Best Project Award ($2500)

One of the most persistent themes in science fiction, for many years, has been that of the self-driving car. Over the last 5 years, advances in machine learning and the advent of extremely powerful, low-cost graphics processing units (GPUs) have begun to make this dream a reality. It promises to make our roads much safer and more efficient.

Unfortunately, most of this research is proprietary, done inside companies such as Google and Tesla. While others would benefit from, and could contribute to these technologies, it is difficult for a newcomer to the field to know the best path to developing an autonomous vehicle. My goal is to test different techniques, based on convolutional and recurrent neural networks and reinforcement learning, to gain valuable knowledge on the most reliable, safe, and efficient techniques (both original and existing) for autonomous driving.

INTRODUCTION

I have built on a set of general machine learning concepts and some previously published results, extending them and combining them with new innovations to develop a self-driving vehicle. Each of the resulting techniques were tested, and performance was compared, both in a simulation and a physical vehicle. My purpose is to explore and understand the relative merits of these machine learning techniques.

My work primarily focuses on the steering system, utilizing neural network-based vision processing alongside conventional and machine learning techniques for path planning. A secondary goal was to produce a complete level 2 autonomous vehicle, which can handle steering as well as speed control on its own, provided a human is available as a fallback to ensure safety. To that end, I developed systems for stop sign detection and location of vehicles ahead to control speed. The steering and speed control systems were interfaced to a computer-simulated vehicle as well as a physical vehicle. Functionality and performance were tested in both environments.

MATERIALS AND METHODS

Simulation Development and Validation

I created a simulation using Unity and C#. A track several kilometers in length was constructed with prefabricated 3D highway segments. It included a section very similar to the road used in the real world – a relatively long, straight section, followed by a moderate curve to the left (Figure I1).

The vehicle was created out of geometric components with physical proportions very similar to those of the real-world vehicle, including mass, centre of gravity, wheelbase, width, and camera position. Mechanical imperfections were simulated, including the dead band in the vehicle's steering system caused by gaps between the steering gears and imperfect fits in the steering linkages, the vehicle's tendency to veer toward the ditch due to road convexity, and the reduced frame rate observed in the real vehicle because of computing hardware limitations.

Once the physical vehicle was complete and steering systems were tested in the real world, I validated the simulation's performance relative to the real-world vehicle, using identical steering techniques and parameters. To achieve similar performance, the

steering dead band was found to be essential – before it was introduced, the vehicle's path was unrealistically smooth. I tested all techniques except one (3.1) in this simulation.

Physical Vehicle Construction and Testing

I constructed a vehicle using a low-cost electric go-cart as a base. For steering control, a servo motor was attached to the steering column via a belt drive. It was operated by a CTR Electronics Talon SRX motor controller. A planetary gearbox was attached to the motor, including a quadrature encoder to facilitate positional control. A plastic box containing most of the electronics was mounted on the back of the vehicle, including the roboRIO main device controller, which interfaced with the motor controller via CAN bus and with the vision processing computer via Ethernet. See Figures I2 and Figure I3.

Sensors included a camera and a LIDAR system. I mounted a Logitech C920 webcam above and behind the driver, and connected it to the computer, which handled vision tasks, calculated a steering angle, and sent it to the roboRIO. The computer also interfaced with the Scanse Sweep LIDAR sensor, which was mounted at the extreme front, so it had a wide field of view of the road ahead of the vehicle.

I performed a series zof experiments on the vehicle to quantify the physical characteristics of its steering system. First, the correlation between the angle of the steering wheel and the angle of the drive wheels was determined through a series of trials in which the steering wheel was rotated to specific angles and the horizontal position measured at the end of a ruler parallel to the drive wheels. I found the steering angle to be linearly correlated with the position of the end of the ruler, meaning that the wheel angle could be approximated as arctan(ax) with a constant a of 1.032. See Figure G1.

I conducted a related experiment to determine the backlash in the steering system. I instructed the steering motor in software to approach a specific angle from the left, and then from the right. The difference between the two resulting wheel angles represents the dead band. This was done several times with different steering angles, and the dead band was found to be 0.57 degrees. See Figure G2.

I tested techniques 1, 2.1A, 2.2A and 3.1 in the field with this vehicle. Due to winter road conditions, the remaining techniques had to be tested in the performance-validated simulation.

Steering System Software Development

The steering system consists of components for vision (breaking the image down into a usable data format) and path planning (generating a steering angle accordingly). I developed and tested Technique 1 for combined vision and path planning. Two (2.1A, B) were tested for lane detection, and four (2.2A, B, C, D) for path planning.

Technique 1 – Combined End to End CNN

This was initially based on Nvidia's paper "End to End Learning for Self-Driving Cars",[1] in which a convolutional neural network (CNN) is used to calculate steering angles based on road images. It is trained on a dataset of images of the road ahead of the vehicle, alongside corresponding steering angles. I recorded these images during several data collection runs, both in the Unity simulation and in the real world. I conducted real-world runs at different times of

Figure I1. Screenshot of Simulation.

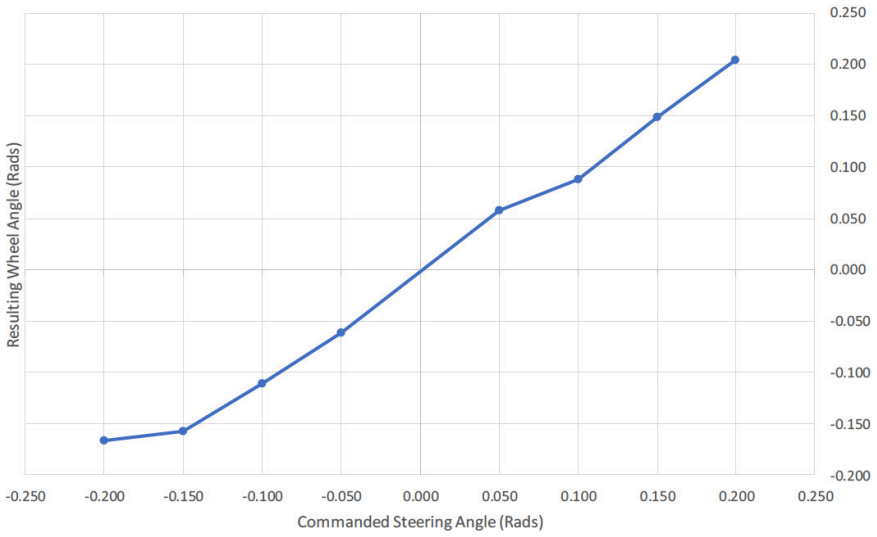

Figure G1. Correspondence between commanded steering angle and resulting wheel angle.

