2017

Autonomous Driving with a Simulation Trained Convolutional Neural Network

Cameron Franke
University of the Pacific, camfranke@gmail.com

AUTONOMOUS DRIVING WITH A SIMULATION TRAINED CONVOLUTIONAL NEURAL NETWORK

by

Cameron Franke

A Thesis Submitted to the

Graduate School

In Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

School of Engineering and Computer Science
Engineering Science

University of the Pacific
Stockton, California

2018

AUTONOMOUS DRIVING WITH A SIMULATION TRAINED CONVOLUTIONAL NEURAL NETWORK

by

Cameron Franke

APPROVED BY:

Thesis Advisor:	Elizabeth Basha, Ph.D.
Committee Member:	Michael Doherty, Ph.D.
Committee Member:	Chadi El Kari, Ph.D.
Department Chairperson:	Michael Doherty, Ph.D.
Dean of Graduate School:	Thomas H. Naehr, Ph.D.

ACKNOWLEDGMENTS

I would like to thank Dr. Elizabeth Basha for her support and encouragement without which this research would not have been possible.

Autonomous Driving with a Simulation Trained Convolutional Neural Network

Abstract

by Cameron Franke

University of the Pacific
2018

Autonomous vehicles will help society if they can easily support a broad range of driving environments, conditions, and vehicles. Achieving this requires reducing the complexity of the algorithmic system, easing the collection of training data, and verifying operation using real world experiments. Our work addresses these issues by utilizing a reflexive neural network that translates images into steering and throttle commands. This network is trained using simulation data from Grand Theft Auto V [1], which we augment to reduce the number of simulation hours driven. We then validate our work using a RC car system through numerous tests. Our system successfully drive 98 of 100 laps of a track with multiple road types and difficult turns; it also successfully avoids collisions with another vehicle in 90% of the trials.

TABLE OF CONTENTS

Chapter 1: Introduction

Self-diving vehicles will provide more than a simple luxury, they will eliminate accidents caused by tired, intoxicated or distracted drivers. By freeing travelers of the need to also be drivers, autonomous vehicles will bolster the productivity of commuters by allowing passengers to work while they travel. As autonomous vehicles become ubiquitous they will also reduce travel times by improving the flow of traffic. Traffic jams caused by accidents, ‘rubber-necking’ and congestion will be eliminated by the ability of autonomous vehicles to relay information about road conditions and traffic patterns to each other. By giving us back the time we currently spend driving and ensuring our safety, self-driving cars will offer significant value to humanity. However, before trusting our lives to self-driving cars, we must ensure that they are able to stand up to the complex challenges of real-world driving.

1.1 Challenges

Humans are effective at driving because of our powerful intuition. The human brain is able to absorb large amounts of data, filter out what is important and use that information to make decisions. Driving has traditionally been a very challenging task for computers because of the wide variety of possible scenarios and lack of intuition. Humans are easily able to identify obstacles, lane marking and other vehicles however, this is a very challenging task for computers. The challenge of mimicking human perception can be solved by machine learning algorithms. Although machine learning algorithms represent a promising approach to autonomous driving, they come with their own set of challenges. One such challenge is the amount of data that is required to train them.

The sheer volume of data needed to create an effective vehicle controller presents a sizable engineering challenge. In order to drive in a particular scenario an autonomous vehicle controller must have human-collected data demonstrating the behavior necessitated by that scenario. For this reason driving has always been a human dominated task. Humans are able to adapt to varying lighting, weather and traffic conditions exceptionally well. Adapting to the wide variety of conditions that can be observed on the road is a challenging task for computers and is one that is best addressed by an algorithm that can learn and generalize in the same way that humans do.

Autonomous vehicles are enabled by a collection of machine learning and other algorithms. These algorithms rely heavily on data collected by human drivers and require a layered approach to transform data into vehicle controls. This data often consists of video streams from multiple on board sensors including cameras, lidar, radar and infrared. This information can be used to tune or train algorithms, essentially acting as experience to be learned from. Environmental changes such as new road types or weather conditions require collection of hundreds of hours of data, processing and labeling this data to train the algorithms, and re-training the system. System changes such as new sensors require modifications to the algorithms and intermediate layers that translate high-level algorithmic outputs into low-level controls.

1.2 Solution Overview

Our work addresses the challenges of autonomous driving through training based on realistic simulation data and providing an end-to-end learning approach. The neural network that we present is reflexive, which means that it accepts raw sensor data as input and directly produces vehicle controls as output. Neural networks are computational representations of brains and are capable of learning in a similar fashion. We used our realistic simulation data to train our neural network to operate a car. We ensure the functionality of our approach through real world experiments on a RC car test system. To the best of our knowledge, this combination of simulated training, a reflexive neural network, and real world verification has not been previously attempted and provides a novel approach to improving autonomous vehicles.

The neural network that we implemented is called a convolutional neural network or 'CNN'. The CNN is a modular architecture consisting of small networks tailored for particular tasks. A smaller CNN is used for image processing as that network style best suits that input, a more traditional multi-layer perceptron network processes the steering data, and a deep multi-layer perceptron network merges the results to provide a steering angle for the vehicle. Parallel to this network is an additional multi-layer perceptron network focused solely on throttle and braking. By breaking up the neural network in this manner we are able to achieve a high level of modularity. Adding additional inputs to accommodate additional sensor readings would be trivial due to this design. Beyond that, separating the steering control and throttle control networks reduces training time and network complexity.

Neural networks perform what is called 'imitation learning' which means that they learn to replicate behavior that they observe. To train the CNN that we designed we needed a large amount of data. Because of the prohibitive cost and potential danger of collecting data in a real-world car we decided to use a simulation environment to gather training data. The simulation that we chose to use is a video game called Grand Theft Auto V [1]. We chose this game as our simulation because of it's realistic environment and the ease with which the game can be modified. We opted to used examples of ourselves driving in Grand Theft Auto V as our training data.

1.3 Data Collection and Training

Before the neural network training process could begin we needed to collect the data with which to train the neural network.

We begin the data collection process by modifying the game environment to our liking and repositioning the in-game camera so that it is fixed to the hood of the in-game test car. The simulated camera provides images in front of the vehicle as it drives on a variety of road types and traffic conditions. A data collection script running in the background collects information about the driver's steering wheel angle and throttle value. Each image from the simulation is labeled with the corresponding steering and throttle values.

After the data has been recorded it is used as an example to train the neural network. The image captured from the in-game camera is used as input to the neural network which is then asked to produce the corresponding steering and throttle values. By identifying the conditions present in each image the network is able to learn to adapt to a variety of driving conditions and make generalizations about the task of driving. To expand the set of data further, we augment the data through cropping selected regions of each image, which provides examples of poor driving and the correction needed in those scenarios. Not only does this increase the volume of data that we have but it imbues the neural network with the ability to correct its mistakes.

1.4 Results

After collecting data and training our neural network, we verified the functionality on a verification data set, which resulted in an average error rate of 1.9%. We then connect this network to a RC car system, which was chosen for safety during the development process. The RC car provides a camera input and receives a steering angle and throttle value. We test the steering sensitivity, reliability, performance, and obstacle avoidance. The car successfully navigates 98 out of 100 laps of a track specifically designed to challenge it with different road types and tight turns. It also successfully avoids another car with a 90% success rate.

These performance results could certainly be improved by adding additional sensors to our system. Another front-facing camera to enable stereoscopic 3D would improve obstacle avoidance. Additional side and rear facing cameras could enable behavior such as backing up, 3-point turns and safe lane changes. Although we lacked these inputs we were still able to demonstrate the potential of end-to-end neural networks as vehicle controllers and the potential for simulations to be used as data sources.

1.5 Contributions

Our contributions to this field of study can be summarized by the following:

- Training of an end-to-end convolutional neural network
- Training of a neural network with data taken from the video game Grand Theft Auto V
- Testing of a neural network trained exclusively with simulation data in the real world
- Instrumentation of a small scale vehicle controlled remotely by a neural network
- Support of the notion that simulations are a viable training ground for autonomous vehicles

1.6 Organization

This paper first describes related work in Chapter 2 followed by a description of our neural network's architecture in Chapter 3. Chapter 4 discusses our experiments with the RC car. Chapter 5 summarizes the results of our experimentation and potential future work.

Chapter 2: Background and Literature Review

In this chapter we will present an overview of neural network concepts to provide context to the forthcoming discussions. Section 2.1 includes the overview of neural networks guided by a walk-though of a theoretical neural network. In Section 2.2, we will discuss existing research that is relevant to our implementation of autonomous vehicles.

2.1 Neural Network Overview

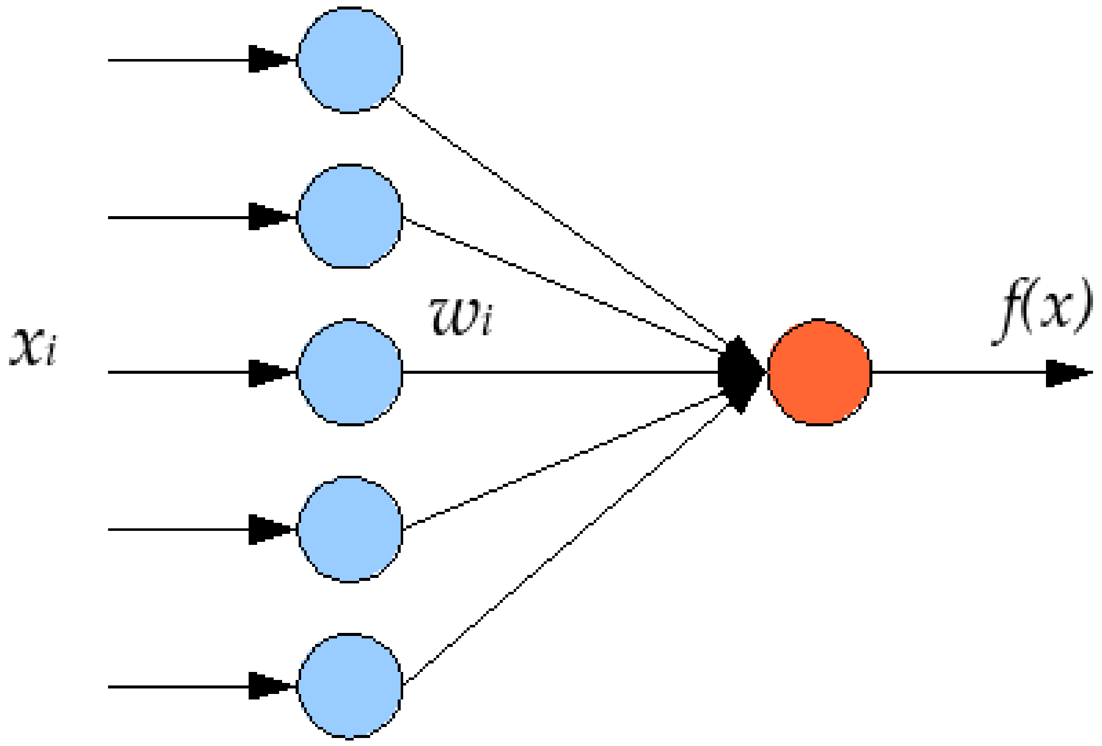

Figure 1: Single Layer Perceptron

Although the computational model for neural networks was first introduced in 1943 [2], it has only been the past twenty years that they have begun to come into their own as a component of Artificial Intelligence. The exact structural implementation of neural networks varies widely from simple, single layer perceptrons to convolutional deep neural networks. Neural networks can be easily represented with graphs. A neuron is simply a node on the graph that accepts one or more pieces of numeric input data, applies some arithmetic function, and produces an output value. The arithmetic function applied by a neuron to its input is called the 'activation function'. The function is named this way because it controls the degree to which the neuron activates. The single layer perceptron featured in Figure 1 is the simplest form of an artificial neural network, or 'ANN', as it features only one neuron. Each of the blue circles represents a feature of the input data point, denoted X. To determine the neuron's output the numeric values for each of those features is multiplied by a weight value, W. The resulting values are then passed to the neuron, which is represented as the orange circle. The neuron will perform a summation of the weighted inputs that it receives. The resulting value will then be passed to the neuron's activation function. Two of the most basic implementations of the activation function are the step function and the sigmoid function. The step function relies on a threshold value and returns 0 if the weighted sum is less than that threshold, and it returns 1 if the weighted sum is greater than the threshold. The output of the sigmoid function, $S(t) = 1/(1 + e^{-t})$, approaches 1 as t goes to infinity and approaches -1 as t goes to negative infinity.

One method of handling neural network output is to have the number of neurons in the final layer be equal to the number of categories that the network is able to identify. In this situation, a given neuron would correspond to one class of output and would fire if the network thinks that the input data is a member or that class. Under this paradigm, the single neuron network in Figure 1 would only be able to determine if its input is a member of one single category or not.

If more neurons are added to this network, more objects can be identified. For example, suppose there exists a data set $\{x, y, z\}$ such that x is the number of legs an animal has, y is the animal's height in inches and z has the value 1 or 0 representing

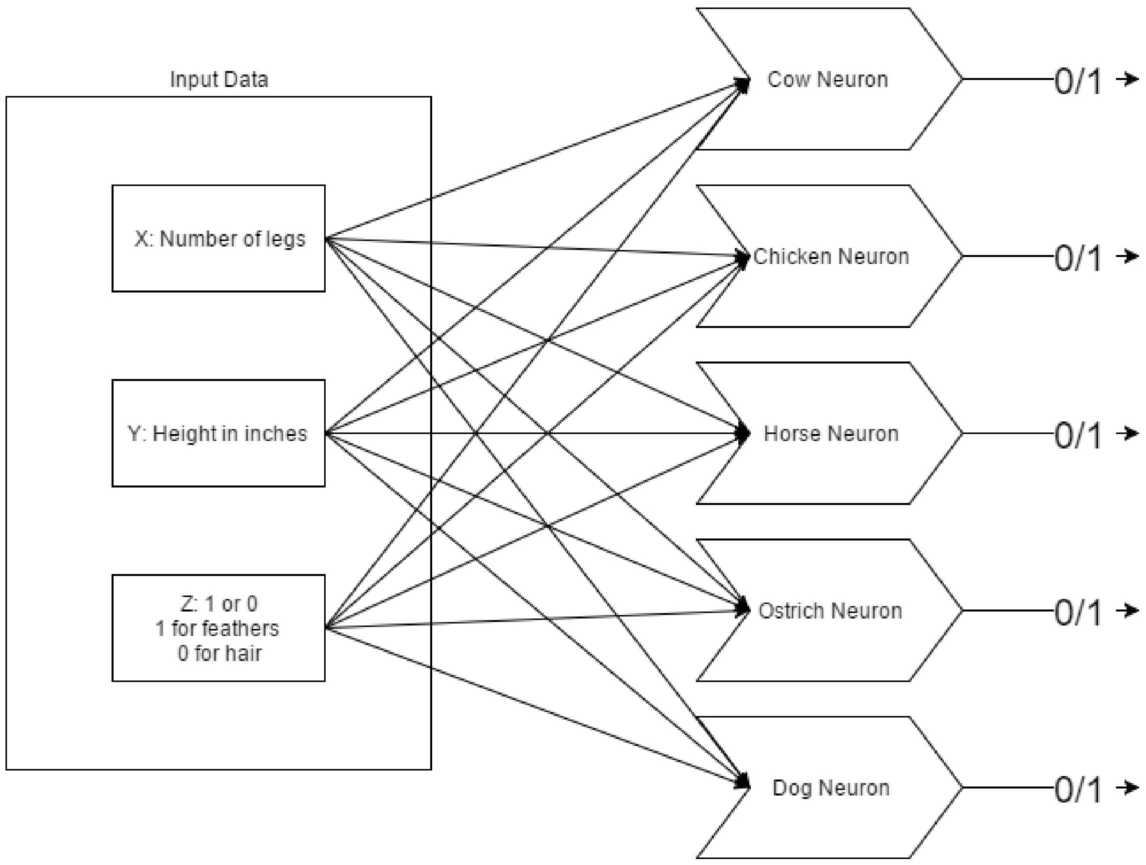

Figure 2: Animal Classifying Neural Network

whether the animal has hair or feathers. If this data set was fed to a single neuron network, like the one in Figure 1, it would only be able to learn to classify whether or not the input data belonged to one specific animal. For example, the neuron would output 1 if it thought that the sample was a cow and 0 if it did not. However, if this same data set were to be provided to a network of five neurons, we may be able to classify a sample as being a cow, chicken, horse, ostrich, or dog. In this situation, each of the neurons would correspond to an animal, as seen in Figure 2. If a given data point representing the features of a cow was supplied to the network, one would expect the chicken, horse, ostrich and dog neurons to all output 0 while the cow neuron, and only the cow neuron, would output 1. This output scheme is sometimes referred to as ‘one-hot’ since only one of the neurons should be ‘hot’ and output a 1.

