2019

End-to-End Learning: Using Neural Networks for Vehicle Control and Obstacle Avoidance

Keith Russell
Georgia Southern University

End-to-End Learning: Using Neural Networks for Vehicle Control and Obstacle Avoidance

An Honors Thesis submitted in partial fulfillment of the requirements for Honors in Mechanical Engineering.

By
Keith Russell

Under the mentorship of Dr. Biswanath Samanta

ABSTRACT

End to End learning is a method of deep learning which has been used to great effect to solve complex problems which would normally be performed by humans. Within this thesis, a neural network was created to mimic the steering patterns of humans in highway driving situations. A Turtlebot was used in place of a car and was tested within a laboratory on a closed loop track to drive within the lanes created for it. The network architecture was based on that of Nvidia's model which was used for predicting steering angles of a vehicle. The network was successfully trained and implemented, however showed poor performance, under fitting the predictions to a single value for all tests performed on it. This error is most likely the result of inconsistent and unclear data, causing the network to fail to recognize any pattern between steering commands and image features.

Thesis Mentor:_____

Dr. Biswanath Samanta

Honors Director:_____

Dr. Steven Engel

April 2019
Department of Mechanical Engineering
University Honors Program
Georgia Southern University

ACKNOWLEDGEMENTS

I would like to acknowledge my parents and my honors advisor Erin Martin for the enormous moral support provided during the writing of my thesis. Without such support, it may not have come to fruition.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

1.1 Why Autonomous Vehicles?

Autonomous vehicles have become a very important topic in engineering and computer science research in the past decade. The majority of major US auto manufacturing companies have released vehicles with semi-autonomous features with some, such as Tesla Motors, planning on full automation by 2020. With the prospect of not being able to drive the car you own within the next decade, many may wonder why this technology should be adapted onto the roads.

The most obvious reason why autonomous technology is advantageous to society is the safety that will come with it. In 2017, there were over 37,000 automotive deaths, or 11.4 per 100,000 people, in the US alone (Federal Highway Administration, 2018). Automotive accidents account for approximately 1.32% of all deaths in the United States (NCHS, 2017). These number of automobile accidents increase in poorer countries with less developed infrastructure and booming populations (Federal Highway Administration, 2018). While only 21% of the population of the United States lives in rural areas, 57% of fatal vehicle deaths occur in rural settings (NHTSA, 2005). While it may seem counterintuitive that the majority of deaths would occur in rural setting, one must consider highways as the main cause of this. The monotonous and straight driving of highways takes a fatigue on the drivers which dulls the senses of possible danger, and thus a dangerous situation is created. Simple errors such as not checking blind spots and broken

brake lights may lead to lethal consequences. Autonomous technology helps to mitigate these errors and therefore the dangers of highway driving.

Besides the dangers posed by highway driving, there are other negative aspects of people assuming responsibility over their transportation. According to Hannson, there is an inverse correlation between commute time and personal health (Hannson, 2011). This added responsibility of driving to work daily is not just stressful, it is physically bad for you. The quantitative and qualitative benefits that commuters and travelers would experience as a direct result of improved autonomous technology is the incentive for development of this emerging field.

1.2 Current Autonomous Technologies

There currently exists a plethora of vehicle technologies designed to increase safety and satisfaction of drivers. The Society of Automotive Engineers (SAE) defines these features on a scale of 0-5 (NHTSA, 2019). At 0, there are very basic capabilities like blind spot warning systems. Vehicles being produced now come standard with features like adaptive cruise control, which would be in around 3 on the autonomy level. A level 5 vehicle would be able to operate independently of the driver entirely during operation, from parking, to sign and lane recognition, obstacle avoidance and path planning. 75.4% of automobile accidents are caused by driving task errors, as opposed to technological errors or road surfaces (Najm, 1995). As the adoption of higher level SAE automation increases, there should be a proportional drop in the number of automobile accidents for this reason.