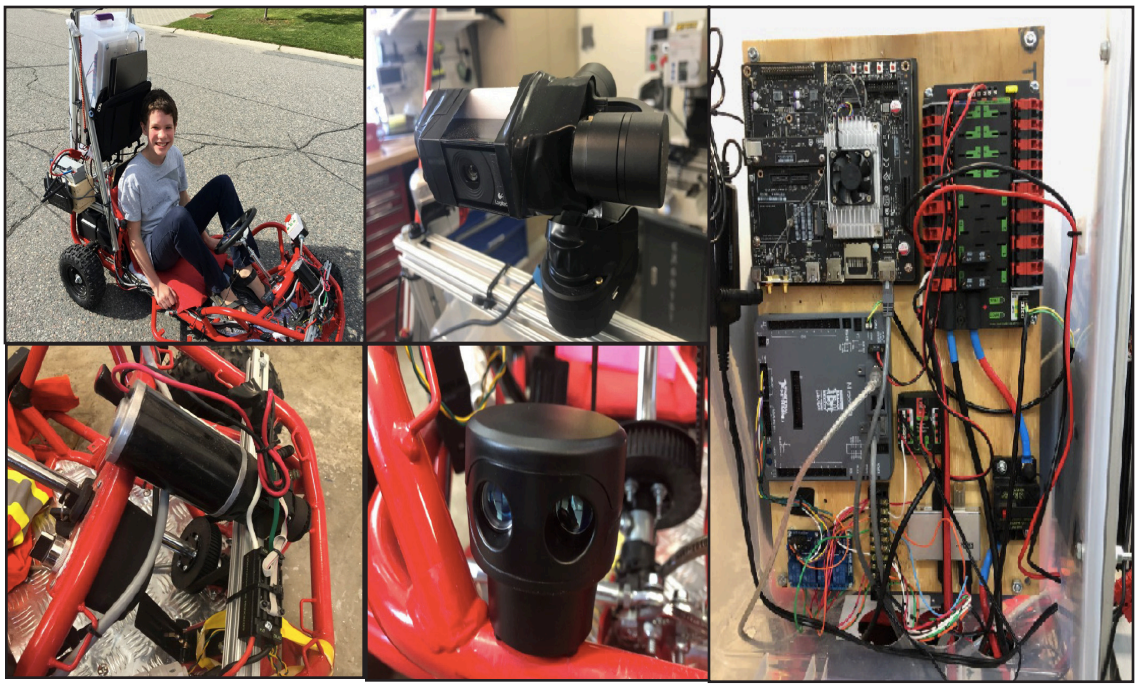

Figure I2. Physical Vehicle Images. (a) Overview, (b) camera and mount, (c) steering system, (d) LIDAR sensor, and (e) electronics board.

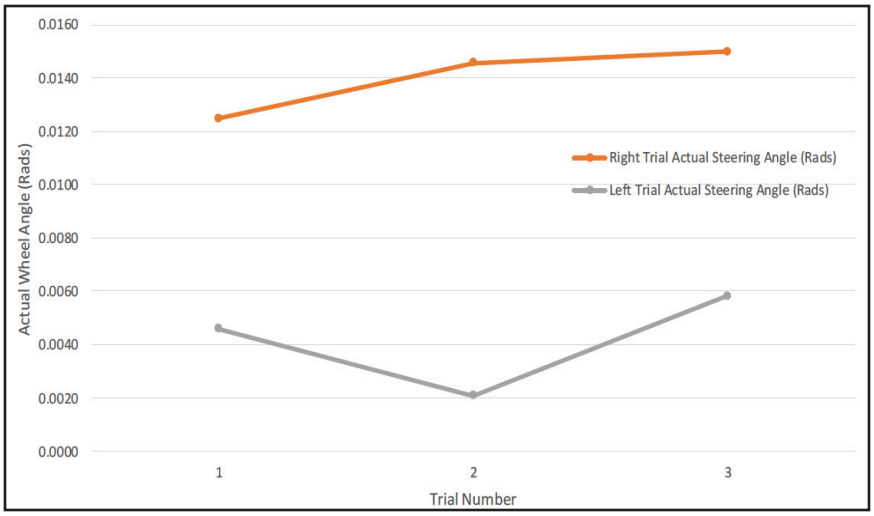

Figure G2. Steering backlash measurement by approaching angle from opposite directions.

the day and in varying weather conditions to ensure a representative set of images.

I built the neural network for this project using the Keras[2] deep learning framework. One network architecture based on the Nvidia paper was developed, as well as several entirely custom models. The hyperparameters were heavily modified over 37 rounds of optimization, using different numbers of convolutional and dense layers, numbers of convolutional filters, kernel sizes, and settings for batch normalization, activation functions, batch sizes, and optimizers.

I conducted this testing systematically in four distinct series, starting by modifying one parameter at a time from the original model, then trying combinations of changes that improved performance, in increasing number with each round. With every model tested, I recorded the training and validation losses and the ratio between them and used it as the main metric by which to evaluate their performance.

To improve the stability of this algorithm and reduce the magnitude of spikes (which damage mechanical gearboxes), I developed and applied a low-pass filter. It functioned by calculating a rolling weighted average of the last n outputs. Each output is multiplied by a weight, and these weighted outputs are totaled and divided by the sum of the n weights. I tested several combinations of weights in the simulation.

I measured performance of the 14 testing runs completed in the simulation using the standard deviation from the lane centre. This was calculated using a series of spatial points located along the centre of the lane throughout the entire track. I developed an algorithm in which the position of the vehicle on a 2D plane is projected onto the line between the nearest two points, and the squared distances between the vehicle and the projection over the course of a testing run are averaged.

Performance of the eight testing rounds done in the real world was initially measured based on the average time before the vehicle exited the lane, since no accurate standard deviation was available. However, once I had developed the lane detection system (Tech 2.1A), its positional error term was used as a proportional approximation of the standard deviation and used to compare its perfor-

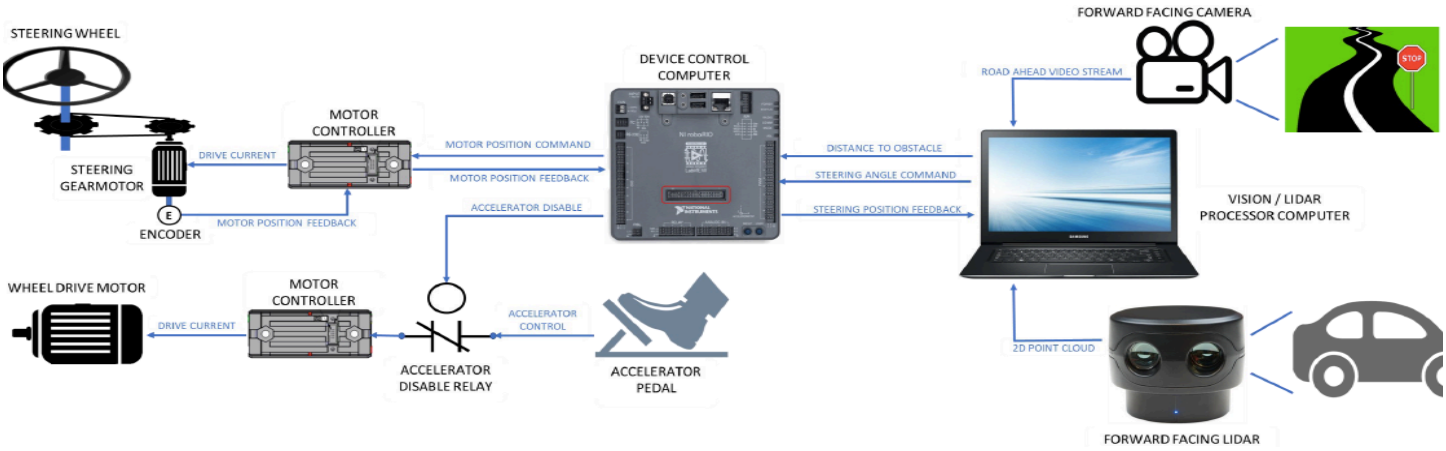

Figure I3. System architecture block diagram.

Figure I4. End to End CNN Visualizer.

mance directly to the real-world tests of Tech 2.1A.

I created a custom visualizer to view the neural network's actions compared to those of the human driver. The steering angles from both are represented graphically with steering wheels on the screen; the stability or erraticism of the output is represented in a graph of the steering angle over time. See Figure I4.

Technique 2.1A – CNN Sliding Windows for Lane Detection

This technique was based on the use of a CNN to predict the presence of road lines in a 16 by 16 pixel window. I created a graphical user interface (GUI) application for data collection, which allowed me to efficiently generate training data by slicing out square windows from images collected during the end to end CNN data collection runs.