While this example illuminates some of the potential of ANNs, it still only deals with single layered networks.

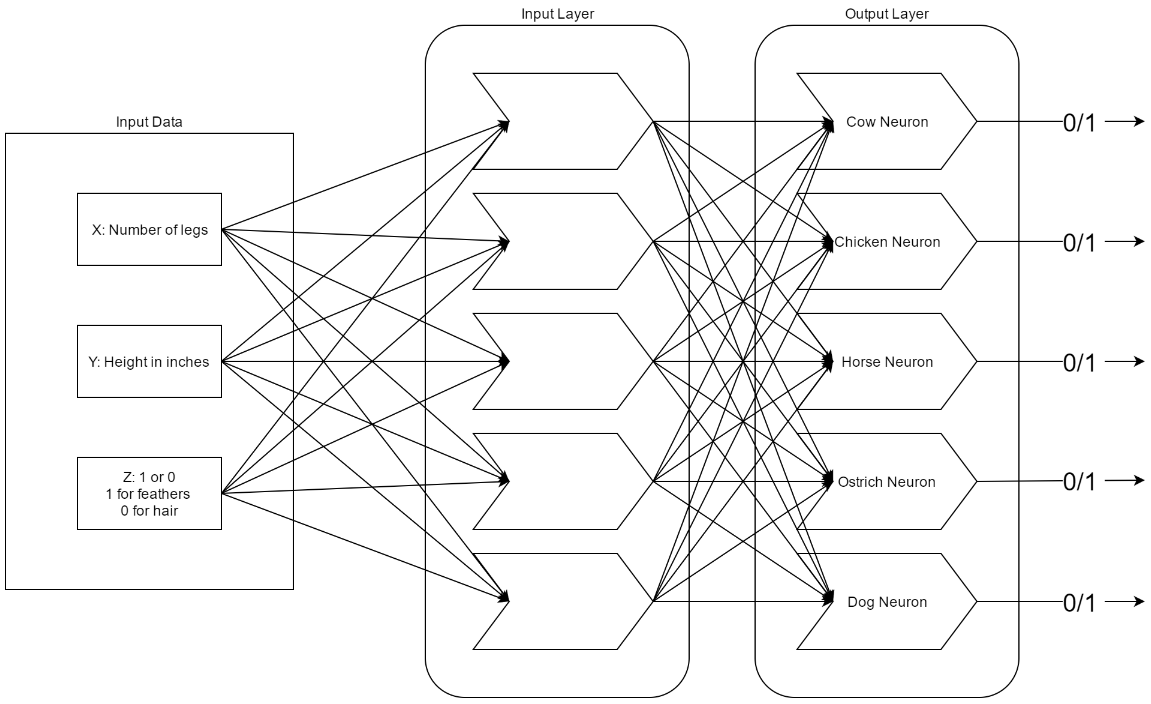

Figure 3: Animal Classifying Deep Neural Network

To expand on the ‘animal classifier’ outlined above, consider a few modifications. First, replace the step function in each of the neurons with the sigmoid function $S(t) = 1/(1 + e^{-t})$. This way, instead of a 1 or 0 each neuron will output a floating point value between -1 and 1. Second, add a row of five more neurons so that the outputs of the first five neurons are the inputs to all of the new neurons. This new layer of neurons will serve as the output layer and will use the step function. Notice that each neuron in the first layer, or the input layer, receives three pieces of information and the neurons in the second layer, or the output layer, receive five pieces of information, one from each of the neurons in the input layer. This new network is called a multilayer perceptron or a Deep Neural Network (DNN) because there is now depth to the network. Because there are more connections in this new DNN 'animal classifier', it is capable of learning more complex patterns and has increased its maximum potential accuracy. Following this strategy, one could continue to add layers to the network indefinitely. Although in this example, increasing the network depth would result in improved performance, that is not always the case. Depending on the complexity of a task the ideal dimensions of a network vary significantly.

Another way to structure neural network output is to use a single output neuron with a sigmoid activation function in the last layer. In the above examples, the output neurons all use a step function. We can see this because they only output a 1 or 0. However, if we change to a single output neuron with sigmoid activation, or another activation function that produces a floating point value, we can produce a wider range of outputs. Producing a floating point value allows us to estimate the certainty of our classifications or classify things that fall on a sliding scale, such as the angle to turn a steering wheel.

2.2 Related Works

Autonomous vehicle research has focused on a wide range of approaches. Our focus is on those using neural networks; in this section we provide an overview of the related research using that approach with a specific focus on those using simulations, end-to-end networks, or real world verification.

2.2.1 Neural networks and non-driving tasks. Although there are a large number of works that fall into this category, we have chosen to include the papers in this section because they express the capability of machine learning algorithms to solve computer vision tasks.

In this section we will review research focused on the use of neural networks to solve image processing problems unrelated to autonomous driving. Although these task are unrelated to autonomous driving itself, the problems are framed in a very similar way. Autonomous driving systems that employ neural networks must perform classifications on images from a video stream. These works describe the same methodology that we use to achieve autonomous driving, but applied to different tasks.

In their research paper published in 2012 Ciresan *et. al.* presented a neural network capable of correctly classifying street signs 99.4% of the time [3]. While the proposed system was trained on images pertaining to driving, the classifier itself could be trained to classify any type of image. In fact, Ciresan *et. al.* published another paper in 2012 using a similar neural network that was trained to classify human handwriting [4]. That network scored a 0.23% error rate on the MNIST handwriting data set. The work of Ciresan *et. al.* goes to show that neural networks are highly capable image classifiers. The authors claim that their classifier is insensitive to contrast and illumination. Dealing with varying lighting conditions is a considerable challenge in image processing problems.

In the paper published by researchers at University of Toronto in 2012, authors Krizhevsky *et. al.* outline a neural network used to classify images from the ImageNet image database [5]. The ImageNet data set contains images from 1000 different classes. Krizhevsky *et. al.* were able to achieve a record breaking error rate of 15.3% in the 2012 ImageNet Large Scale Visual Recognition Challenge.

Convolutional neural networks have become well known for their effectiveness in solving computer vision problems. Karpathy *et. al.* created a neural network trained to determine the sport being played in a video clip [6]. Video classification can be challenging due to the additional dimension in the data space. By looking at multiple frames across a period of time, networks can learn to identify spaciotemporal features. Karpathy *et. al.* presents a variety of methodologies for feeding the information from multiple video frames into a neural network. Although the classifications are different in autonomous driving tasks, the ability to perform classifications on a stream of images is highly relevant.

2.2.2 Autonomous driving without machine learning. In this section we will review significant attempts to create autonomous vehicles that do not incorporate machine learning methodologies.

The most commonly known facet of autonomous driving research is the DARPA Grand Challenge [7]. The first DARPA Grand Challenge took place in 2004 and required competitors to instrument vehicles to drive a long distance course in the Mojave Desert. In the first race, none of the competitors successfully completed the course. In 2005 DARPA held a second grand challenge which was not only completed but won by Stanford Racing Team's Stanley vehicle [8].

To the best of our knowledge, none of the cars used machine learning approaches instead opting to perform vehicle control with more conventional algorithms dependent on input from a variety of sensors such as cameras, GPS, and lidar [8, 9]. Machine learning algorithms are generally applied to images when used for autonomous driving tasks. In the DARPA Grand Challenge camera input was less useful than it normally would be due to the terrain. In the first DARPA Grand Challenge the track was entirely off-road and much of it covered uneven terrain. Because much of the road and surrounding terrain was brown, color had no easily discernible meaning. Thus camera input could not be relied upon as much as radar and lidar.

In 2007 DARPA hosted a second grand challenge, the DARPA Urban Grand Challenge. The Urban Grand Challenge took place at George Air Force Base in Victorville, California and covered 60 miles of urban road. Urban grand challenge teams were provided with aerial imagery of the region of the challenge. Team Tartan from Carnegie Mellon University entered the competition with a robot built on a Chevrolet Tahoe chassis [10]. Team Tartan's robot used camera imagery, lidar and radar as input to it's control algorithms. Like past grand challenges, lidar and radar played a large role in navigational decision making processes. In this case however, camera imagery was used for basic line following functionality. Team Tartan did in fact use machine learning for the challenge but only in off-line preprocessing. Tartan used a convolutional neural network to aid in the process of extracting road information from the provided aerial imagery which was later used for high-level path planning. Since the neural network was not used at drive-time, this paper is not included in Section 2.2.3.

2.2.3 Machine learning and autonomous driving. In this section we will review research pertaining to machine learning and autonomous driving.

To the best of our knowledge the first use of a neural network for driving was Carnegie Mellon University's ALVINN [11]. ALVINN used a reflexive neural network to control its steering output. Reflexive neural networks map input information directly to a control output. In this case, the input is imagery from ALVINN's camera and the output is a steering angle [12, 13, 14]. The network used to control was trained with real world data as well as tested on a real world vehicle. Due to the limited computational power of the time, ALVINN's neural network needed to operate on very small inputs; the images provided to the network were only 30x32 pixels.

The mediated perception approach to autonomous driving with neural networks involves classifying objects such as lane markings and other cars in an image; additional algorithms then utilize the classified objects to determine the driving approach [15, 16, 17]. Johnson *et. al.* and Filipowicz present two separate methodologies for modifying GTA V specifically for the purpose of supporting the training of mediated perception systems [18, 19].

Filipowics used his system to train a network to classify the distance to stop signs [17, 18]. This network was trained on simulated data and tested within the same simulation environment but, not verified on a real world system. Like the work of Johnson *et. al.*, Filipowicz's paper is predicated on the idea that the graphical quality of GTA is high enough to be passable for reality. The performance on Filipowicz's classifier supports the notion that GTA V is a viable training ground for autonomous vehicle control systems.

Johnson *et. al.* also created a system for annotating images from GTA V [17]. Instead of classifying distance to stop lights Johnson chose to detect vehicles in images. To test his vehicle classification system Johnson used 7481 images from the KITTI dataset. Johnson was able to show that neural networks trained with annotated data generated from simulations were as effective at identifying vehicles in the KITTI dataset as networks trained with real-world data.

Although it does not deal with directly with steering control, Vallon *et. al.* present an approach to controlling lane changes autonomously [15]. Vallon *et. al.* use support vector machines, or SVMs, to learn the optimal lane change timing based on the distance to other cars that may be present on the road. This paper addresses one of the many challenges involved in achieving an autonomous vehicle with a high degree of autonomy.

Huvel created another neural network powered autonomous driving system that would fall under the mediated perception category [16]. The system proposed by the Stanford research team is capable of classifying other vehicles on the road and lane markings. Vehicles are classified by sub sampling small context windows and determining if a vehicle is present in that window. Many sub samples are taken and the results are aggregated and used to place a bounding box over the vehicle or vehicles in the original image. The results of this process are paired with the return from a radar sensor and used to determine the distance to each vehicle.

Chen *et. al.* designed a “direct perception” approach. As described by the authors “direct perception” is the mid-way point between reflexive neural networks and mediated perception systems. Rather than classifying objects and then using the object’s location in the image to determine it’s distance, the “direct perception” system classifies distances to objects such as cars or lane markings directly. This saves the programmer from having to perform one step of the processing pipeline needed for autonomous vehicle control [20]. The paper utilized real world data for training and then tested the system in the ‘Torcs’ simulation environment. This is an interesting contrast to our own neural network which was trained in a virtual world and tested in reality.

In 2016, NVIDIA researchers Marius Bojarski *et. al.* trained a reflexive network to drive a Lincoln MKZ [12]. NVIDIA's system was trained with real world-data, verified in a simulation with real world data and tested in the real world. To collect the large volume of training data needed to perform their research, Bojarski *et. al.* drove in a car for approximately 70 hours while recording video of the road and the actions of the driver. NVIDIA's work demonstrated the potential of end-to-end systems given modern processing technology.

Muller *et. al.* developed an off road car capable of obstacle avoidance using an end-to-end network and two input cameras to produce what is essentially stereoscopic 3D [13]. The system was trained on real world data and tested in real world experiments. Muller's work shows that neural networks are capable of learning to perceive depth from a two camera system but, the car had no path to follow and would wander aimlessly.

Our approach combines aspects of these works to extend existing knowledge. To the best of our knowledge, no prior work combines the use of a reflexive neural network trained using simulation data and tested in real world experiments.

Chapter 3: Neural Network

In this chapter we will define the architecture of the neural network that we designed. Discussion of the architecture will begin in Section 3.1. In Section 3.2 we will cover the data collection process and in Section 3.3, we will discuss the neural network training process. Section 3.4 will cover the neural network revision process and Section 3.5 will cover the validation process.

3.1 Architecture

Our overall neural network is a convolutional network, consisting of a convolutional network for image processing, a multi-layer perceptron network for past steering, a deep multi-layer perceptron network for merging image and steering, and a second, independent, convolutional network for braking. Modularizing the architecture to combine different networks allows tailoring of the network type for the different input data and needs. Additionally, the architecture enables easy additions for different driving tasks, such as a future expansion to support traffic lights and input from additional sensors. We used the Python and the Keras API to implement our neural networks [21]. Keras is a wrapper around Google's 'Tensorflow' neural network back end API.

Figure 4 outlines the overall hierarchy and the details of the smaller networks. Within each network box, the colored lines represent its layers with the line width representing the number of neurons and the color representing the type of network layer.

The system accepts camera images and the current steering angle as input and produces a steering angle as well as throttle. This structure of minimally processed sensor input data and control outputs is characteristic of an end-to-end network ar-

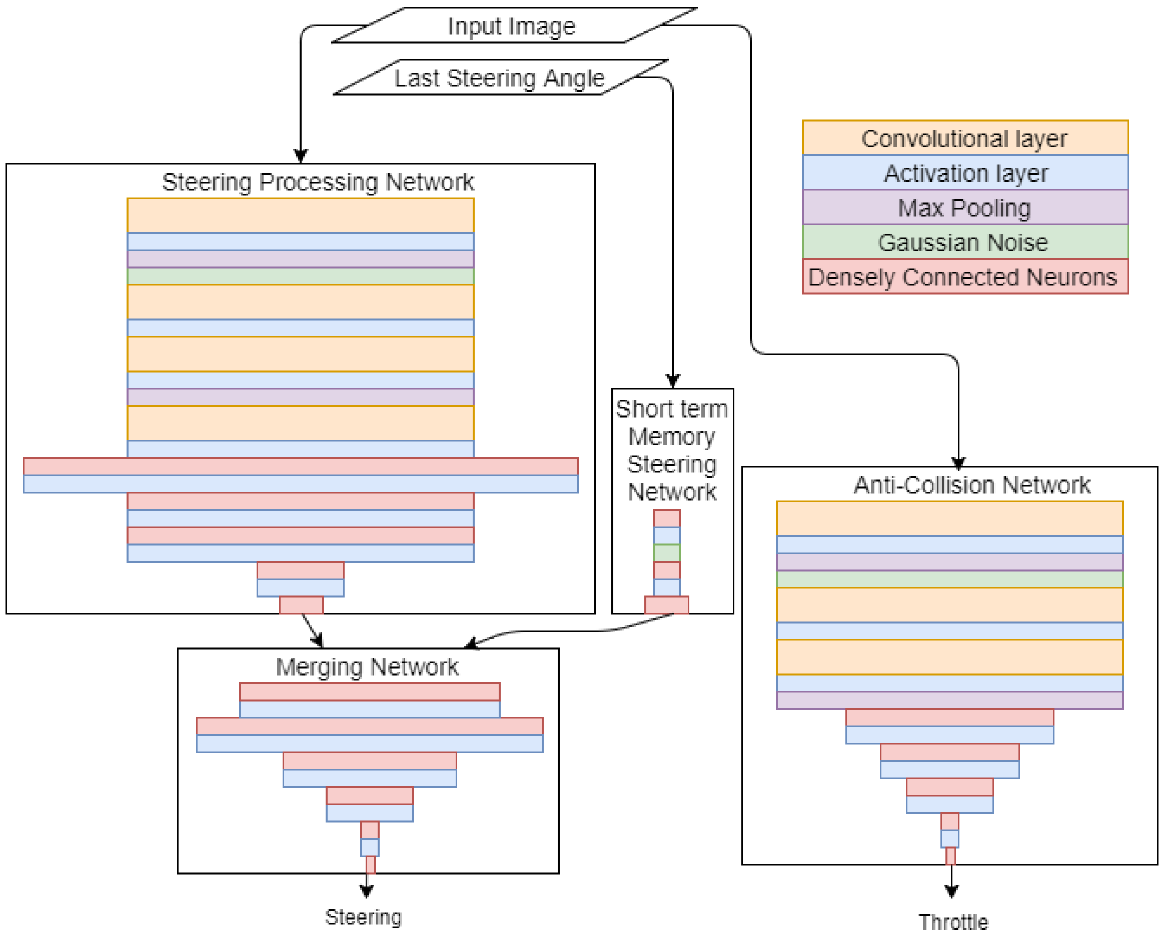

Figure 4: Overall Network Architecture

chitecture or a reflexive system. The alternative to reflexive systems is mediated perception in which case a neural network is used to classify objects in an image such as lane markings, vehicles and pedestrians for later use by another control algorithm. Reflexive systems require less human time to implement and maintain and less computing power to operate. These benefits are at the expense of operational clarity; the neural network is a black box that provides limited insight into how it will operate.