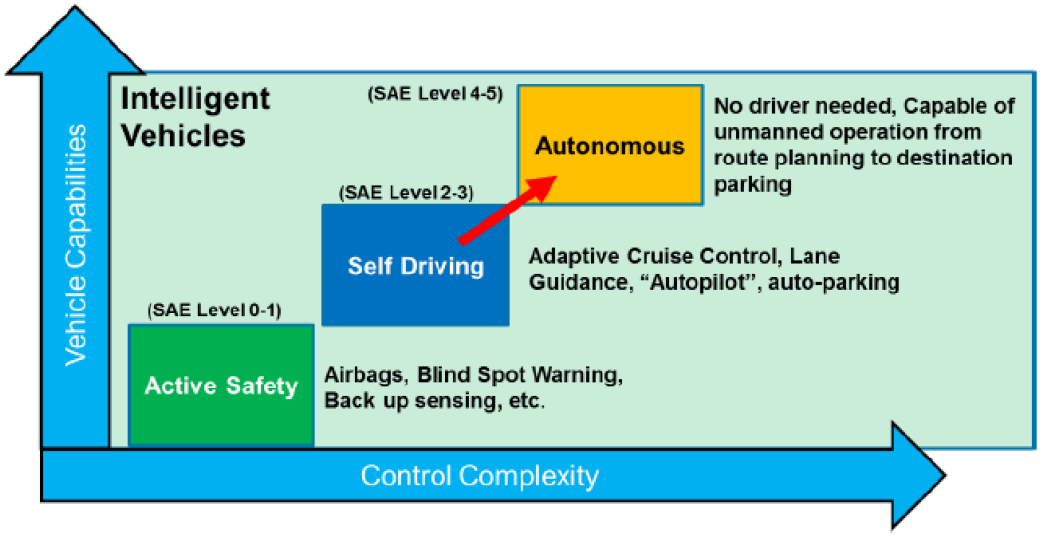

Figure 1. SAE Levels of Automation. (Ibru, 2017)

While there will always be variables that cannot be controlled for on the roadways, the number of accidents can be greatly reduced.

One of the technologies already mentioned, adaptive cruise control (ACC) has already been implemented for a number of years now. This technology utilizes radar to accurately estimate the position and speed of the vehicle in front of it during cruise control so it can adjust speed and avoid rear end collisions due to operator negligence (Bahtia, 2003). This is good for improving existing technologies but still requires much attention from the driver.

One of the more advanced technologies being used in autonomous vehicles in LIDAR, which stands for light detection and ranging. It uses pulses of infrared light to localize itself within an environment, and can detect other vehicles as well as static

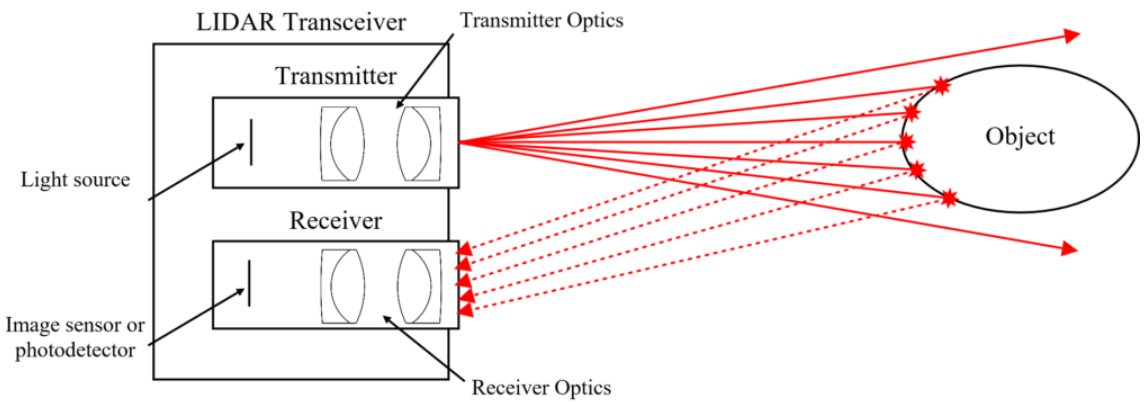

Figure 2. LIDAR detection system. (Eckhardt, No Date)

objects like building and signs (Schwarz, 2010). While LIDAR is a very useful method of creating depth fields surrounding a vehicle, it has two major downfalls. It cannot extract detail from an environment, such as differentiation between a person and a sign. It also is at a disadvantage in adverse weather conditions and in direct sunlight.