Nine different neural network architectures with varying hyperparameters (number of convolutional and dense layers, number of filters, kernel size, activation function, batch normalization, and training batch size) were trained. I compared performance (training and validation losses) and computational efficiency (inference time).

The core sliding window algorithm works as follows:

1. The image is divided into overlapping windows with regular strides along each axis
2. Neural network inference is run, predicting the presence of a road line in each window
3. This produces a heat map of neuron activations, bright where road lines are present (See Figure I5).

I could not use this heat map directly for path planning; an algorithm was required to find the centre line of the road. The first attempt was a naïve approach as follows:

1. The brightest point to the left and right of each row is calculated
2. The horizontal "centre of mass" of each of the two road lines is calculated
3. The average of the two road lines is calculated to approximate the horizontal centre of the road
4. The error of this point off the centre of the image is used for path

planning

For more stable and robust calculation of the centre line, I applied a second, original iterative algorithm instead:

1. Starting at the bottom of the heat map, search laterally out from the centre to find a value exceeding a defined threshold on both sides
2. Calculate the average of the two points to define a point along the centre of the lane
3. Advance upward to the next row and repeat, using the lane centre as the new starting point for the lateral search (ensuring that if the road slants heavily to one side, the search will follow it and not begin outside)
4. If peaks could not be located on both sides, try again, starting a few pixels left or right (so that the lane will still be located if the car is on the edge of the road and both road lines appear on one side of the image)
5. Record the average point for each row, generating a series of points along the centre of the lane

This series of points on the lane centre can be used to compute a line of best fit. However, the outliers caused by vision errors are apt to cause incorrect centre lines, which could produce extreme steering angles and compromise the passenger's safety. My resolution to this was to run a series of outlier rejection passes:

1. Compute the line of best fit to the series of points using the normal equation
2. If all points are closer to the centre line than a certain threshold, stop and return the line of best fit
3. If not, remove the point farthest away (which represents an outlier due to a vision error) and repeat

Once again, I developed a visualizer application to verify this technique. It graphically represents the edges of the road and the centre line with points drawn on the image, and the line of best fit is superimposed. See Figure I6.

I modified many parameters of the sliding window and lane detection system during real-world experimentation: vertical and horizontal strides, image cropping parameters, the lane centre computation algorithm, the outlier rejection threshold, and the ideal lane centre position. In addition, I reimplemented the sliding window algorithm to process several windows in parallel, for improved com-

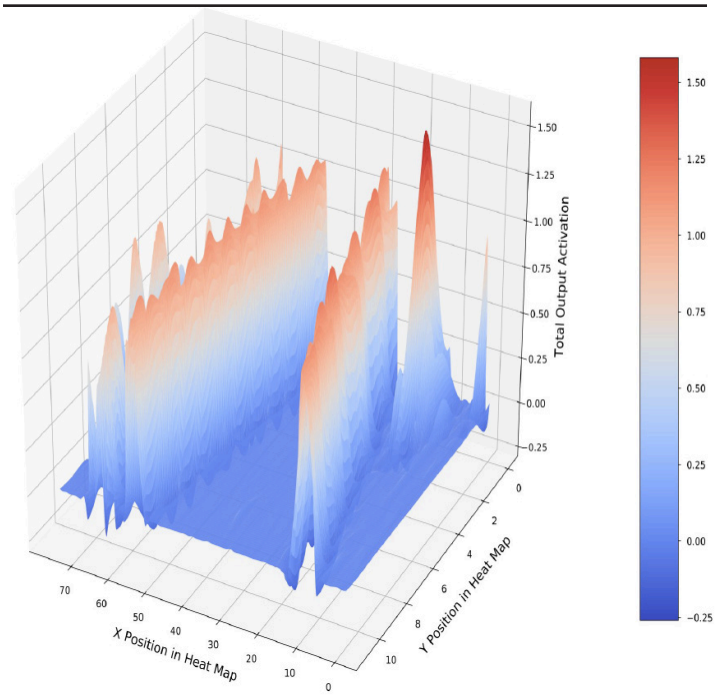

Figure I5. Sliding Window CNN Activation Heat Map.

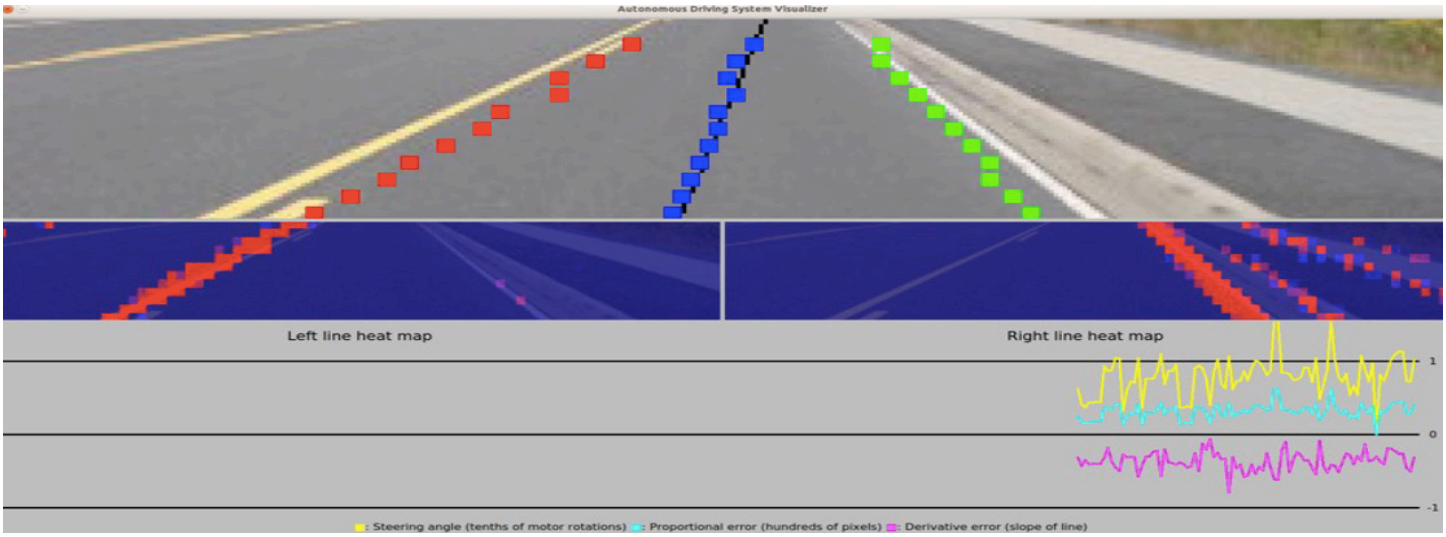

Figure I6. Lane Detection Visualizer.