We have four network modules within the network: (1) steering image processing, (2) short term memory processing, (3) merging, and (4) anti-collision. The next sections describe each network in detail. We will begin Section 3.1.1 with a discussion of our first attempt to create a neural network driver. In Sections 3.1.2 through 3.1.5

we will discuss the finalized neural network architecture.

3.1.1 Initial architecture. Our first attempt to create a neural network featured a significantly different architecture from our final implementation. Following in the footsteps of Karpathy *et. al.* we initially implemented a neural network that leveraged the early fusion video processing methodology [6]. Early fusion is characterized by feeding multiple adjacent frames from a video stream into one branch of a neural network at the same time. Our intuition in this case was that by providing the neural network with multiple frames it would learn to infer the way that the car was moving. Figure 5 illustrates various methods of aggregating time series information in a neural network. The gray rectangles at the bottom of the figure represent frames from a video stream and the colored rectangles above represent the architecture needed to support the given fusion methodology.

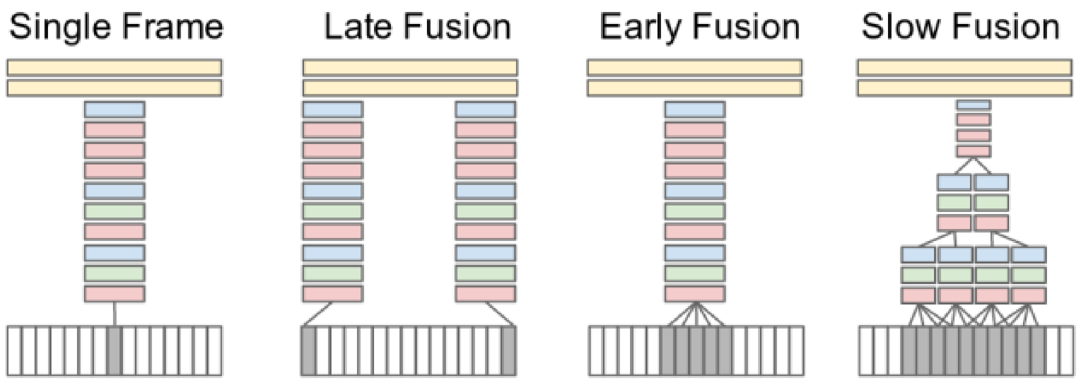

Figure 5: Temporal Fusion Methods [6]

Validation results on this neural network were not as good as we would have liked. The validation error rate at this stage was 2.5%. In order to improve the system, we considered what additional information humans use to drive that we could provide to the network. What we realized is that the neural network had no means of determining the current position of the steering wheel. The position of the steering wheel is critical to the task of driving but is a piece of information that human drivers do not consciously consider. We added recursiveness to our neural network by feeding the most recent steering classification back into the neural network. This approach is valid because the steering action at any given instant is closely related to the steering action at the preceding instant.

We performed a variety of tests on the initial version of our architecture which will be covered in detail in Section 3.4. The results of those tests lead us to implement the modular architecture described in Sections 3.1.2 through 3.1.5.

3.1.2 Steering processing network. We first need to learn from the camera images for which we use a convolutional neural network (CNN). CNNs are an effective method for image classification and are often used to identify objects in an image. However, in our implementation, the CNN identifies the steering angle necessitated by a given input. The neural networks that we implemented are primarily composed of three layer types: convolutional layers, max pooling layers and densely connected neuron layers. Convolutional layers are composed of a series of convolutional kernels that aid in pattern detection. Max pooling layers serve to reduce the size of the throughput data by simplifying a ‘pool’ of data points to a single value. This value is selected by taking the maximum value from the pool. Densely connected neuron layers are standard perceptron-style neuron layers in which each neuron in a given layer has a connection to every neuron in the subsequent layer.

The image processing network features several layers of convolution and pooling followed by several layers of densely connected neurons; the selection of the layers was optimized during development to provide the best results [22]. The input layer is a convolutional layer followed by a max pooling layer. After that, there are two more convolutional layers followed by a max pooling layer. Then there is one final convolutional layer before five layers of densely connected neurons. The layers contain 1500, 1000, 1000, 256 and 128 neurons.

By including a large number of initial neurons as well as large internal layers, the network has more connections, allowing it store and learn more information. The depth of this network allows it to learn more complex behaviors and the narrowing of the network towards the end allows for the consolidation of information as it flows through the network.

In addition to the number of layers and neurons, the network requires an activation function for each layer's neurons; this activation function determines the degree to which a neuron fires, or activates. For the steering network, the neurons in each of these layers use a 'relu' activation function. The term 'relu' is an abbreviation for rectified linear unit and it is an effective activation function for convolutional neural networks. Neural networking defines 'relu' functions as $f(x) = max(0, x)$ [23].

3.1.3 Short term memory steering network. An image classifier on its own could make an effective driver; however, knowledge of the current and past positions of the steering wheel provide valuable information to leverage while driving. To leverage this information, we create a multi-layer perceptron network dedicated to processing an array of previous classifications made by the network.

The steering short term memory (STM) branch is a smaller network than the image processing branch, requiring only a small amount of processing power. The image classification branch must handle the large, multidimensional arrays that represent images. The steering branch, however, is only responsible for handling a single array of at most 12 of the previous steering angles. The network starts with two layers of 12 densely connected neuron layers followed by a single layer of 128 neurons. This final layer is bigger than needed due to the requirements of the implementation environment. In order to merge the STM network with the image processing sub-network, all networks being merged must have the same number of neurons in the bottom layer [21]. Therefore, to match the image processing network, we expand the steering STM network to contain a 128 neuron layer. Finally, like the steering processing network, every neuron in each layer of this network uses ‘relu’ activation.

3.1.4 Merging network. Merging the two neural networks allows the resulting compound network to decide on its steering based on the greatest amount of useful data. The merged network serves to process the output of each of the two previously described networks. To achieve this, we use a deep multi-layer perceptron network consisting of six layers. After both of the previous networks have produced their output, the results are concatenated together and fed to the merging network as if it were simply input from a previous layer. In a sense, the three networks form a tree shape and act as if they are one network.

The bottommost layers of the image processing branch and the steering short-term memory branch both contain 128 neurons. Thus, the merged network starts with the outputs from the combined 256 neurons. The merged network then feeds through several layers of densely connected neurons. The first layer of the merged network contains 768 neurons followed by layers containing 1000, 512, 256 and 16 neurons. The final layer of the neural network contains a single neuron that outputs the final steering angle. Finally, all of these layers employ ‘relu’ activation except for the last layer of 16 neurons which uses linear activation. We change to linear activation for the final layer because ‘relu’ activation only allows for positive outputs and our network needs to produce values between -100 and 100 to cover left and right turns.

3.1.5 Anti-collision network. The braking network runs independently from the other three networks. This reduces the computational complexity, allowing it to respond promptly. It receives exactly the same input as the steering processing network. The braking network is a simpler network than the steering processing network as its sole job is to ensure the safety of the vehicle by applying the brakes when an object gets too close.

The braking network starts with a convolutional layer followed by a max pooling layer and a Gaussian noise layer to reduce overfitting while training. There are then two convolutional layers before a final max pooling layer. Next are five layers of densely connected neurons with 384, 256, 128, 16 and 1 neuron respectively. Just like the previously described merging network, the braking network uses 'relu' activation at all layers except for the second to last layer which uses linear activation. The benefit to the separation of this network is that it not only provides modularity but simplifies and accelerates the training process.

3.1.6 CNN implementation.

We implement the neural network using a Python API called Keras [21]. Keras is a neural network API that is capable of running on top of Tensorflow [24], CNTK [25], or Theano [26]. We chose to implement the neural network in Keras because of its usability and high-performance capability. We could have built our network using only Tensorflow or Theano, but Keras provides all of the same functionality with a much easier to use interface. Additionally, the Keras API provides a variety of popular activation functions including 'relu' and linear as mentioned previously as well as sigmoid, 'tanh', 'softmax', and others.

In this case, the Tensorflow back-end is used with the GPU acceleration feature enabled. GPU acceleration enables the very high throughput needed to support the task of autonomous driving. The laptop used in the real-world RC car test contains a Nvidia GTX970m, which allows us to utilize the GPU acceleration offered by Keras. Without GPU acceleration, we would not have been able to perform our real world tests because the neural network would take more than a second to process each frame on the CPU.

With our neural network implemented we were then able to collect a large training data set.

3.2 Data Collection Process

Although GTA V is well suited to our data collection needs, the default configurations unusable for our purposes. After modifying the camera controller and environment in GTA V we were able to implement a script to collect raw data.

3.2.1 Simulation justification In Section 3.2.1 we will discuss the aspects of simulation environments that make them good candidates for training data collection.

When addressing a task with a neural network, there are two criteria that must be met. First, the task must be transformed into a classification problem. Second, a source of training data must be acquired. For autonomous driving this can be a very expensive task. In order to get training data for a car, actual driving data must be taken so that the neural network can train and gain 'experience' by example. Many companies and research institutions with large budgets often opt to rig a car up with all of the required sensors, have a human driver operate the car and record all of the sensor data to use as their training set [12]. The first problem with this approach is financial. In order to speed up the task of data collection companies or research teams might opt to instrument multiple vehicles to collect data simultaneously. These vehicles are expensive to purchase and time consuming to build. Using a simulation can reduce or eliminate this cost. Second, collecting data on the open road deprives developers of the ability to control their environment. Using a simulation could allow for control of factors like weather, traffic conditions, time of day, road type and more.

One approach to finding a usable simulator for autonomous vehicle training is to build it yourself. This, however, can be even more time consuming and expensive than instrumenting vehicles for real world driving. A custom built simulator does have its advantages but for many developers finding a preexisting simulation is a better choice. To support our research, we began looking for simulators built for autonomous vehicle training. We quickly found that there are only small number realistic driving simulators built specifically for autonomous driving training. Unfortunately, those simulators are all proprietary. Even if they were open to public use, most simulators are build with mediated perception approaches in mind. What we turned to instead was video games. Open world driving games have become popular in recent years and have become highly realistic to improve the impressiveness of the gameplay experience.

One of our primary research questions is to determine the viability of using a simulation as a training environment for autonomous vehicles. What this question comes down to is how accurately a simulation is able to match reality. Driving is currently a human dominated task and humans are considered to be very good at it. If a simulator were to be so accurate that it could fool a human into thinking it is reality then that simulation would certainly be good enough to use as an autonomous vehicle training platform. Unfortunately, no such simulation exists. Since, no perfect simulation of reality is likely to ever exist the question becomes ‘can an imperfect simulation be effectively used to train a self driving vehicle?’. One reasonable place to start exploring this question is with virtual worlds that humans use for entertainment. Video games provide immersive and interactive environments which players can explore. One such video game is Grand Theft Auto Five.

Grand Theft Auto Five (‘GTA’) is an open world action action game published by Rockstar Games [1]. GTA V takes place in the fictional city of Los Santos which closely imitates the real-world city of Los Angeles, California. The game was released on the PC in April of 2015 and was commended for its rich environment and detailed graphics. GTA V also received several “Game of the Year” awards. GTA V is an ‘open-world’ game which means players can complete missions in whatever order they chose. It also means that players are allowed to freely explore the game world. As many open-world games do, GTA developed an online community of modders. ‘Modders’ are people who modify games in order in order to create a different gameplay experience. Common game modifications include adding missions, vehicles, weapons, new locations and graphical alterations. The existence of this community means that there are readily available tools to assist with the game modification process. These tools allowed us to create a controllable, consistent environment in which we can collect training data for our neural network [1, 18].

Using GTA V as a platform for data collection has a variety of benefits. First and foremost GTA V is cheap; a $30.00 investment is all it takes to secure a license of the game. This is an insignificant amount of money compared to the cost of a car and necessary sensors. Second, GTA is a highly realistic game. This game was designed to appease a community of online gamers with a keen eye for details and as a result the game is visually impressive. Although impressive, even at extremely high resolutions such as 4K (3840 x 2160) GTA V is not passable for reality by the human eye. However, computer vision systems do not require high levels of detail for tasks like following roads. In fact, most computer vision systems downsample images to as low as 20 x 20 pixels. ALVINN, the grandfather of autonomous cars, used input images of 30 x 32. The resolution of the images in the primary data stream fed into the neural network we have employed are 195 x 110. This resolution provides for significantly more detail than ALVINN's system was able to provide but it saves a significant amount of processor time as compared to a full resolution 1920 x 1080 frame. At a resolution of 195 x 110, GTA V is nearly indistinguishable from reality to the human eye. Because of this low resolution subsampling and the realistic in-game environment provided by Rockstar Games GTA V is an excellent candidate for data collection. What further strengthens GTA's case is the diversity of environmental factors that are controllable thanks to the game's modding community. Things like the number of cars on the road, the number of pedestrians walking around, the time of day and the weather are all controllable via game modification script.

3.2.2 GTA V modifications The first step of the data collection process was to modify GTA V. The most important modification that needed to be made was to the camera position and angle. The standard camera options available in GTA V are designed to be cinematic and are largely variations on a third-person perspective. Third person camera angles would be infeasible to reproduce on a real-world car so those options could not be used for data collection. There is one first-person camera setting available to players, but it places the camera inside the car where the head of the driver would be. We wanted to place the camera on the virtual car in the same place that we would position the camera on a real-world car. We decided to place the camera on the hood of the in-game car and the default camera controller was overwritten so that the camera would move and rotate as if it were physically attached to the car the same way it would be on any real-world test system that we might later use. The green circle in Figure 6 represents the placement of the camera on the in-game car. The camera was also tiled down 12.5°. When the camera was pointed parallel to the ground, the horizon line on the images taken from that viewpoint was approximately half way up the image. By tilting the camera down we were able to ensure that a greater area in the resulting image contained information about the road. We did not tilt the camera further down because we still wanted the neural network to view the environment around the road. We hoped that by providing some environmental information to the network, we would enhance its ability to regulate speed by allowing it to determine the differences between, for example, a residential road and a freeway.

Figure 7 depicts the difference in point of view before and after the camera is tilted down 12.5°. Image A shows the point of view after tilting downwards and Image B shows a neutral camera angle. The horizontal line in each image represents the horizon line. Notice that a significantly larger portion of Image A is below the horizon line. By angling the camera down, we are able to increase the amount of space that the road takes up in our images and by extension, information density in our training images.

During the development of the neural network the number of pedestrians was decreased to the lowest available settings and the number of vehicles on the road was also reduced to the lowest available setting. This allowed for data collection in a less chaotic environment and resulted in a data set with fewer variables. Later in the

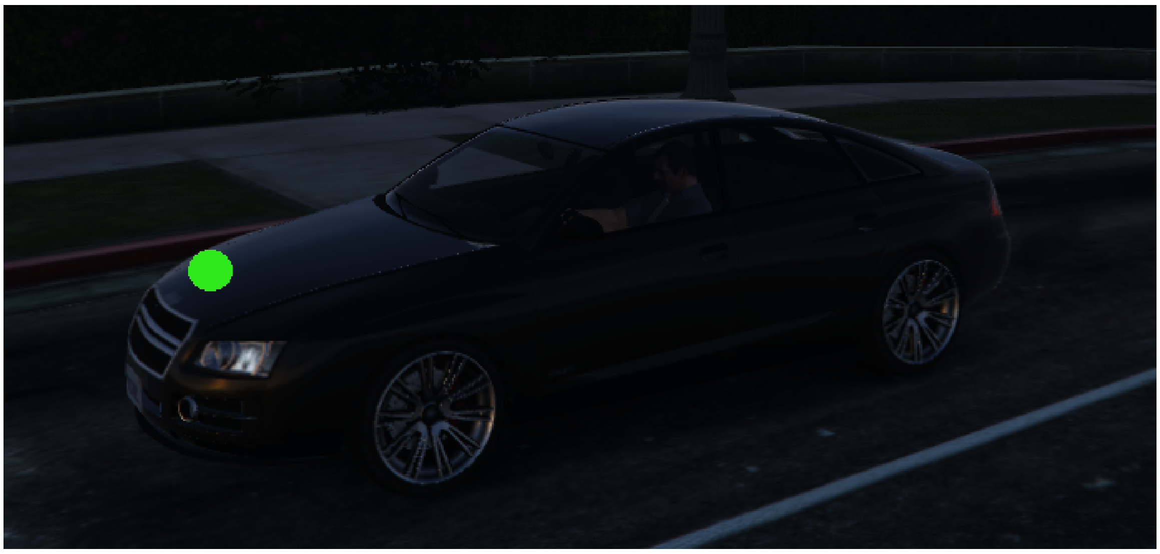

Figure 6: Camera Position on In-Game Car

development cycle, once the viability of the neural network driver had been verified, these values were changed to their highest setting. This was done to ensure that the neural network had experience driving with other cars on the road under high traffic conditions.