This leads to the kind of autonomous technology that will be the subject of this thesis: computer vision. Computer vision is useful for its ability to extract features from an environment. Computer vision can be used in conjunction with other technologies as well, such as ACC, where the speed limit would be read and set as the new cruise speed (Ljubo, 2001 a). Computer vision can perform a multitude of other useful tasks such as stereo-based object detection and tracking, pedestrian recognition, Lane detection and tracking, traffic sign detection, traffic light recognition, road marking and crosswalk recognition (Ljubo, 2001 b).

1.3 Deep Learning

In recent years, deep learning has become a popular tool for computer vision to efficiently extract features from an environment, especially when trained on large and diverse enough data sets.

1.3.1 A Brief History

The Term “deep learning” was coined by Rina Dechter in 1986, although the mathematical concepts and even applications of it date back much further (Juergen, 2015). Deep learning was first successfully used in 1965, though it would be another couple decades before the technology was used for any computer vision purpose (Ivakhnenko, 1973). In 1980, the Neocognitron was used to read handwritten characters (Fukushima, 1980). Similarly, in 1989, an algorithm was developed which successfully deciphered handwritten zip codes on mail (LeCunn, 1989). Until very recently, smaller researches have not had the computing power necessary to effectively use deep learning for research, however with the improvements in GPU performance in the past decade, many smaller institutions have been able to make contributions to this increasingly saturated field.

1.3.2 Deep Neural Networks

What are neural networks? And what makes them “deep?” Neural networks themselves actually aren’t algorithms, rather a framework for connecting individual algorithms together in order to process multiple inputs in a way that computers will understand (DeepAI, No Date a). Deep neural networks are just neural networks which contain many layers of computation (DeepAI, No Date b).

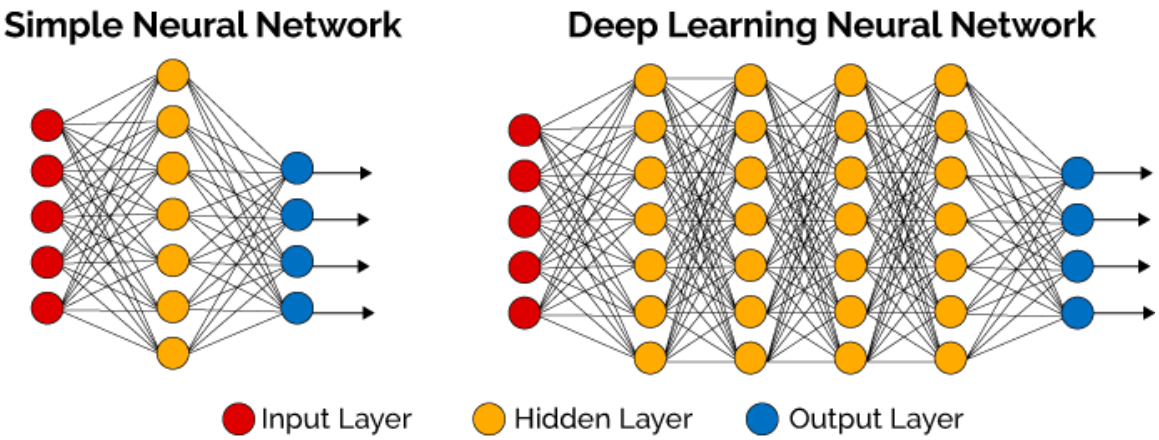

Figure 3. Simple vs Deep Neural Network. (Vázquez, 2017)

The method of deep learning I use in this paper is unsupervised learning, in which no data is labeled and there are no defined parameters which ought to be optimized, so it becomes impossible for the person training the network to know exactly how it works (DeepAI, No Date b).

1.3.3 Convolutional Neural Networks

To further specify, the network I will use will be an unsupervised convolutional neural network, which are neural networks which replace at least one of their matrix multiplication layers with a convolutional one (Admin, 2017). Convolving in deep learning is a process of filtering data to extract and concentrate features, and is generally used to process image data (Stanford, No Date). Within an image, a convolutional layer may reduce its size by only representing the most important parts of the images, such as edges. This helps to reduce the number of inputs, since unfiltered input of a colored 66 X 200 image would be 39,600 inputs.