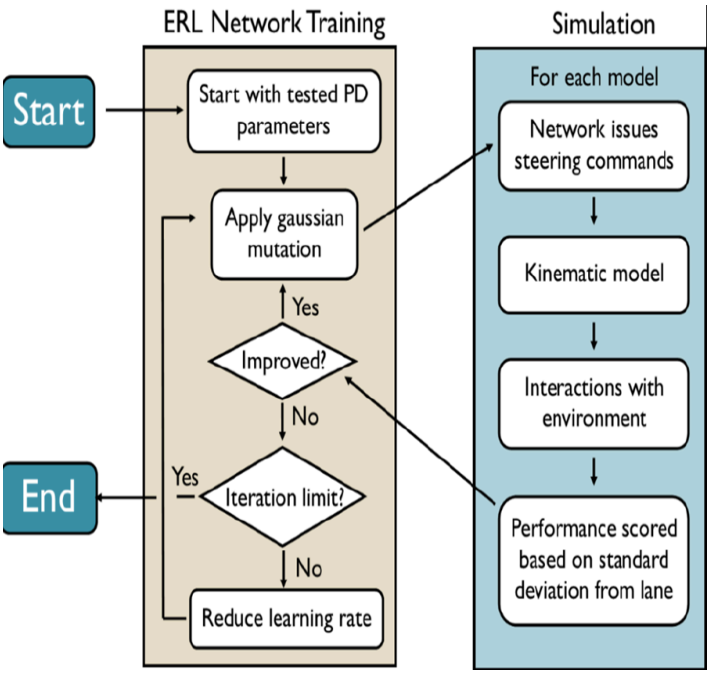

Figure I7. Reinforcement Learning via Evolutionary Optimization.

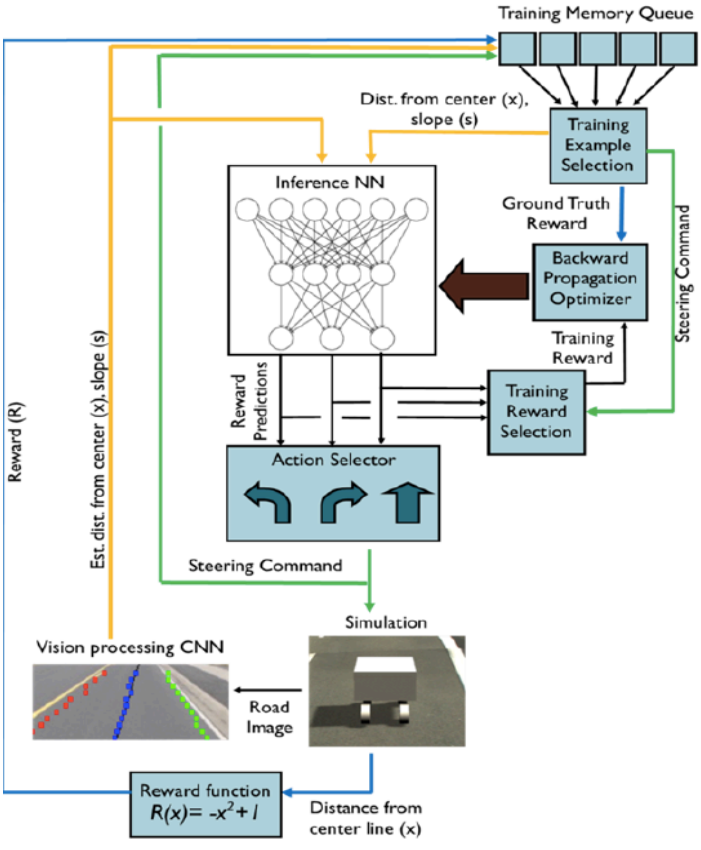

Figure I8. Deep Q-Network Diagram.

putational efficiency.

Technique 2.1B – CNN Wide Slice

This technique is based on a convolutional neural network that processes 16-pixel-high slices of the image and predicts the horizontal position of the lane centre in that slice. For inference, the road image is split into several vertical slices and each is processed individually, producing a horizontal position for each given vertical position.

Technique 2.2A – Proportional-Derivative Control

This is a classical control technique for path planning, which I used to generate steering angles from the lane's centre line. It is a conventional negative feedback loop where the error and the derivative of the error are each multiplied by constants and added up to produce a steering angle that should smoothly intercept the centre line of the road. One of my innovations was to use the slope of the centre line on the image (in x=my+b format, where 0 represents a vertical line) as a stable approximation of the derivative term, which in its original form is typically noisy or slow to respond. I optimized both the proportional and derivative terms in the real world over 33 rounds of testing.

Technique 2.2B – Evolutionary Optimization

Technique 2.2B is a refinement to proportional-derivative (PD) control that employed an evolutionary algorithm to optimize the PD control parameters. This is done by adding Gaussian noise to one of the parameters and testing the resulting model in the simulation. If the standard deviation is lower after the mutation, it is kept as the baseline for the next round of mutation. If it is higher, the mutation is discarded, and the previous baseline is left. (This means that performance can never get worse.) See Figure I7.

I also generalized this optimization algorithm to a four-neuron neural network (NN). It was initialized with two weights equal to the PD parameters such that its outputs are almost the same as the PD control system, except for the slight distortion introduced by the hyperbolic tangent activation function.

Technique 2.2B – Deep Q-Network

Technique 2.2C is a reinforcement learning concept originally devised by DeepMind,[3] in which a neural network is trained to predict future rewards for various actions, based on the reward variable provided by the simulation based on the model's performance. Specifically, the network is provided a state (in this case, the lateral position and slope of the centre line) and its outputs are the expected future rewards for each of a set of predefined actions. These consider not only the immediate reward, but also exponentially discounted future rewards. See Figure I8.

I developed several following variations of this algorithm spe-

cific to this project. I tested several different schemes for discounting future rewards. The set of available actions initially included simple increments of the steering angle to the left and right, but later I tested absolute (non-incremental) steering angles, as well as a more complicated function which behaves as an increment until the direction reverses, at which point it resets to the centre. Finally, I created an innovative training loop, in which, rather than only taking place at the end of an epoch (after the vehicle reaches the edge of the road), training is run as rapidly as possible in a coroutine, in parallel with inference for steering. This coroutine yields to the inference loop after each individual example, so it never locks up steering.