The final modification was to set the in-game player's 'wanted level' to a static value of 0. If the player's wanted level increases past 0 they start to be chased by the police. By setting the wanted level to 0 and preventing it from increasing we were able to eliminate the need for the researcher to reload the game in the event that they make a mistake what would cause them to be chased by the in-game police.

The game's built-in day and night cycles were left intact in efforts to imbue the neural network with some degree of light invariance. The weather cycle was also left in place. The weather conditions that are present in the training data include varying levels of sunniness, cloud cover and light rain.

3.2.3 Data recording. Data collection is performed by having a researcher drive around the virtual world of GTA and recording the video frames and control

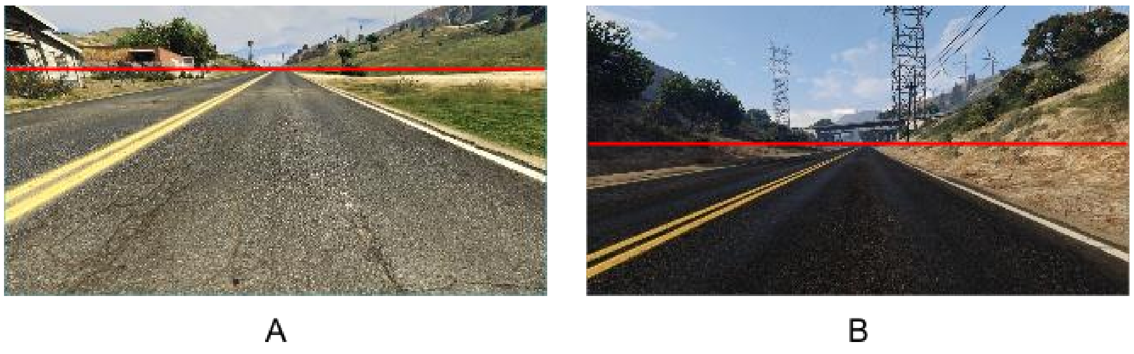

Figure 7: Camera Tilt Comparison

inputs of the researcher. As the researcher drives, a Python script runs in the background to record data. First, the script takes a screenshot of the game window using the ‘win32gui’ package and downsamples it to a resolution of 640x320. This image is immediately written to a .JPG file and stored for later use. The name of the .JPG is generated using the driving session identifier and the frame number within the driving session. Immediately after the screenshot is taken the ‘pygame’ API is used to poll for the angle of the steering wheel, the gas pedal activation level and the brake pedal activation level. Pygame is a Python API that provided utilities for making games, including easy access to controller inputs. After being collected, the steering, gas and brake values are then appended to a .CSV file along with the numeric identifier associated with the image.

Because the goal of the neural network is to follow the road that it is on, the researcher driving the in-game car was not allowed to execute road transitions at intersections while recording. To allow the driver freedom to drive around the city a button on the controller was programmed to allow the driver to pause recording while the button was held down. This flexibility prevented the driver from getting stuck on one road. In addition, before being written to a file, all frames were stored in a buffer for approximately 10 seconds. This design decision was made so that another button on the controller could be programmed to delete all of the frames in the buffer. This afforded the researcher the ability to delete the last 10 seconds of driving in case they made a mistake.

3.2.4 Input hardware. Originally, a gamepad controller was used as the researchers input mechanism. Gamepad controllers are considered by the gaming community to be preferable to keyboard input for driving games. While this is true, using a gamepad for scientific data collection leads to an unacceptably noisy dataset. This is because analog sticks can be mechanically difficult to operate accurately. Very slight thumb stick movements can result in dramatic steering changes. The creates a tendency for the car to swerve back and for on the road as the driver continuously over-steers and over-corrects. Upon discovering the shortcoming we decided to upgrade to a racing wheel controller. The racing wheel used for data collection clamps onto a table which provides a familiar driving experience for the operator. There are also foot pedals on the floor to more accurately imitate a car-like environment. The steering wheel allows for very fine steering adjustments which eliminate the jerky steering tendency of the gamepad. As a result, the amount of noise present in the data was significantly reduced.

3.2.5 Data augmentation process. The simulation environment provides a reasonable set of images. However, to increase the amount of data as well as the different experiences, we augment the data. This allowed us to transform one training drive in simulation to the equivalent of fourteen different drives by creating viewpoints of driving the car closer to the different lines on the road.

Figure 8 shows the data augmentation process and resulting set of seven images. We first resize the 640x320 images to a resolution of 160x90. Then the center 90x90 square of the image is selected and given the same label as the original image. The next step is to shift the square to the left or right five pixels, selecting and creating new images with each shift [11]. For each five pixels that the image is shifted to the left or right, the labels are incremented or decremented by 0.75. This augmentation

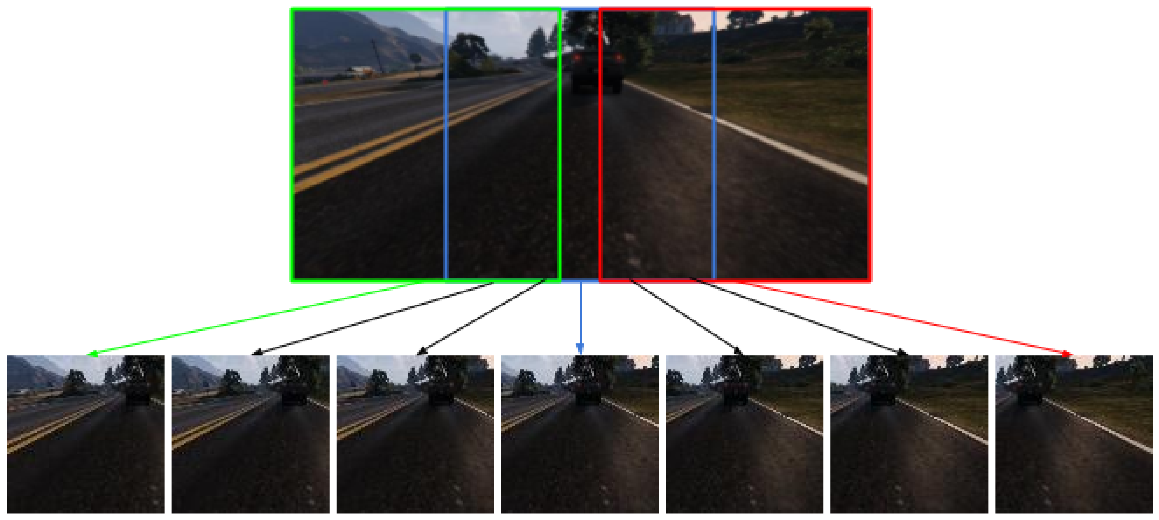
Figure 8: Data Augmentation Process

process provides the network with the ability to handle situations outside of normal centered driving, allowing it to better correct for its mistakes. The data set used to train the neural networks was the result of approximately five hours of driving and contained 35,000 images. Using the data augmentation process, the dataset expanded to 490,000 images. The augmentation process essentially gave the neural network 70 hours worth of training data.

In addition to providing new new viewpoints on which to train, this system can also be repeated during operation of the RC car. By cropping out 1:1 images from the 16:9 image provided by that car's camera, multiple different classifications can be made and then used to determine the final steering angle.

3.3 Training

The training process for our neural network is identical to the standard process for training back-propagation neural networks. Each branch of the steering control network receives its input simultaneously and the entire network produces an output prediction. If the prediction does not exactly match the label the weights of the inter-neuron connections are modified accordingly and proportionally. This process is repeated for the entire dataset. Since the steering control network and the anti-collision network do not interact they are trained interdependently. One iteration through all of the training data is called an 'epoch'. For a complex task such as driving it will often take between fifty and one hundred epochs before the networks become proficient. The entire training process is automated by the Keras machine learning package. Keras leverages Google's Tensorflow technology to create networks and accelerate them using GPUs. This network was trained using a GTX970 GPU.

3.4 Architectural Development

This section will cover the tests that we performed on the first iteration of our neural network described in Section 3.1.1.

3.4.1 Parameter analysis. Before we commenced with our on-track tests, we needed to optimize the network to determine the optimal input configuration. The initial neural network used to drive the remote control car could handle a multitude of input data configurations. As previously stated there are two primary pieces of input data: images and past steering angles. The images and steering angles are fed into the network as arrays. Let us call the most recent image frame n and the number or previous frames j. The set of images provided to the neural network can be defined as n to $n - j$. The same can be said for the previous steering angles. Let us call the most recent steering angle m and the number of previous steering angles i. The set of steering angles provided to the network can be defined as m to $m - i$. The purpose of parameter analysis is to determine the values of i and j which result in the most effective driver.

To keep the architecture of the network static, the range of values for j were limited to $\{2 \ldots 8\}$. As discussed in the network architecture section, the neural network leverages three-dimensional convolution as a means of data dimensionality reduction. Due to the nature of three-dimensional convolution, reducing j below 3 will result in an invalid data configuration partway through the network. The upper bound is set at 8 because any value larger than 8 will result in 'out-of-memory' errors on the systems used to run the networks. The values tested for i are defined by the set $\{1, 2, 4, 6, 8, 10\}$.

Tests were performed by training a network with each of the possible (i, j) combinations for 20 epochs and then testing each network's validation performance in terms of accuracy and throughput. Accuracy is given in terms of average steering error. Accuracy is the most obvious metric to consider, but data throughput is nearly as important. The longer it takes the network to classify a given data point the further the car will have traveled during processing and the less relevant the resulting control action will be. The training dataset consists of 8167 data points and the testing set consists of 1843 data points. There is no overlap between the two sets. Each accuracy score is the average of five tests. It is worth noting that this is a small test by machine learning standards and network architecture changes may yield different results, especially as j approaches 8.

The clearest result of this testing is that i has no appreciable effect on processing time. The processing time was constant across all values of i for each value of j. This is not surprising seeing as each image contains $160 * 90 * 3 = 43,200$ values and at most the past steering angles will contribute 10 values. Figure 9 shows the relationship between the number of frames per data point and the time needed to process a given data point. Since the processing time is constant for all values of i at a given j value, the graph has been limited to include only j.

At $(i, j) = (4, 4)$ the network achieved its lowest error rating and processing time is fairly low. It somewhat defies intuition that more data does not correlate to better accuracy, but there is only some much knowledge that can be stored in a network of a given size. At $(i, j) = (4, 4)$ this neural network reaches its balance point. Figure 10 shows the average error rate for all values of i at a given value of j. From this chart we can see the four frames per data point offers the lowest error rate.

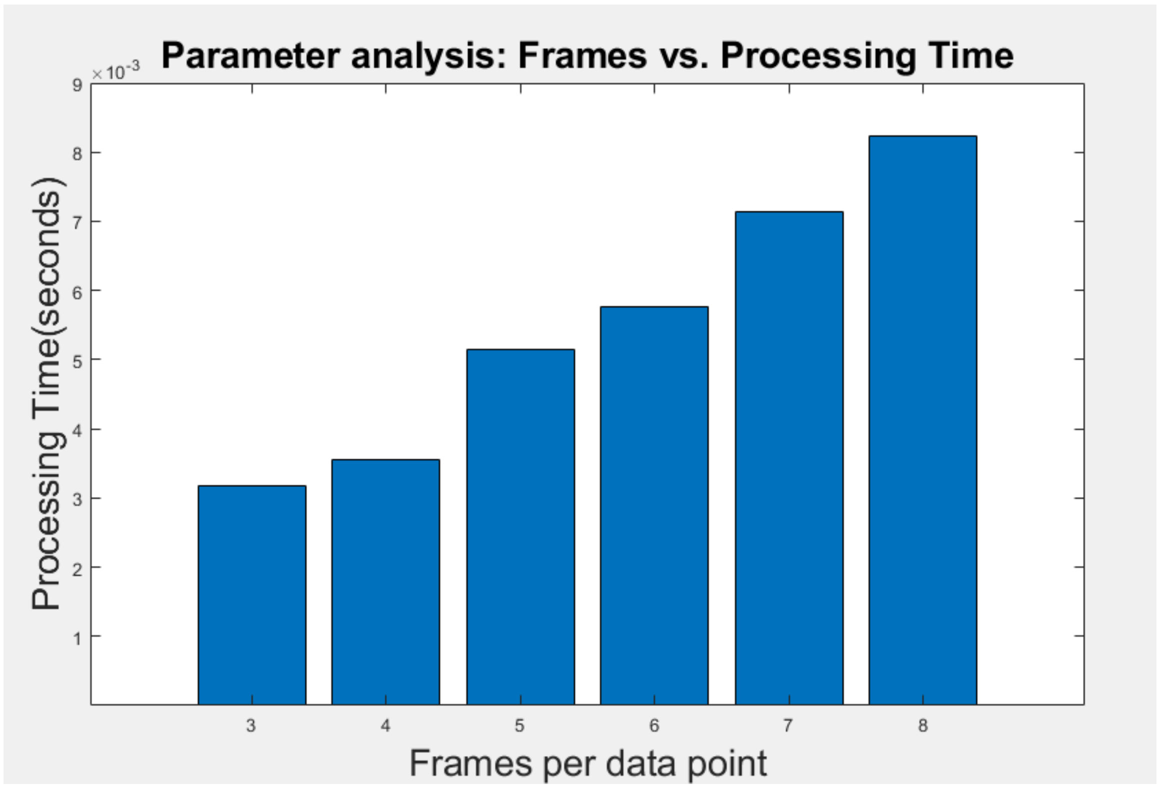

Figure 9: Frames per data point with respect to Computation Time

3.4.2 Analysis. The early fusion methodology proved to be a GPU memory intensive approach. Limited GPU memory and large input data sets meant that the neural network needed to be shallower to fit into GPU memory. This early fusion network with four frames performed well in validation tests with a 2.2 percent error rate. Once implemented on the RC car, the neural network proved ineffective, unable to successfully complete a single lap of out test track. One possible reason for this performance deficit is the slow movement speed of the RC car. Since the car moved so slowly, each of the input images were similar and thus each successive frame provided little to no additional useful information. Another possible reason is that too much GPU memory was used for managing the large volume of input data. This meant that the underlying perception style network needed to be very small to fit into the GPU's memory. The reduced neuron count of the underlying network limited the classification ability of the entire network.

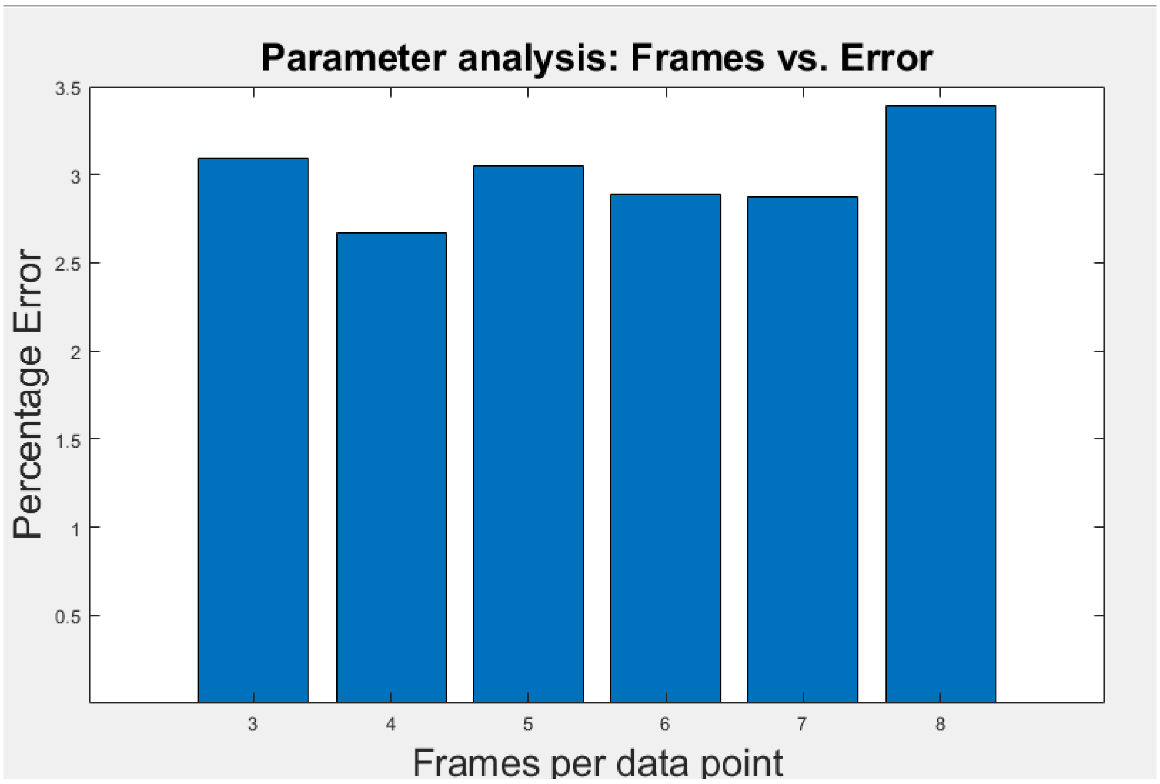

Figure 10: Frames per data point with respect to error rate

One of our concerns about the RC car performance stems from the video streaming system that we employed on our RC test system. Streaming video introduces a large amount of control latency as frames need to be streamed via wifi to a laptop. This delay meant that our control inputs would always be late and is what necessitated the slow movement speed mentioned above. The longer the delay, the less relevant our control inputs are. The addition of long computation time can exacerbate this issue. The problem that arises from this situation is that the RC car tends to get itself into situations that it had never seen in training by driving too close to the edge of the track. Once in an unfamiliar road position the performance of the car could become unpredictable.