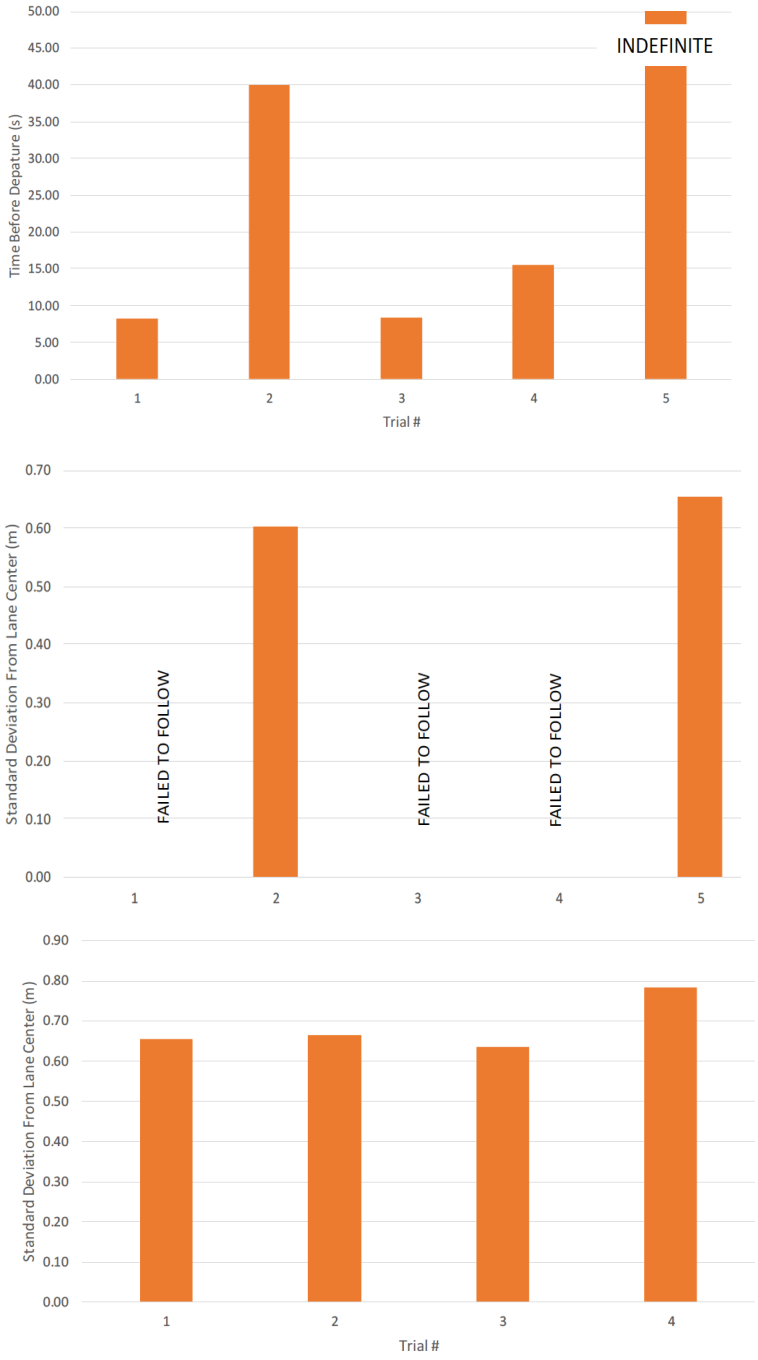

Figure G3. Early end to end CNN trials in simulation. (A) Initial algorith and simulation adjustments. (B) Intermediate hyperparameter optimizations. (C) Low pass filter parameter optimizations.

Performance was measured based on the number of inference steps taken before exiting the lane, and also the rolling standard deviation over the previous 5 minutes at any given time. (Training and performance measurement take place continuously, independent of departures from the lane, since the vehicle is reset to the center of the lane after departing it so that it does not spend any length of time unable to react and learn.)

Technique 2.2B – LSTM (Long Short-Term Memory)

An LSTM is a type of recurrent neural network (RNN) which has a complex memory subsystem that allows it to recall information from inputs in the long-term past. I trained it on time sequences of inputs and corresponding outputs, in this case centre lines and steering angles, respectively. I chose an LSTM with the hope for it to more effectively optimize the complex physical environment including the momentum of the vehicle and topology of the road, which is not ideally represented by the comparatively simplistic derivative term in PD control.

Speed Control System Software Development

Technique 3.1 – Adaptive Cruise Control

This technique employed a LIDAR sensor, which is a rotating radar that uses time of flight to create a point map of objects around the vehicle. I developed software to search a 10-degree arc ahead for an object within a certain distance, and a relay controlling the drive motors was switched to moderate the vehicle's speed. Once again, I developed a visualizer so that the algorithm for computing the position of the car ahead could be verified.

This was a software development effort, included for the sake of completeness and not specifically experimental, so no experimental results are included in this section other than to note that it performed successfully.

Technique 3.2 – Stop Sign Detection

I developed this technique with a vision pipeline very similar to that used for lane detection. A CNN was trained on manually selected stop sign images. Stop signs (and segments of stop signs) of all sizes were used to train the same CNN; size classification was done at a higher level on the heat map.

For inference, it was convolved over the image with a low stride of one pixel. I used a conventional computer vision algorithm for blob detection to identify bright blobs in the heat map, which are associated with stop signs. The position of the centre of the blob corresponded to the location of the stop sign in the origi-

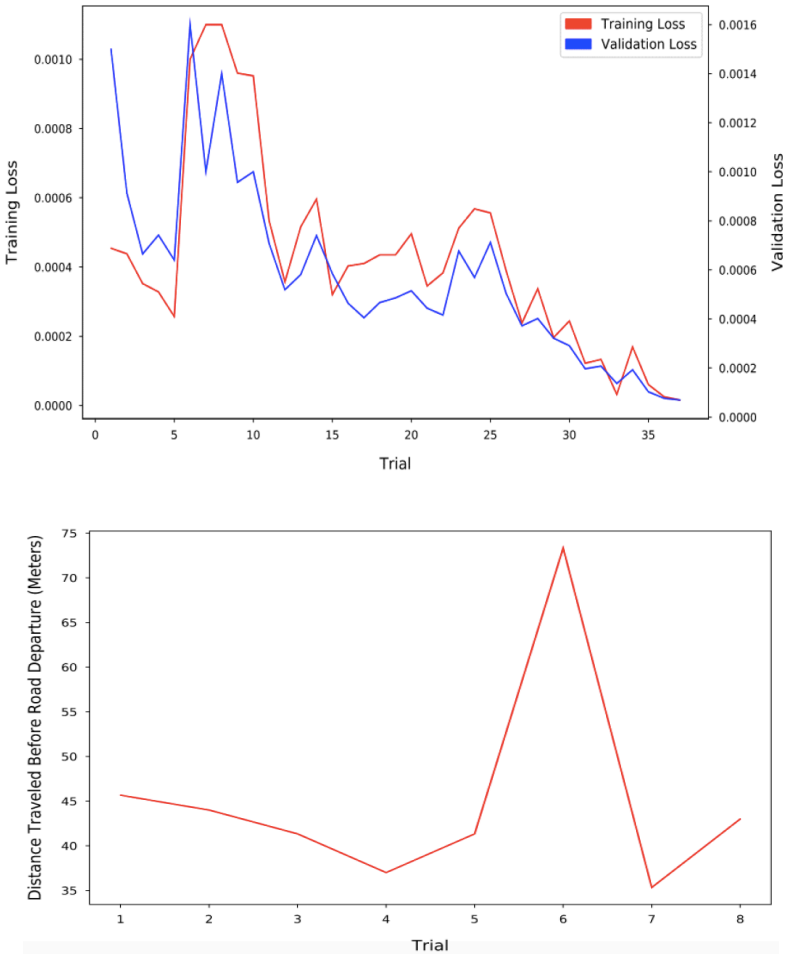

Figure G4. End to end CNN optimizations in real-world environment. (A) Loss function over training trials with varying hyperparameters. (B) Distance traveled before road departure during real-world trials.

nal image, and the area of the blob was used to approximate the distance to the stop sign.

I modified the sliding window CNN visualizer to display bounding boxes with the positions and apparent sizes of the stop signs, as well as labels showing their approximated distances.

RESULTS

Technique 1 – Combined End to End CNN (Simulation and Real World)

This technique was highly successful in the simulation. Initially, it failed to correct from slight errors, because it was trained on data collected by driving down the centre of the lane. I corrected this by adding a further training dataset including images of the driver correcting from off-centre positions. Optimized, the system successfully navigated the road for an indefinite distance, maintaining a standard deviation from the lane centre of 0.604 m. See Figure G3.

In real world testing, this technique was unsuccessful. With the initial training dataset, the vehicle exited the lane after an average of 45.7 m. I extensively optimized the architecture of the network and collected several new training datasets. Its loss function improved drastically, but it never successfully navigated more than 73.3 m. See Figure G4. The neural network failed to learn the complex, variable visual characteristics present in the real world.

Technique 2.1A – CNN Sliding Windows for Lane Detection (Simulation and Real World)

The sliding window CNN proved an effective technique for detecting the road lines; it reliably produced clean and high-con-

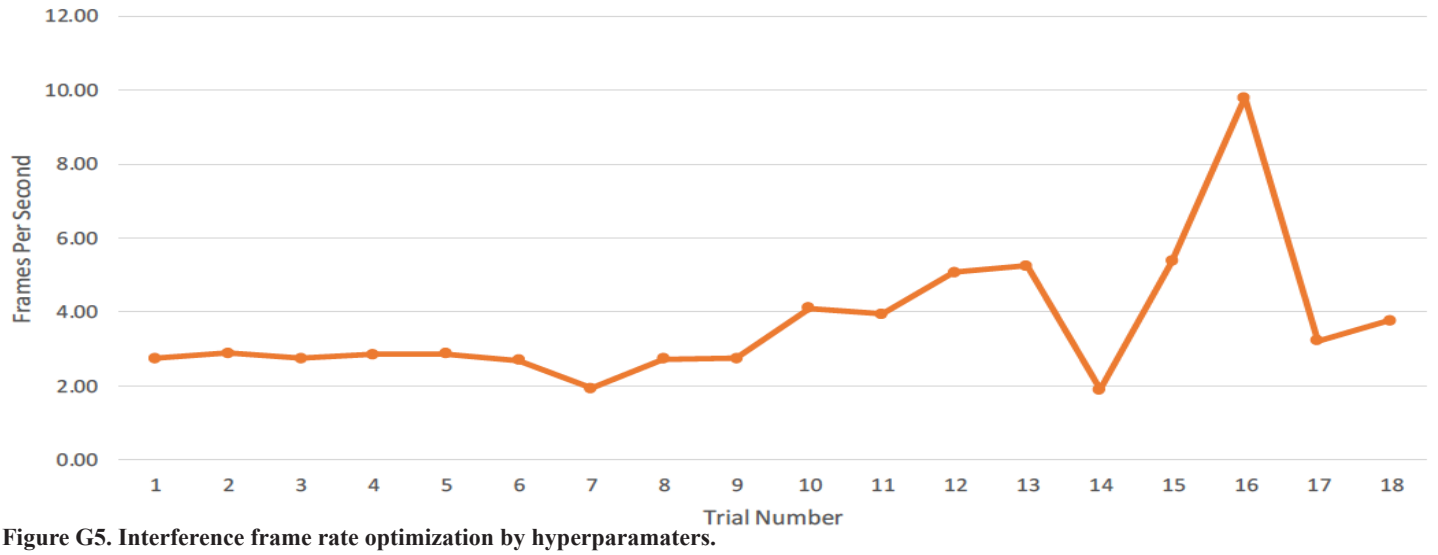

Figure G5. Interference frame rate optimization by hyperparamaters.

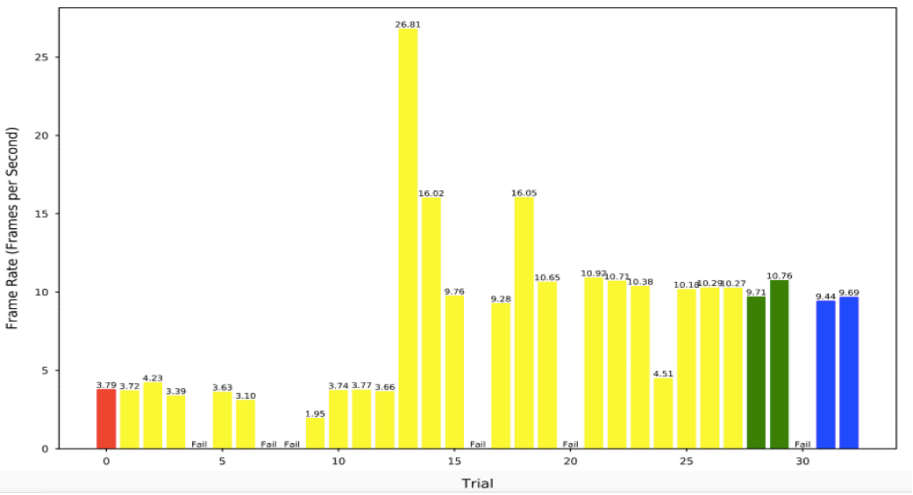

Figure G6. Frame rate over real-world trials of lane detection system.

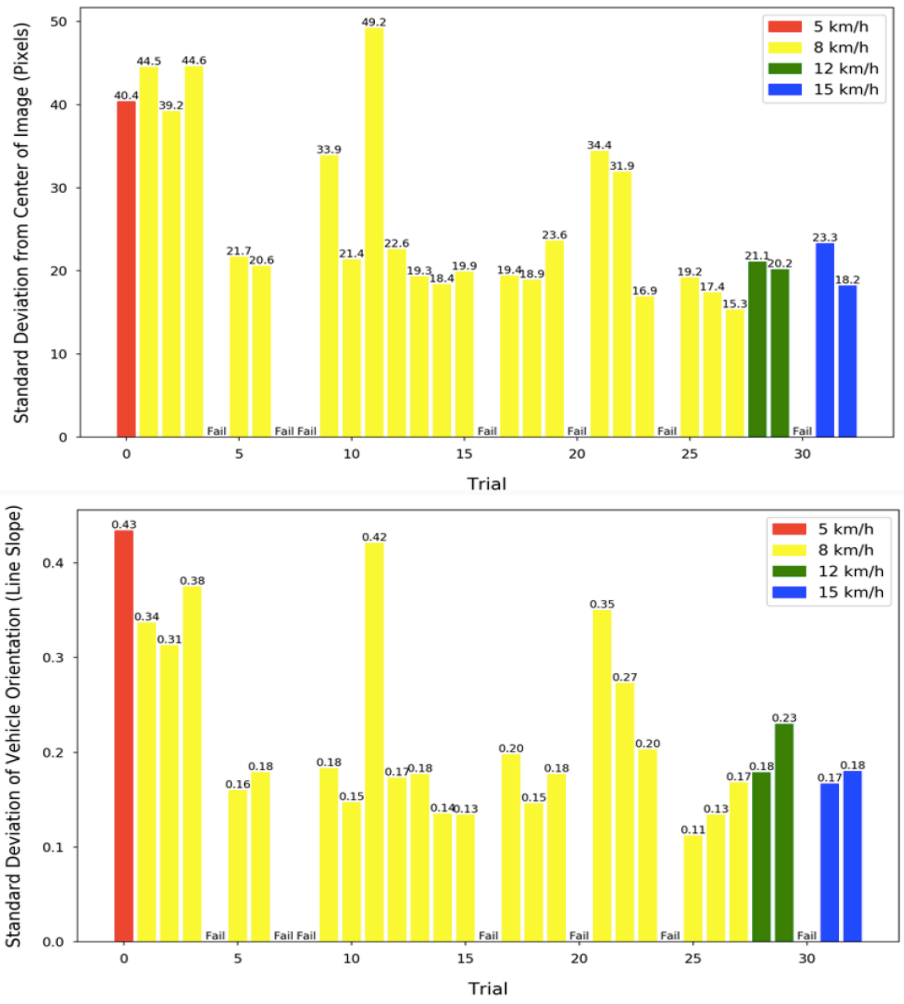

Figure G7. Positional and angular standard deviations during real-world trials (PD control). (A) Positional standard deviation from center of image. (B) Angular standard deviation from alignment with road.

trast heat maps. The best architecture I found included 2 convolutional layers, 16 filters, 2 dense layers, a hyperbolic tangent activation function, and an Adadelta optimizer.

Computational efficiency was a significant issue. During the first real-world tests, I observed frame rates under 4 frames per second (FPS). During initial experimentation, I found that neural network hyperparameters had very little impact, but that changes to image cropping and sliding window strides produced frame rates of up to 9.8 FPS. Using a new, vectorized sliding window algorithm, I increased this to 26.8 FPS. See Figures G5 and G6.

The first algorithm for lane centre calculation, involving a simple average of the horizontal positions of the road lines, was found to be very inconsistent. If the car was close to one side of the road, the line on that side would appear close to vertical and the other side would be very sharply sloped. This would result in a centre point too far in the opposite direction, leading to an overcorrection and oscillation. It was also very sensitive to outliers; one point on the edge of the image caused by imperfect classification would have a large effect on the steering angle.