3.4.3 Revisions. To address our concerns we altered our neural network architecture to use only single frames. This allowed for deeper neural networks to be loaded into GPU memory, which in turn allowed the network to learn a wider variety of behaviors. It also allowed us to implement the data augmentation methodology described in Section 3.2.5. After learning from our mistakes, we were able to design and implement the neural network described the Section 3.1.

3.5 Validation

After completing the training process, we validated the neural network. The data used for validation was collected from GTA V in the same manner as the training data. Validation provides an intermediate step between training and real world testing by allowing us to test the accuracy of the neural network on data similar to the training data. During validation testing, the neural network classified 10127 augmented images to which it had not been exposed during training. Using the existing labels, we computed the mean absolute steering error between the network's calculated steering angle and the labeled steering angle. In validation testing, the network used for track testing produced a mean absolute error of 1.9%. By achieving a error rate this low, we can be fairly confident in the neural network's ability to mimic the behavior demonstrated by the human driver during data collection.

3.5.1 Road type analysis. To gain a deeper understanding of the capabilities of the neural network we collected additional validation data and evaluated the neural network's performance on four different types of road. An example of each type of road is provided in Figure 11. The difference between dashed center line, double center line and no outer white line roads is the way that the lane boundaries are marked. The freeways, however, feature two, three or four lanes going in each direction with dashed white lines separating the lanes.

Figure 12 illustrates the performance of the neural network across the four road types that we tested. Each point in Figure 12 represents the mean error rate on the respective road type. The error bars around each point represent one standard deviation. The mean values are specifically enumerated in the table in Figure 13. The error rates are fairly consistent across all of the road types except for the road

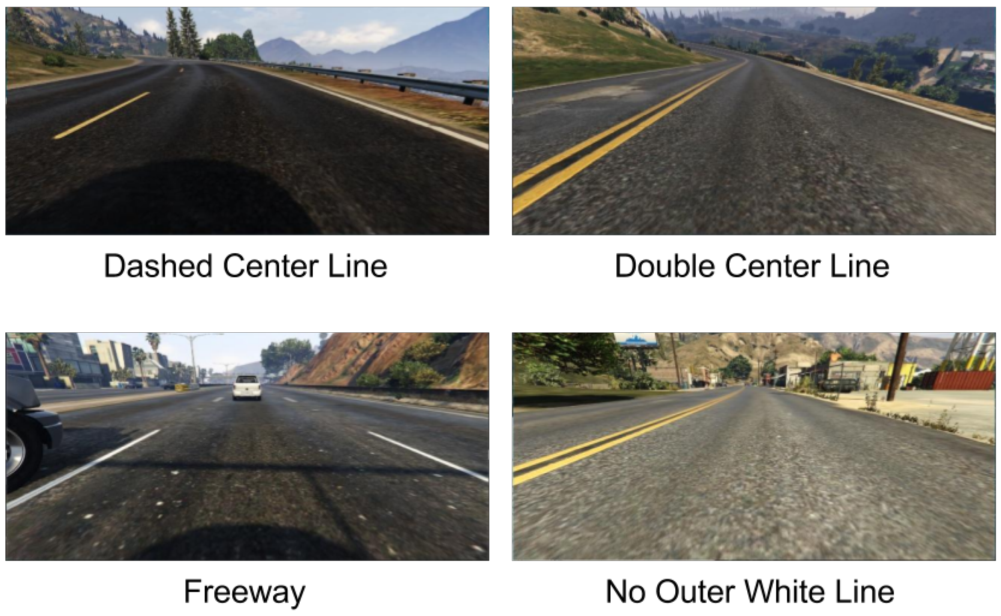

Figure 11: In-game road types

without outer lane markings. The error rate on roads without outer lane marking is roughly four times higher than the three other road types. This discrepancy indicates to us that our autonomous vehicle control system performs significantly better on roads with outer lanes and needs improvement to drive safely on roads without outer lanes. Because positive steering values correspond to right turns, negative steering values correspond to left turns, and the error rates across all road types are negative, we can conclude that the neural network has a leftward bias. This may be due to the fact that the the charcoal gray color of the pavement and the yellow color of the road markings creates a significant contrast that is easily identified by the neural network.

Figures 14, 15, 16 and 17 extend this analysis by illustrating the specific error rates on each road type.

Figure 14 depicts the neural network's error rate on each of the validation frames with dashed center lines. When turning left on roads with dashed center lines the neural network had an average error of -1.35% per frame. When turning right the neural network exhibited an error rate of 1.27% per frame. In Figure 14 we can see

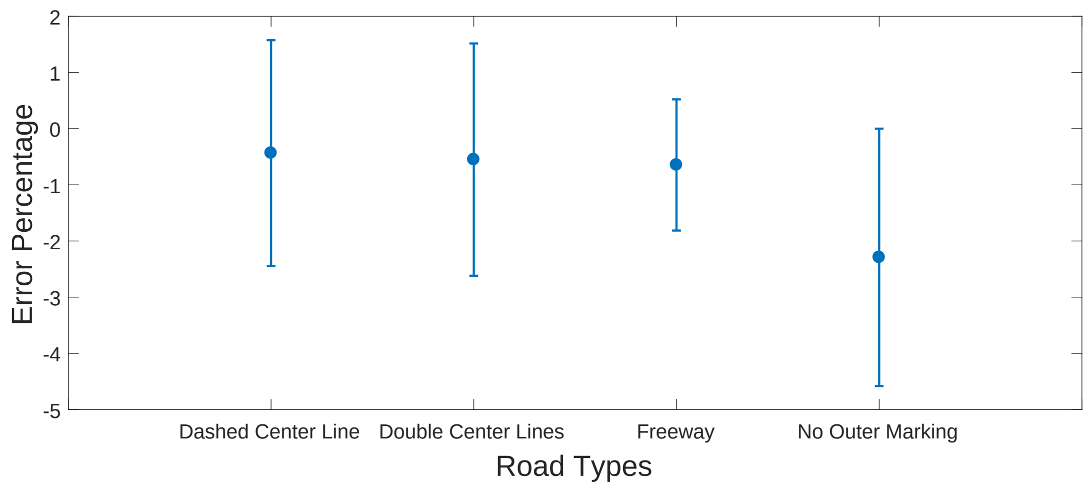

Figure 12: Mean error percentage with respect to type of road

Dashed Center Line	Double Center Line	No Outer Line	Freeway
-0.434	-0.550	-2.290	-0.646

Figure 13: Mean error percentage with respect to type of road

that there is a slight increase in error rate in the first 400 frames and another, more significant, increase in error rate from Frame 1100 to Frame 1600. The first 400 frames take place at night, which likely accounts for the slight increase in error. The second cluster of high error rates towards the end encompasses a group of frames that take place at sunset when the car is pointed almost directly at the sun. The sunset causes these frames to have a significant orange tint and the angle of the vehicle relative to the sun means that there are reflections of the sun on the road. The combination of these two characteristics creates oddly bright frames that are most likely the cause of the sharp increase in error rate. Collection of additional training data under similar conditions would likely correct for this performance deficiency.

Figure 15 depicts the neural network's error rate on each of the validation frames with a double yellow line. When turning left on roads with double center lines the neural network had an average error of -1.35% per frame. When turning right the

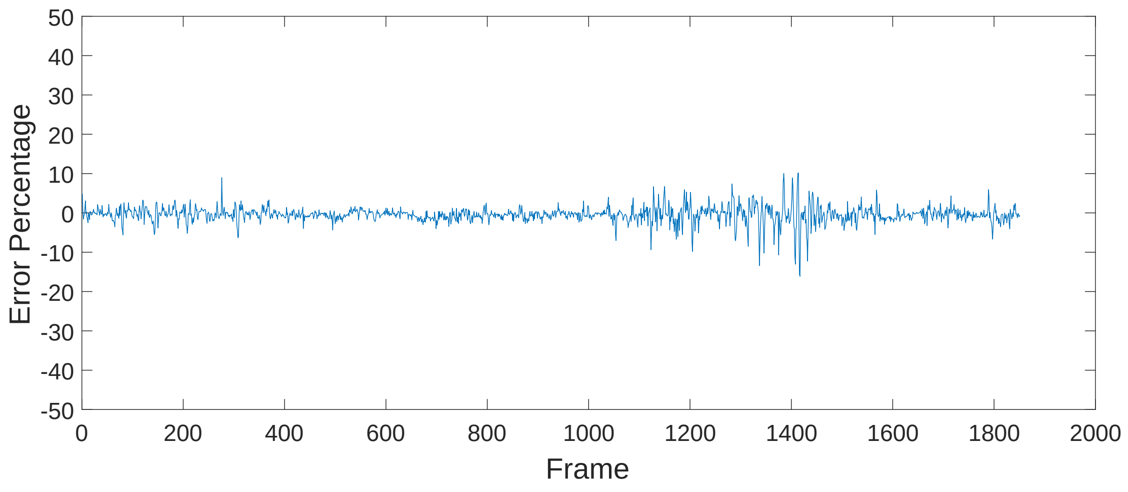

Figure 14: Error percentage by frame on roads with dashed center lines

neural network exhibited an error rate of 1.41% per frame. Similarly to Figure 14, Figure 15 exhibits an increase in error rate in the first 500 frames and another around the 5000th frame. Once again, the first spike in error rate is likely due to the fact that it is nighttime in those images and the road is poorly lit. However the large spike around the 5000th frame is likely due to the color of the road. The frames around Frame 5000 cover a section of road which is a very light gray, unlike the typical charcoal color of most roads. The pavement is so light that is difficult to discern an edge between the yellow line in the center of the road and the road itself. The neural network likely relies on the contrast between the color of the line and the color of the road to identify the lane.

Figure 16 depicts the neural network's error rate on each of the validation frames with no outer white line. When turning left on roads with no outer lines the neural network had an average error of -3.13% per frame. When turning right the neural network exhibited an error rate of 1.90% per frame. Both of these error rates are substantially higher than the previous two road types. It is also interesting to note that the boundary line on the right side of the road has a significant effect on the neural network's ability to accurately classify a left turn. This is likely due to the fact

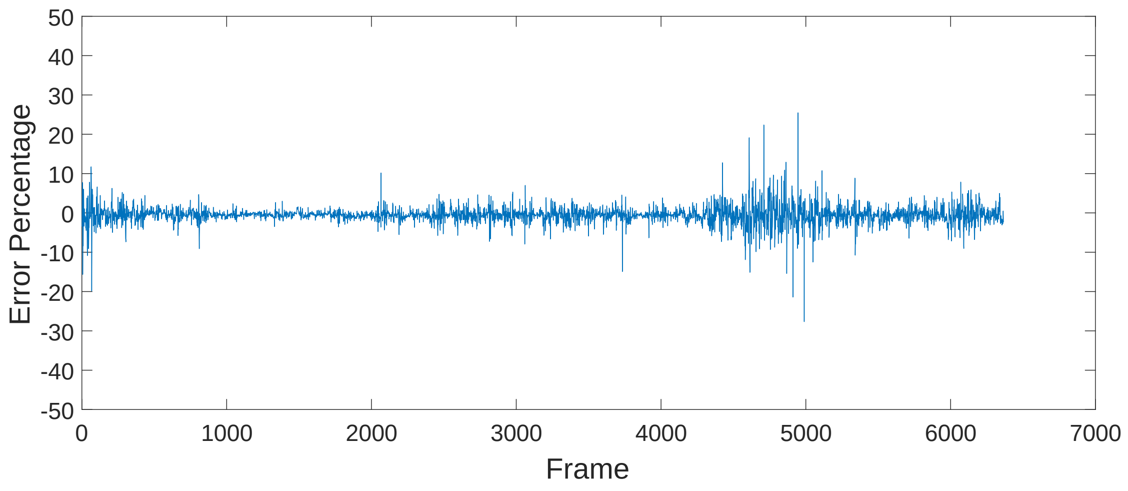

Figure 15: Error percentage by frame on roads with double center lines

that the neural network had learned to use the distance from the outer white line as a reference point.

Figure 17 depicts the neural network's error rate on each of the validation frames which take place on a freeway. On freeways the neural network exhibited the lowest error rates on both left and right turns. On left turns on freeways the neural network showed an error rate of -0.65% and on right turns an error rate of 0.72%. These low error rates are most likely due to the fact the freeway driving generally only involves gentle turns and minor corrections.

After completing the design of our neural network and verifying its ability to navigate we moved on to real-world testing. In Chapter 4 we describe the tests that we performed to quantify the capabilities of our neural network control system.

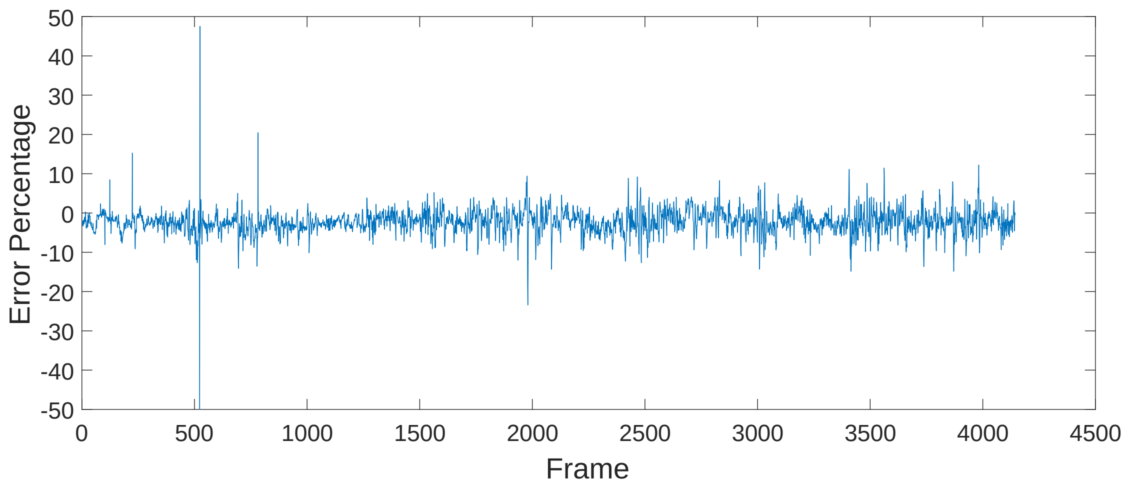

Figure 16: Error percentage by frame on roads with no outer white line

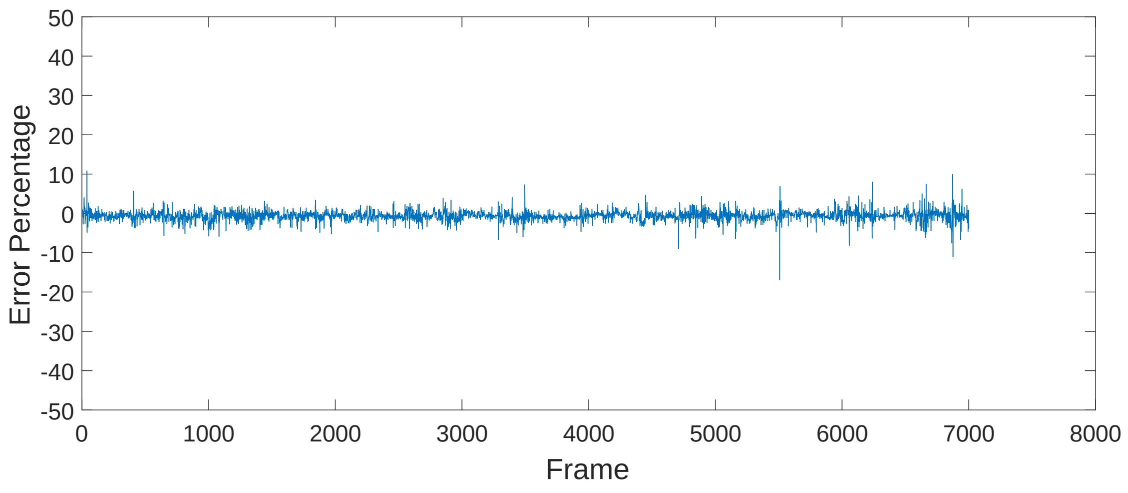

Figure 17: Error percentage by frame on freeways

Chapter 4: Testing and Evaluation

In this chapter we will discuss the various tests performed to tune and characterize both the RC car system and the neural network. This chapter will begin with a discussion of the instrumentation of the RC car in Section 4.1. Section 4.2 will cover track tests including track design, sensitivity tuning, performance testing, throttle characterization and anti-collision testing.

4.1 Remote Control Car

To verify the neural network in a real world scenario, we instrumented and modified a RC car, which we then connected to the neural network running on a laptop. The RC car allowed us to build a scaled down environment to test multiple road types and turns that are not feasible on a larger car. The car is not large enough to carry the computational system and the batteries needed so we modify the remote control to communicate with the car as well as with a laptop running the neural network. Figure 18 shows the different system components and how they connect.

We used a Latrax 1/18th scale rally RC car; it has a reasonable cost and provides the correct style of remote controls for easy instrumentation. To allow the car to drive at slower speeds and not stall in corners, we designed and 3D printed new wheels that reduce the gear ratio. The RC car utilizes a SainSmart Wide Angle Camera with a 160° wide angle lens. The camera captures images at a resolution of 200x116 using UV4L and runs at a framerate of 40fps. The wide angle lens allows for more information to be captures per frame and the relatively high frame rate allows for frequent control updates. The camera attaches to a Raspberry Pi Zero W that streams the images over WiFi to the laptop.

The laptop accepts the images over WiFi and keeps track of previous steering

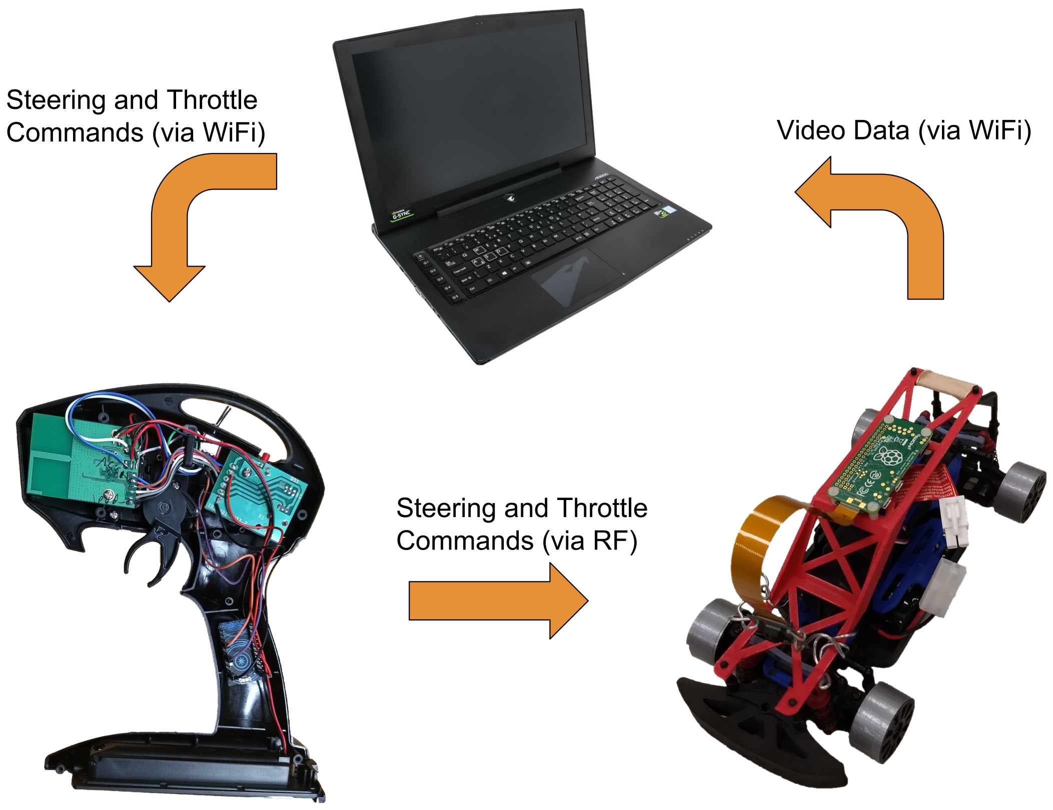
Steering and Throttle Commands (via WiFi)
Video Data (via WiFi)
Steering and Throttle Commands (via RF)

Figure 18: System Overview

output locally. After receiving the source image, the laptop scales the image to 160x90 and then generates two additional images. One image is taken from 10 pixels to the right of center and the other is taken 10 pixels to the left. The steering neural network then classifies each of the images as inputs and computes the steering angle. By averaging two classifications from different points of view, we can widen the field of view and reduce the effects of errors in a single classification. The anti-collision network is provided with a center 90x90 crop of the original image to maximize its ability to spot oncoming vehicles.

This steering angle provided by the neural network is a percentage ranging from -100 to 100. However, the RC car system requires a value between 0 and 4095 with 2047 corresponding to pointing the wheels straight ahead. Translating between the two values is challenging due to the differences between the RC car and the GTA car, including angle value requirements, wheelbase, and steering rate. As such, we need to tune the sensitivity of this translation. The translation is made by multiplying the neural network output by the absolute value of itself and then multiplying by a static tuning amplification constant. Despite the added tuning, we decided on this translation layer to allow the network to control different vehicles. However, the translation process will need to be tuned to each specific vehicle system.

Throttle control operates slightly differently. Because we have a video transmission delay due to WiFi, it is important that the car moves slowly while driving. During track testing, the throttle was set to a static value so that the car would not stop anywhere on the track or accelerate to a high speed. During anti-collision testing, the neural network is allowed to control the throttle unless the throttle output drops below a threshold determined by our throttle characterization tests, in which case the speed is reduced to 0. This way the car will stop if it gets too close to another vehicle.

The laptop then streams steering and throttle commands over WiFi to a Photon microcontroller in the existing RC remote control. The Photon uses two DAC outputs to bypass the existing manual controls and provide steering and throttle values to the car.

4.1.1 Experiment setup and methodology. Once the RC car system was functional, we designed a track and experiment methodology to test the network. We performed four experiments using the RC car system: (1) tuning of the sensitivity of the control system, (2) validation on the complete track of the system performance and stability, (3) characterization of the throttle output, and (4) validation of the anti-collision performance.

4.1.1.1 Track design. The test track that we developed for testing consists of multiple road and turn types as seen in Figure 19. The track features three road types: (1) two lanes each direction with a double yellow line running down the middle,

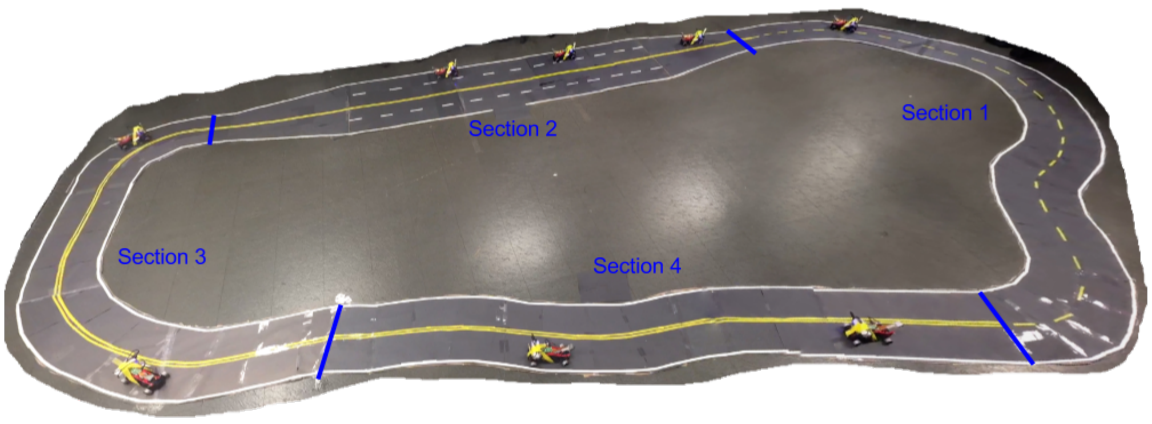

Figure 19: Test Track

(2) single lane each direction with a double yellow line going down the middle, and (3) a single lane each direction with a dashed yellow line going down the middle. There are also a variety of turn types including a long straight, gentle s-bends, wide s-bends, sharp turns, and turns with varying angles. These track conditions were chosen to test the robustness of the control system across a variety of common road types. Multiple vehicles are shown in Figure 19 to illustrate the size of the track relative to the test vehicle.

We subdivided the track into four sections as labeled in Figure 19 with each section containing a specific road type. Section A is composed of Type 1 shown in Figure 20. Section B is composed of Type 3 while Sections C and D are composed of Type 2.

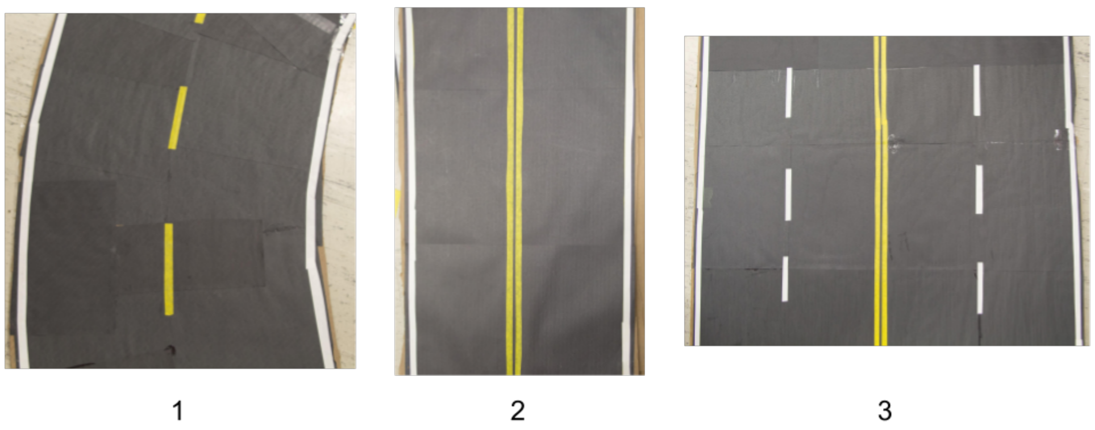

Figure 20: Road Types

4.2 Track Tests

Real-world track testing is one of the most vital parts of our research process. In this section we will present the testing methodologies for four different performance characteristics.

4.2.1 Sensitivity tuning methodology. The goal of control sensitivity testing was to find the sensitivity setting that results in the smoothest possible driving. The sensitivity tuning process consisted of two phases. The first phase determined which sensitivity settings to analyze further and the second phase tested those settings under stricter conditions.

First, the preliminary testing process allowed the car three trials to complete a lap of the track. If the car completed any of these trials, the car passed that setting. If the car was unable to complete any laps, the car failed that setting.

For the settings that the car passed, the second phase of the tuning process determined the smoothness and the safeness of the viable sensitivity settings. In these tests, the car drove around the track five times at its lowest speed. We tracked the number of instances where either of the front two wheels came in contact with a lane marking. If the left wheel of the car touched the middle yellow line or the right front right wheel of the car touched the outer white line, we added a point to that trial's score. If the car committed some sort of unrecoverable failure such as completely crossing the yellow lane divider or driving off the track so far as to lose sight of it, then that trial received an automatic score of one hundred points. We used the sensitivity setting with the lowest score for the performance testing.

Figure 21 illustrates the different scenarios for point deductions. Image A of Figure 21 represents a neutral road position, maintaining this road position would result in no deductions. Images B and D represent the conditions that constitute touching the yellow or white line, respectively. Images C and E represent the conditions that constitute crossing the yellow or white line, respectively. Please note that the rear wheel positions do not count towards scoring.

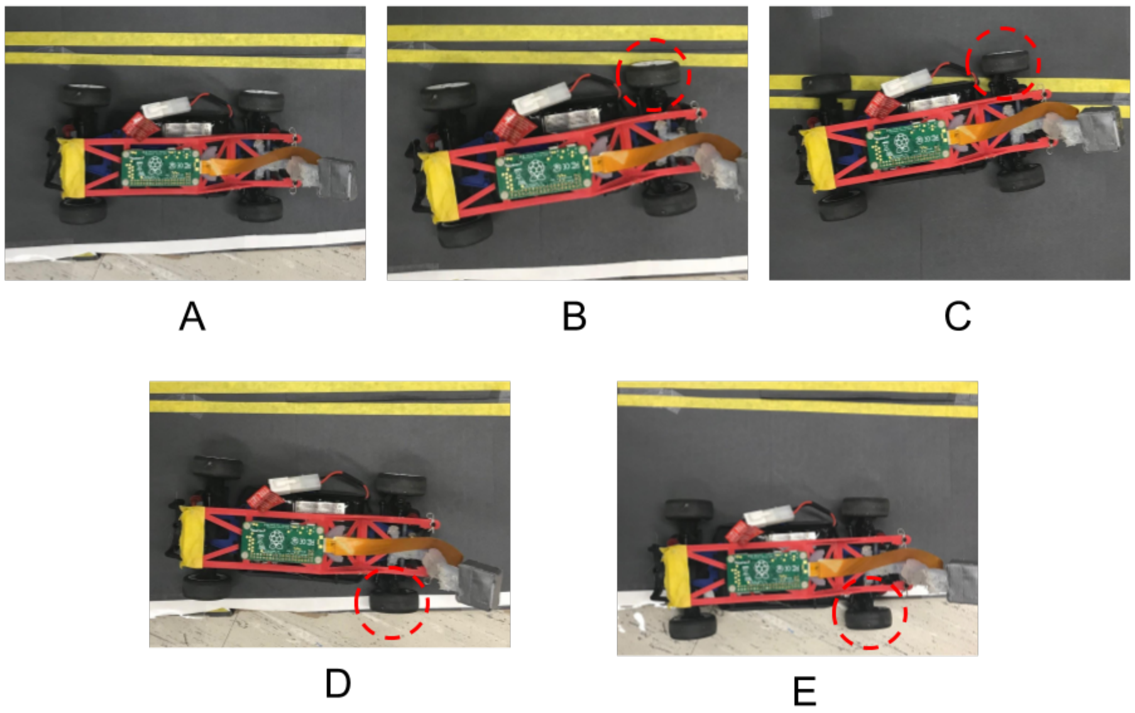

Figure 21: Point Deduction Scenarios

4.2.2 Performance methodology. The goal of the performance testing was to determine the robustness of the navigational capability of the steering control neural network. In track testing, the car attempts to drive 100 counterclockwise laps of the track. To ensure accurate assessment of neural network performance, errors due to video streaming failure or the RC transmitter were not counted against the system as these would not be present in a full size implementation. The final metric of robustness is the number of laps that can be performed before committing an unrecoverable error (crossing the yellow lane divider or exiting the track). In addition, the track was split into four sections; the number and type of errors were recorded for each section. The errors include touching either the center or outer lines with a wheel and crossing either the center or outer lines with a wheel.

4.2.3 Throttle characterization methodology.

To evaluate the anti-collision system, we first characterized the throttle output from the anti-collision network when another car is placed at various distances in front of the autonomous RC car. The lead car was placed at discrete one inch intervals from the front of the autonomous car ranging from 1 to 20 inches. During this characterization test, the autonomous car was not moving and the result was taken directly from the neural network's output.

4.2.4 Anti-collision performance methodology.

After characterizing the anti-collision network performance, we performed an experiment to assess its capabilities while driving. The RC car was placed on our track six feet behind another car on the track. The second car remained stationary for this test. The RC car started driving and approached the stationary car. When the RC car stopped due to the anti-collision network, we measured the distance between the two cars. We performed 20 trials of this test.

4.3 Results

After determining how to test the system and the appropriate metrics, we performed all experiments.