With optimization, the second iterative technique was highly successful in identifying the lane centre line. I optimized its parameters alongside those of Technique 2.2A (PD Control) below.

Technique 2.1B – CNN Wide Slice (Visualizer Using Real-World Data)

Technique 2.1B failed to converge over multiple training trials. Its expected output is the position of the line in pixels. This is a suboptimal application for a neural network because it has no inherent concept of the position of a pixel, and it should end up learning parameters for a layer whose purpose is to map bright points on an activation map to a horizontal position. This makes it effectively a less robust version of the sliding window system.

Technique 2.2A – Proportional-Derivative Control (Simulation and Real World)

This system followed the lane indefinitely on

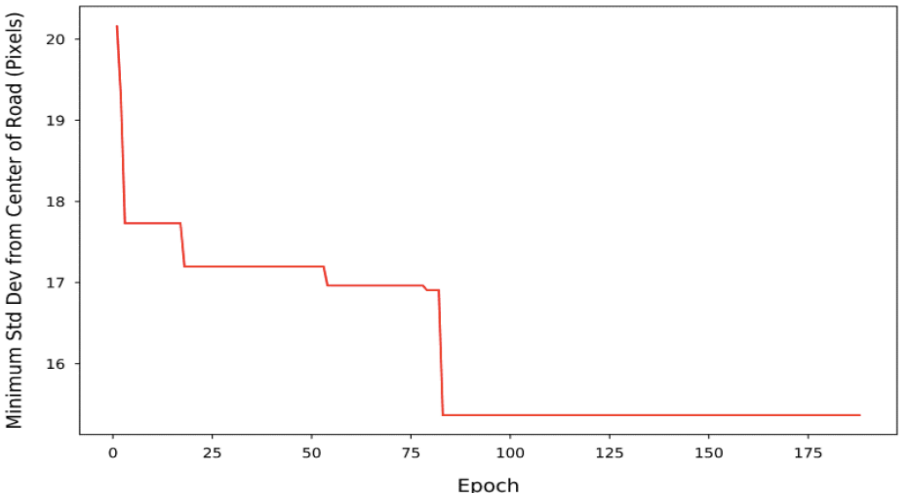

FIgure G8. Standard deviation over trials with evolutionary optimization.

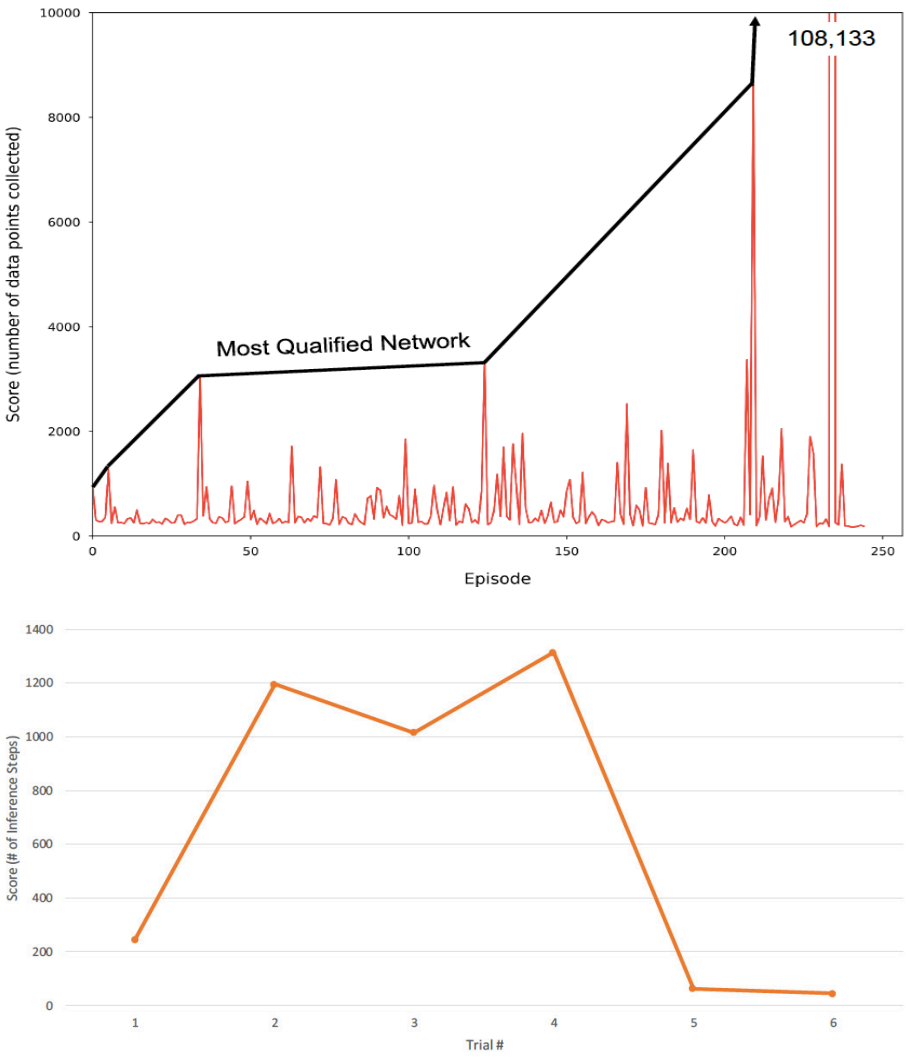

Figure G9. Time before failure during deep Q-network trials. (A) Score over episodes with initial training algorithm. (B) Maximum score over trials with accelerated coroutine training algorithm.

the first real-world trial (which included only the proportional term, with no derivative). Its positional standard deviation from the lane centre was 40.4 pixels, when driving at 5 km/h. Further refinements, including the second algorithm for centre line calculation, reduced this to 15.3 pixels at 8 km/h, or 18.2 pixels at 15 km/h. See Figure G7. The derivative term was incorporated later, but never reduced the standard deviation by more than 1.6 pixels. Even when using the best-performing parameters and neural network models, some higher-frequency oscillation was still evident.

Technique 2.2B – Evolutionary Optimization (Simulation)

In the best of the 10 rounds of testing with different algorithms and parameters, this technique improved the positional standard deviation significantly, reducing it from 20.16 pixels with manually optimized parameters to 15.37 pixels. See Figure G8. When it was generalized to a four-neuron network initialized using the PD parameters, its positional standard deviation was 14.4% worse than the optimized PD after evolutionary optimization.

Technique 2.2C – Deep Q-Network (Simulation)

After I had tested a series of action space configurations and run 234 training iterations, the deep Q-network demonstrated the ability to drive for over an hour, albeit with significant erratic oscillation (like the early end to end models) and a high positional standard deviation of 0.793 m. Many subsequent tests that I conducted using the new training algorithm and modified parameters failed to produce equivalent performance. See Figures G9 to G11. The deep Q-network may have an issue similar to the end to end CNN: it is a black box that is difficult to debug or optimize by hand, so it is impractical to determine in advance which parameters will improve performance.

Technique 2.2D – LSTM (Simulation)

Use of a long short-term memory RNN for steering based on time sequences of inputs failed due to the same problem observed the earliest tests

Figure G10. Data collected at maximum time before failure during deep Q-network trials. (A) Episode number at maximum time before failure. (B) Epsilon (discount factor) at maximum time before failure. (C) Rolling average standard deviation at maximum time before failure.

of the end to end CNN. It was unable to drive for more than 9.3 seconds after being trained on a dataset collected while driving down the centre of the road, not including off-centre images. See Figure G12. An ideal dataset is time-consuming to collect, especially for a recurrent neural network, which requires extended time sequences of consecutive images. With this problem observed, I pursued this technique no further.