4.3.1 Sensitivity results.

The first phase of sensitivity testing resulted in a sensitivity multiplier of 25. There is a clear lower bound for the steering multiplier. When the steering multiplier was set to 11, the car was unable to turn hard enough to complete many of the turns. However, at 12, the car successfully completes a lap of the track on its first trial. The range of values that allow the car to successfully lap the track range all the way up to 200. However, at any setting above 100, the behavior of the car clearly becomes erratic as the car continuously turns too hard and then over-corrects.

Having determined the range of viable sensitivities, the second phase of sensitivity testing started at the minimum viable setting of 12. We next tested at a setting of 15 and incremented the steering multiplier by five until we reached 60. Figure 22 demonstrates the number of errors made at each sensitivity level. The settings of 25, 30 and 35 all performed well, scoring an average of 2.2, 2.6 and 2.4 points respectively. Above 35, though, a trend began to emerge. The car began to make jerky movements and would over-correct for its mistakes. Each setting after 35 performed worse, resulting in a higher score than the previous setting. Once the steering multiplier reached 60, we concluded testing; at that point the car earned an average score of 8.6.

The multiplier setting of 25 provided the best result with the lowest score of 2.2. We used this value for our performance testing. Although the performance did vary across the settings tested, all of these settings, other than the first, were able to successfully complete multiple laps around out test track. This is possible because the neural network is able to correct for its errors if it happened to steer too far. This behavior demonstrates a high level of flexibility and indicates that the same control system could be applied to other vehicles.

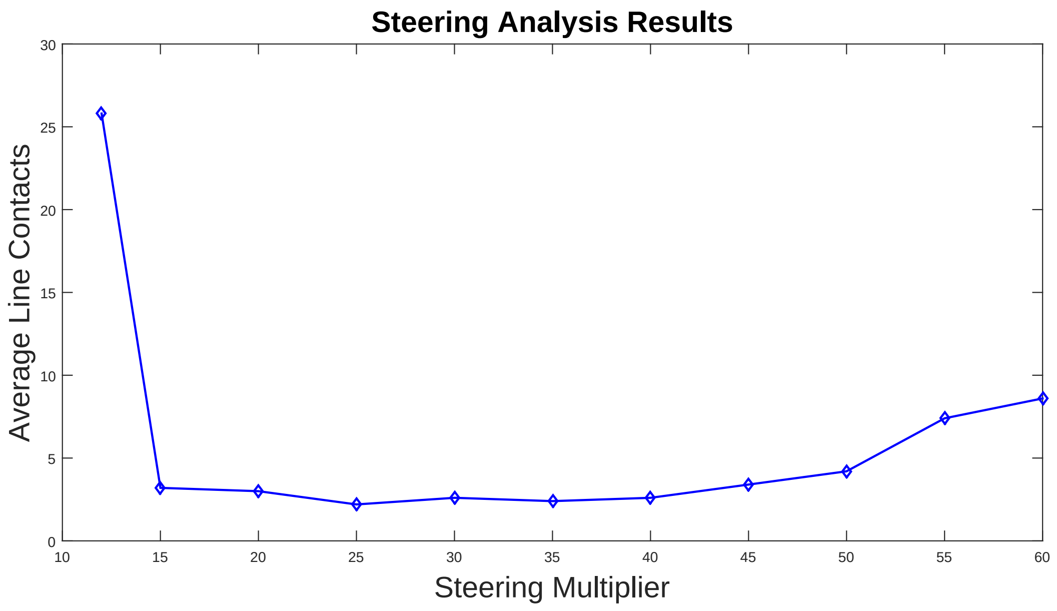

Figure 22: Sensitivity Results

4.3.2 Performance results. On the 100 lap track test, the car successfully completed 98 laps with only two failures due to navigation issues. The first occurred at Lap 77 and the second occurred at Lap 94. Both of these failures caused the car to drive off the track at which point we replaced it back on the track at the same location it exited and continued the test. This results in a mean distance to failure of 47 laps.

Additionally, we monitored how the car performed in each section shown in Figure 19. Figure 23 shows the results of these tests with the road sections along the x-axis and the average error rate along the y-axis. These rates do not include the two laps where the car experienced navigation failures.

As shown in the figure, Section A performed the worst. This section of the track is characterized by a single lane in each direction, a dashed center yellow line, and several wide turns. While driving this section, the car touched the outer lane marking an average of 1.28 times per lap and crossed the center line an average of 0.56 times per lap. However, the car never crossed the center lane. While not ideal, this is most likely due to the relatively narrow field of view and could be easily corrected by averaging more croppings of the original image take from different offsets.

Section B performed the best. It features two lanes in each direction with a double center line. On this section, the car crossed a lane marking only once and touched either lane marking only 0.28 times per lap. The car always merged directly into the left lane and stayed there until the two lanes merged back together.

Overall, on average per lap, the car experienced 1.52 center lane touches, 3.16 outer lane touches, 0.0 center lane crossings, and 0.64 outer lane crossings. From this information we can see that the system displays a strong bias against the center yellow line. As no other papers provide this level of detail on their results, we cannot compare to prior work. We would like to reduce crossings and touches even further by reducing video latency and augmenting our data more significantly by using viewpoint transformations.

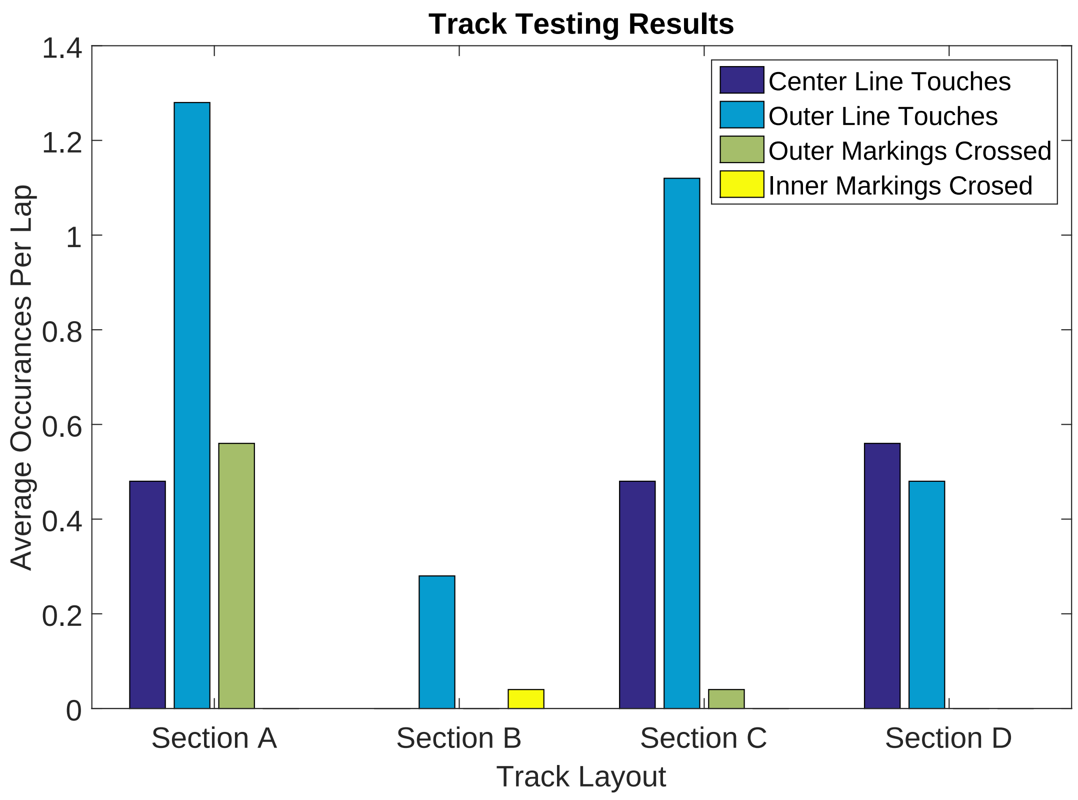

Figure 23: Performance Results

It is worth reiterating that we had the test car drive counterclockwise around the track. We did this because the test car was unable to consistently drive around the track clockwise. In Section 4.4 we discuss the cause of this performance deficiency.

4.3.3 Throttle characterization results. Figure 24 shows the results of the throttle characterization tests. The x-axis shows the distance between the two cars while the y-axis depicts the throttle value produced by the neural network.

These results depict two clear characteristics of the system: a minimum and a maximum effective distance. Below four inches, the lead car takes up the majority of the field of view of the camera leaving no view of the road to use as a reference. Therefore, the throttle control outputs large and meaningless values. Above nine inches the throttle output varies between 20 and 25, indicating that the car in front of the camera is either too small to see or too small to warrant braking action. However, between the range of four and nine inches, we can see the the car consistently applies additional brake pressure as the leading obstacle approaches.

4.3.4 Anti-collision performance results. We performed 20 trials of the anti-collision test. The car successfully avoided a collision in 16 of the trials with the anti-collision network applying the brakes and stopping the car. The average stopping distance of these successful trials was six inches.

However, in two of the 20 trials, neither the anti-collision network nor the steering control network were able to prevent a collision and the system hit the stationary car. The other two remaining tests were not collision failures, but stopping failures. In one of these other tests, the steering network drove off the road to avoid the car before the anti-collision network could apply the brakes. It appears that the steering control network may have also learned to avoid certain obstacles by driving around them. In the other test, the car drove off the road to avoid the car and then drove back onto the road after passing the stationary car.

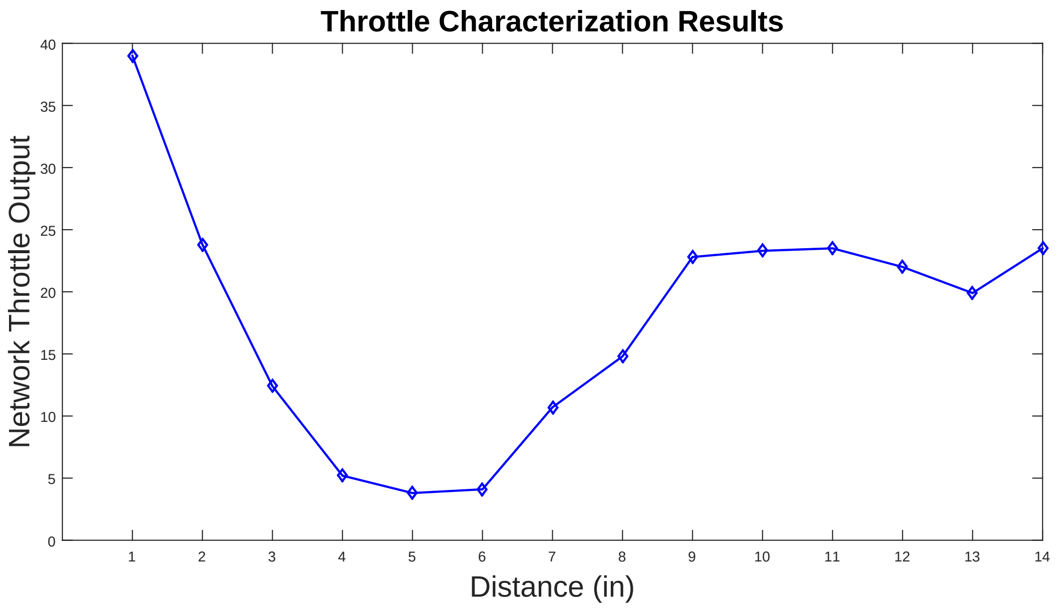

Figure 24: Throttle Characterization Results

4.4 Turn Bias Analysis

The strangest aspect of our track testing was the inability of the RC car to reliably complete counter-clockwise laps of the test track. Upon observation of this behavior we returned our focus to neural network validation testing.

4.4.1 Left vs. right analysis. We first hypothesized that the neural network may be less accurate on right turns than left turns, which would explain the behavior that we observed. To test this hypothesis, we designed a test to see if there was a difference in accuracy on left turns and right turns in our validation data. To do this we grouped together multiple frames that encompassed driving through a particular turn on the road. In the context of this analysis we will use the term ‘turn’ to reference a set of frames that represent a particular arc in the road. Our validation test set broke into a set of 79 left turns and a set 53 right turns. Figures 25 and 26 illustrate the mean error rates of the neural network on left and right turns. The data points are sorted in an attempt to determine whether or not the system displays a bias towards any given side. The error bars around each point represent the standard deviation of the error rates across all of the frames in a given turn. Because there are approximately equal numbers of turns with positive and negative errors in each turn direction we can conclude that there is no significant bias to the left or right while turning. Because the vast majority of the error rates on both turns are close to zero and the standard deviations are relatively small for both turn directions we were able to conclude that there was no significant difference between the neural networks performance on left or right turns.

4.4.2 Turn sharpness analysis. In pursuit of an explanation for the observed bias against right turns we returned to the test track. We noticed that since our track is a loop, the lane on the outside of the loop is a greater distance than the lane on the inside of the track. This mean that driving along the inside of the track resulted in sharper turns than the outside lane. With this information we then

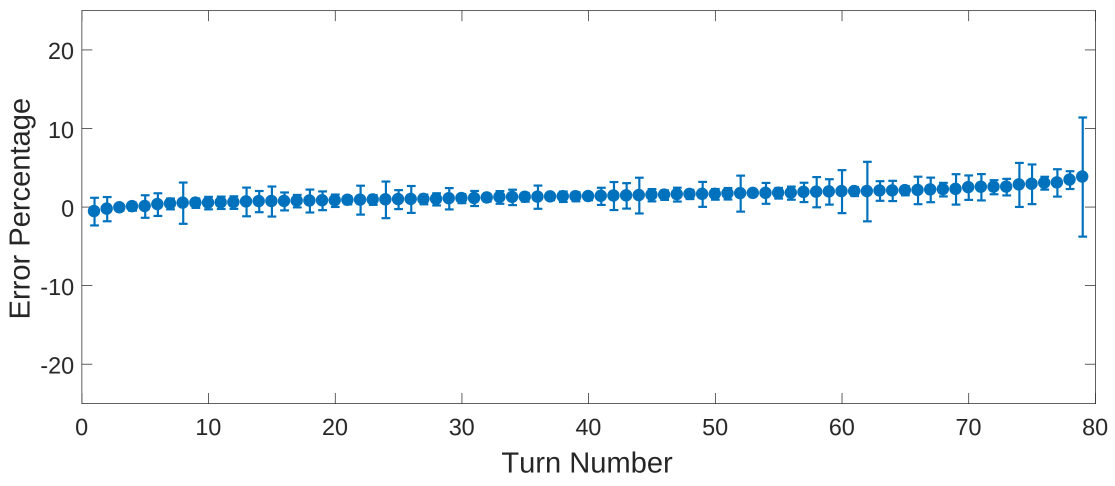

Figure 25: Mean error rate by turn: Left Turns

hypothesized that higher turn angles would result in higher error rates which could explain the anomalous behavior that we had observed on the test track. To test this theory we plotted each individual frame from our turn data on a scatter plot such that the x-axis represents the absolute value of the correct steering angle and the y-axis represents error after classification. If our hypothesis is correct we could expect to see error percentage increase as the absolute value of the steering angle increases. This plot is represented in Figure 27. In Figure 27 we can see that there is no connection between magnitude of steering angle and error, once again disproving our hypothesis.

4.4.3 Left vs. right with latency analysis. Having seen no difference in the neural network's ability to turn left or right in validation data, and seeing no difference in error rate based on turn sharpness, we considered the characteristics of our testing setup. One of the limitations that we had to deal with throughout our real-world testing was the video transmission latency. We hypothesized that our video transmission delay may be affecting left and right turns differently. To test this hypothesis we delayed our steering classifications by one frame. This way Frame n would be labeled with the steering angle from Frame $n + 1$. This creates the same effect as video delay.

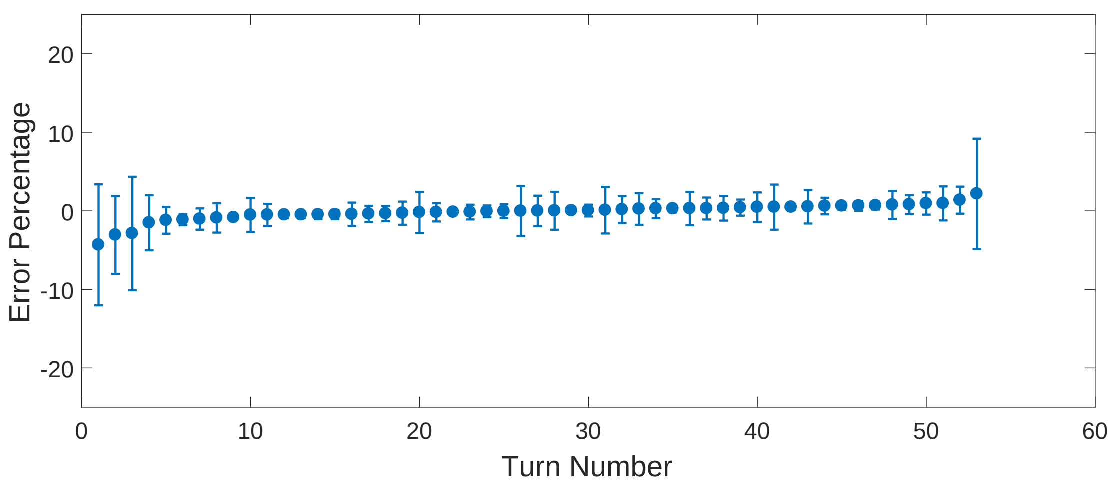

Figure 26: Mean error rate by turn: Right Turns

After shifting our labels we performed the same test as we did in 4.4.1 to produce Figures 28 and 29. These figures show significantly higher standard deviations than Figures 25 and 26. We can see that right turns are affected disproportionately by this delay penalty. Figure 29 shows a greater number of turns with high standard deviations. The average of the error for each of the left turn is 3.0 and the average of the error of all of the right turns is 3.7. It should be noted that these metrics are averages of averages and are therefore somewhat diluted. This phenomenon effectively explains the inability of the RC car to consistently complete clockwise laps of the test track. These results make sense given the difference in viewpoint for right and left turns created by driving on the right side of the road. From the right lane, a driver can see further down their path along a left turn than they can around a right turn.

It is also worth noting that left turn 57 and right turn 1 take place at night. As mentioned in Section 3.5.1, poor lighting has an adverse effect on vehicle performance. The combination of poor lighting and simulated latency is most likely the cause of the high standard deviation in these turns.

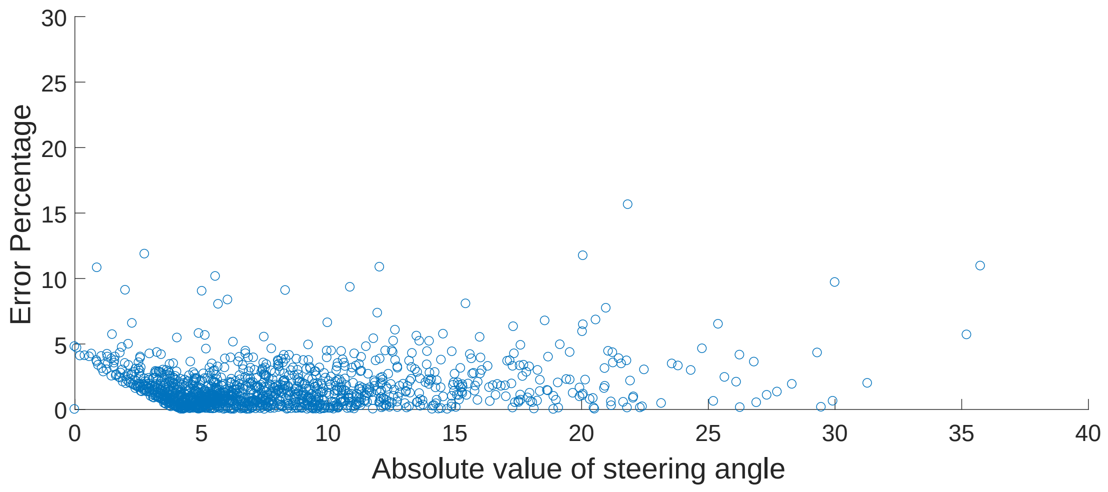

Figure 27: Steering angle ground truth with respect to classification error

4.5 Discussion

The results of our testing indicate that virtual training data is a viable option for neural network training. Our RC car is able to drive for long periods of time without making an unrecoverable error as well as to detect oncoming obstacles and stop to avoid a collision. Therefore, we can conclude that simulation training does translate to the real world if the simulated environment matches reality closely enough. In addition, the control system is able to drive effectively under a wide range of steering sensitivity settings. This indicates that the control system is flexible enough to be applied to a variety of different vehicles.

Our test track features three different road types, providing a good indicator of the flexibility of our system on different road types. In Section 3.5.1 we observed that the system performed well on roads with an outer lane marking. The results of our real-world experimentation provide additional support for that notion. Not only was the car able to complete 98 of 100 laps without an unrecoverable error but, on the multiple lane road section, the car always chooses to drive in the left lane closest to the white line.

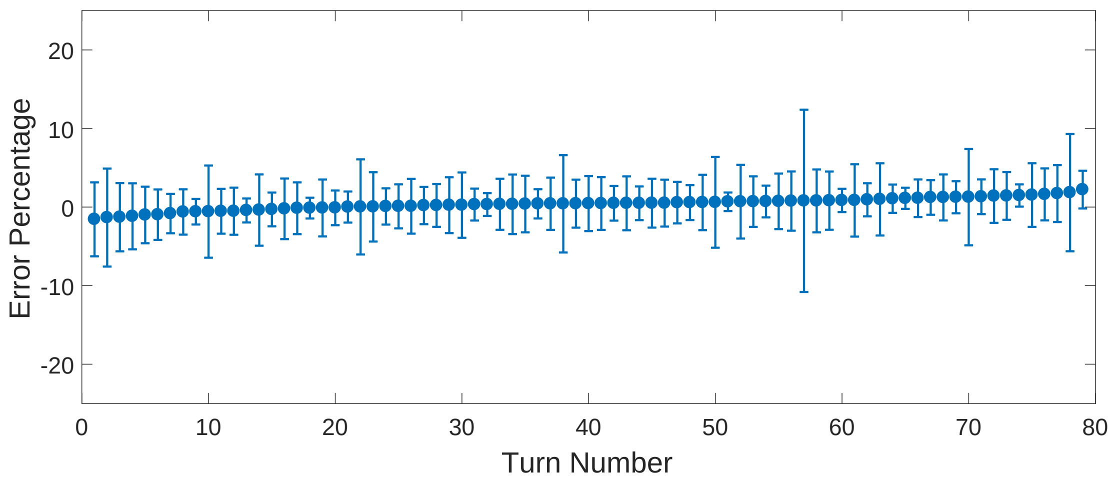

Figure 28: Mean error rate by turn: left turns with simulated delay

Our tests in Section 4.4.1 showed that the neural network performed significantly worse after we introduced artificial latency and that the latency would negatively affect right turns more than left turns. Our track tests allowed us to observe both of these phenomenon in the real-world. To compensate for the performance penalty of the video transmission delay we had to ensure that the RC car drove at a very low speed. On our track tests, the car drove relatively long distances without an error due to the control system despite the data transmission delay. In addition, the test car was able to consistently complete counterclockwise laps of our track, which contain mostly left turns. However, the test car struggled with clockwise laps because that are dominated by right turns.

Without the delay of transmitting video and instructions via WiFi and the RC remote control, we can safely assume that the RC car's performance would be smoother. We can also assume that the test car would be able to successfully navigate right turns a greater percentage of the time. On a full-sized vehicle, there would be plenty of room for computing equipment, which would eliminate the video transmission delay. Because of the delay, our track tests presented the system with a more challenging test than real roads.

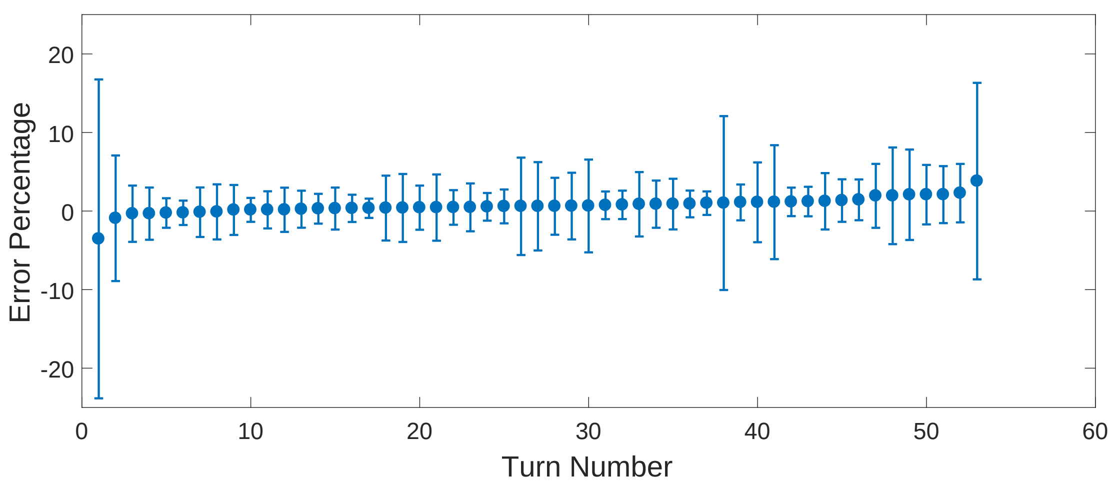

Figure 29: Mean error rate by turn: right turns with simulated delay

Our system is reliant on a single, fixed camera, which presents a few limitations. Depth perception and turn radius are both limited by the monocular view provided to the neural networks. The depth perception needed by the anti-collision network is achieved by comparing relative size. Furthermore, the turn radius is limited because the camera is unable to pivot, which would keep the most relevant information about the road inside of the field of view. These limitations could easily be solved by using more cameras. The performance of the system could be further enhanced by providing the system with a greater variety of sensors. This, however, would likely require a custom built simulation environment for data collection.

Chapter 5: Conclusions

5.1 Conclusion

Our work supports the notion that autonomous vehicles can operate using reflexive convolutional neural networks trained with augmented simulation data. We implemented a neural network and trained it using data taken from the popular video game Grand Theft Auto Five. We then tested the performance of the neural network using testing data also taken from the game. Then, we verified the capabilities of the network by having it operate a small scale car in the real world. Real world validation is critical to autonomous driving research. As we discovered in our own experiments, simulation performance and real world performance can differ significantly. This is why it was so important that we verify the functionality of our system in the real world. Some researchers opt to limit their explorations to simulation and never test their systems in the real-world. Our work illustrates the importance and effectiveness of transitioning from simulations to the real-world.

Through a number of real world experiments, our system successfully drove 98% of the laps on our track without navigation issues as well as achieving a 90% success rate in avoiding collisions. We were able to achieve these results using only a single camera as input. This demonstrates a useful and novel method of creating and training an autonomous vehicle system that allows for fast redeployment under new conditions and new systems.

In the future, we plan to expand our work to support traffic lights and roads missing outer painted lines. Additionally, we plan to confirm operation on an autonomous golf cart, increasing the scale of the real world experiments to further prove the validity of our approach.

5.2 Future Research

In this section we will propose a variety of hardware and software changes. The changes that we propose in the forthcoming sections would allow us to either test our system more extensively or improve the system's performance.

5.2.1 Neural network retraining. In alignment with the conclusions that we drew in Section 4.4.3, a natural extension of this research would be to retrain the existing neural network with the original data with offset labels. By offsetting labels to simulate video transmission latency we may be able to, not only improve performance on right turns, but improve performance in all scenarios. Even though there would be no video transmission delay if our control system were to be implemented on a full size vehicle, there would be some delay associated with the time needed for the neural network to process the frame. Retraining the neural network to account for processing and transmission delays would likely improve performance significantly.

5.2.2 Additional vehicles. Having verified the functionality of our autonomous car controller on a small scale RC car the next logical step is to implement then same controller on a full scale vehicle. One avenue that we are currently exploring is instrumenting a golf cart. A golf cart is large enough to drive on regular roads and is easier to work with than a regular car due to its relative simplicity. While a golf cart would allow us to perform a wide variety of tests it would not allow us to test our system on the freeway. To perform freeway tests with our system we would need to instrument a car capable of driving at freeway speeds safely. Instrumentation of a full scale vehicle would allow us to perform a comprehensive set of tests to formally define the capabilities of our control system.

5.2.3 Stereoscopic 3d. Due to the modular nature of the neural network that we designed it would be simple to add an additional input image from an additional camera. By observing two camera angles the neural network would be able to learn to determine the distance to objects in the scene. This could greatly improve the obstacle avoidance capabilities of the system. The only challenge would be collecting data from the game. The current data collection tools would need to be redesigned. Fortunately, GTA V supports 3D gameplay it may be possible to leverage existing graphics drivers to capture the additional camera angle and image from it.

5.2.4 Neural network architecture. There is more exploration that could be done in the realm of neural network architecture. One of the limiting factors that we had to deal with throughout our research process is the size of GPU memory. Given a GPU with a greater amount of on-board memory we could implement more complex networks and experiment with different structures. Given the additional system resources it may be possible to combine our throttle control network and steering control network into a single, branching, neural network. This could potentially save us the overhead of needing to run two networks simultaneously.

5.2.5 Increase autonomy. In its current state our system is only able to achieve autonomy by following the road that it is on. In the future, we plan to expand our work to support traffic lights, roads missing outer painted lines and turn signal input. By recognizing traffic lights and stop signs our system will be able to maintain autonomy in a greater number of driving scenarios. This addition would significantly improve usefulness in urban environments. By supporting roads without outer lane markings we would further increase the number of roads that our system would be able to function on. Finally, by accepting the state of the turn signal our system could learn to cope with intersections. Furthermore, by accepting turn signal input and learning to correctly handle intersections, our controller could then be connected to a GPS navigation system to achieve nearly full autonomy.

REFERENCES

[1] R. Games, "Grand Theft Auto V," Accessed 2017-1-10. [Online]. Available: http://www.rockstargames.com/V/

[2] W. P. Warren S. McCulloch, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathmatical Biophysics v5*, 1943.

[3] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification," *Neural Networks*, vol. 32, no. Supplement C, pp. 333–338, Aug. 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608012000524

[4] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column Deep Neural Networks for Image Classification," *arXiv:1202.2745 [cs]*, Feb. 2012, arXiv: 1202.2745. [Online]. Available: http://arxiv.org/abs/1202.2745

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[6] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[7] DARPA, "Darpa Urban Challenge 2007," Accessed 2017-1-15. [Online]. Available: http://archive.darpa.mil/grandchallenge/

[8] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Stanley: The robot that won the DARPA Grand Challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, Sep. 2006.

[9] A. Broggi, C. Caraffi, P. P. Porta, and P. Zani, "The Single Frame Stereo Vision System for Reliable Obstacle Detection Used during the 2005 DARPA Grand Challenge on TerraMax," in *2006 IEEE Intelligent Transportation Systems Conference*, Sep. 2006, pp. 745–752.

[10] D. U. C. Team, C. Urmson, J. A. Bagnell, C. Baker, M. Hebert, A. Kelly, R. Rajkumar, P. Rybski, S. Scherer, R. Simmons, S. Singh, A. Stentz, W. Whittaker, and J. Ziglar, "Tartan Racing: A Multi-Modal Approach to the DARPA Urban Challenge," *Robotics Institute*, Apr. 2007. [Online]. Available: http://repository.cmu.edu/robotics/967

[11] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991.

[12] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars," Apr. 2016.

[13] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. L. Cun, "Off-road obstacle avoidance through end-to-end learning," in *Advances in neural information processing systems*, 2006, pp. 739–746.

[14] J. Mannes, "Training self-driving cars on the streets of

Los Santos with GTA V just got easier." [Online]. Available: http://social.techcrunch.com/2017/01/11/training-self-driving-cars-on-the-streets-of-los-santos-with-gta-v-just-got-easier/

[15] C. Vallon, Z. Ercan, A. Carvalho, and F. Borrelli, "A machine learning approach for personalized autonomous lane change initiation and control." [Online]. Available: http://www.me.berkeley.edu/ frborrel/pdfpub/machine-learning-approach.pdf

[16] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates, and A. Y. Ng, "An empirical evaluation of deep learning on highway driving," *CoRR*, vol. abs/1504.01716, 2015.

[17] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan, "Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?" *arXiv:1610.01983 [cs]*, Oct. 2016. [Online]. Available: http://arxiv.org/abs/1610.01983

[18] A. Filipowicz, J. Liu, and A. Kornhauser, "Learning to Recognize Distance to Stop Signs Using the Virtual World of Grand Theft Auto 5," in *TRB 96th Annual Meeting Compendium of Papers*, 2017. [Online]. Available: https://trid.trb.org/view.aspx?id=1439152

[19] Johnny Manson, "GTA V Scripting Tutorial | Installation & First Script | #1." [Online]. Available: https://www.youtube.com/watch?v=NmuoN6z7C8Y

[20] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[21] "Keras Documentation," Accessed 2017-09-05. [Online]. Available: https://keras.io/

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[23] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 315–323.

[24] TensorFlow, "Tensorflow," Accessed 2017-1-17. [Online]. Available: https://www.tensorflow.org/

[25] Microsoft, "Microsoft cognitive toolkit (cntk)," Accessed 2017-1-17. [Online]. Available: https://github.com/Microsoft/CNTK/

[26] Theano, "Theano," Accessed 2017-1-17. [Online]. Available: https://github.com/Theano/Theano