Technique 3.2 – Stop Sign Detection (Simulation)

The combination of a sliding window system and a blob detection algorithm proved very effective; it was able to compute the position and size of stop signs accurately enough to draw precise bounding boxes. However, performance was an issue; because the stride is only one pixel, inference requires just under 50,000 iterations to process a 320x180 image. Inference took 5.92 seconds per image with the initial model.

To resolve this, I tested several combinations of hyperparameters, including varying numbers of convolutional and dense layers, convolutional filters, dense layer neurons, and varying convolutional kernel sizes. I found the best combination of performance and computational efficiency in a network with two convolutional layers (with 12 filters) and two dense layers. It was able to run inference on a 320x180 image in 1.67 seconds. See Figure G13.

CONCLUSIONS

1. Producing a full level 2 autonomous vehicle is attainable with limited resources. This included a steering system, stop sign and forward vehicle recognition.
2. CNN Sliding Windows combined with PD control (optimized by hand or with evolutionary algorithms) proved the most effective and practical technique for steering the vehicle, both in the simulation and the real world.
3. Combined End to End CNNs worked in the simulation but proved too fragile and sensitive to optical variation to be used in the real world without substantially more training data. The black-box nature of the technique means that it is exceedingly difficult to troubleshoot. It is academically interesting but impractical.
4. Deep Q-Networks demonstrated potential but require further work. There is some question about whether the discrete action space of deep Q-networks is appropriate for this application.

ACKNOWLEDGEMENTS

I created all electronics and software for this project without any guidance or mentorship; however, I would like to thank Steve Matusch for guidance on the mechanical construction of the vehicle.

REFERENCES

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski et al, "*End to End Learning for Self-Driving Cars*". 25 Apr 2016, https://arxiv.org/pdf/1604.07316.pdf.

François Chollet et al, *"Keras"*. https://github.com/keras-team/keras.

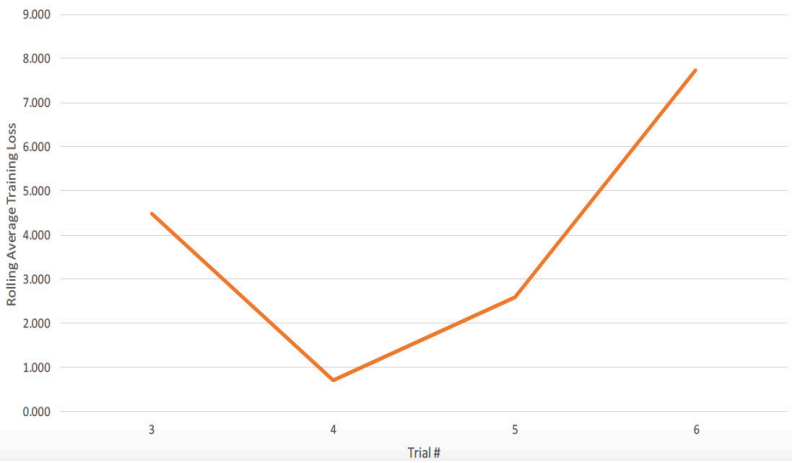

Figure G11. Rolling average training loss at maximum time before failure.

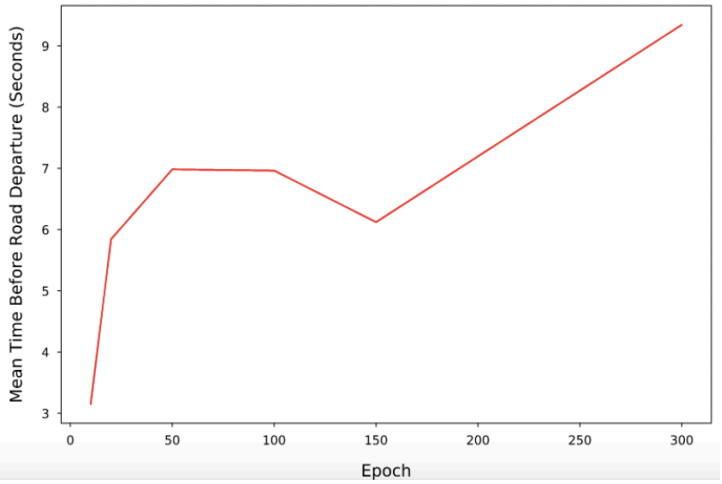

Figure G12. Time before road departure over LSTM steering trials.

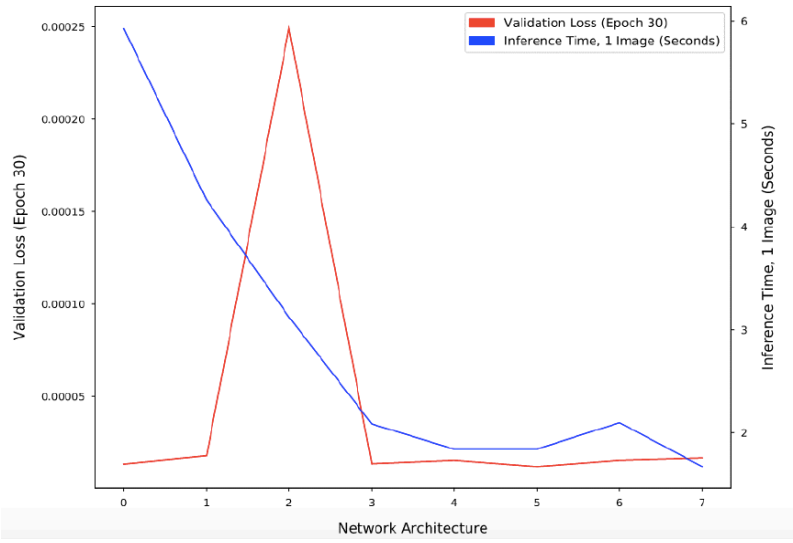

Figure G13. Accuracy and computational efficiency by trial for stop sign detection.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver et al, *"Human-level control through deep reinforcement learning"*. 26 Feb 2015, https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf.

BIBLIOGRAPHY

Andrew Ng, *"Sliding Windows – Application Example: Photo OCR"*. https://www.coursera.org/learn/machine-learning/lecture/bQhq3/sliding-windows.

Christopher Olah, *"Understanding LSTM Networks"*. 27 Aug 2015, http://colah.github.io/posts/2015-08-Understanding-LSTMs.

Keon Kim, *"Deep Q-Learning with Keras and Gym"*. 6 Feb 2017, https://keon.io/deep-q-learning.

Soumith Chintala, Adam Paszke et al, *"PyTorch"*. https://github.com/pytorch/pytorch.

ABOUT THE AUTHOR

My name is Brendon Matusch. I am in grade 10 at Lo-Ellen Park Secondary School. My hobbies include FIRST Robotics (I am on Team 4069), playing the piano, and of course science fair. I plan to go to the University of Waterloo for software engineering, and then move to Silicon Valley. My project this year, in which I developed an autonomous vehicle, was inspired by Tesla (especially Andrej Karpathy and of course Elon Musk), who are doing some fantastic and exciting work to bring us closer to practical autonomous and electric vehicles. I was also inspired by previous visits to the Canada-Wide Science Fair in 2016 and 2017, where I was originally introduced to the incredible world of machine learning and neural networks. The most important advice that I have for other students interested in science fair is the value of hard work and perseverance. I have been working steadily on this project since July, and it has been extremely difficult at times. I have learned never to give up, and that lesson has paid off.