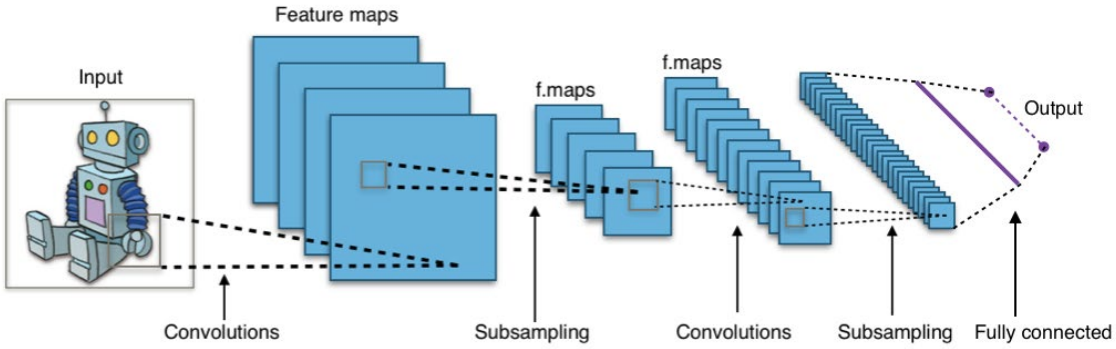

Figure 4. Convolutional Processing. (Stanford, No Date)

1.3.4 End to End Learning

End to end Learning is a method of training a complex system using a single architecture (Glasmachers, 2017). This will require the network to be very adaptable and robust, able to predict the outcome of a system given any potential unforeseen occurrences within the system. This method of training thrives on very large data sets because of this, and can also be behave very differently depending on the quality of the data used to train it. This method of training has proven very successful, such as Google's Deepmind AI, which beat the world champion "Go" player to become the best in the world (Silver, 2016).

1.4 Hypothesis

Assuming that a network is designed and trained properly, I should be able to use this trained network to steer a robot without intervention down a constructed pathway. This would substitute for a real vehicle driving on the road and demonstrate the ability of a neural network to act as an effective autonomous technology.

1.5 Criteria for Success

In order to determine that this hypothesis is successful, there are a number of objectives which must be met:

- There must be a network designed for and trained by image data of the created pathway
- The network must be able to accurately predict the steering patterns when compared to a human driver
- The robot must be able to make it through the path without infractions or straying away from the path

This thesis can be considered partially or fully successful depending on the number of the above objectives it has met.

CHAPTER II

LITERATURE REVIEW

Within the past few years, there has been a sudden growth in interest of all deep learning topics, including end to end learning for vehicle control. Many advancements have been made in this field recently, with researchers successfully creating working systems which can be taught to drive on and off roads, avoid obstacles, and be integrated with other path planning technologies to more usefully implement the technology.

2.1 End to End Learning for Highway Driving

Nvidia was one of the first entities to create a successful model for self-driving using end to end learning. Their paper, "End to End Learning for Self-Driving Cars" used large datasets of steering angles associated with images to train the network to drive based on an input of images, achieving fully autonomous driving 98% of the time (Bojarski, 2016). This proved that computer vision for self-driving vehicles could be used for more than simple classification purposes and was very accurate at driving predictably in a highway setting. At most, this technology has the ability to replace conventional, expensive technologies like RADAR and LIDAR, and at least has the ability to aid in lower level autonomous functions like adaptive cruise control and lane keeping during highway driving. This thesis is based mostly upon this Nvidia's paper, and uses a similar network structure and testing method, only with a Turtlebot instead of a car.

2.2 End to End Learning for Off-Road Obstacle Avoidance

One of the downfalls of end to end learning is the requirement of predictable environments. While the majority of driving situations do not require human level

intuition, there are many examples of unpredictable events occurring on the road. No matter the size of the training set, there will never be enough data to cover all possible situations. For example, if a semi-truck wrecks and flips onto its side, the network may have trouble predicting an optimal steering angle due to its lack of experience with traversing wrecks. Another situation is off road driving, which is filled presents a challenge of frequent obstacle avoidance. Off road capabilities have been achieved using end to end learning though, with great success in replication of steering commands given by humans (LeCunn, 2005).

2.3 End to End Learning with High Level Commands

Another problem with end to end learning for vehicle control is that when trained only with images, the capabilities of driving are reduced to highway driving, essentially. Much research has been done to bring end to end learning into relevance by implementing a path planning algorithm that uses LIDAR. Wei Gao et al. managed to create an indoor driving system using both an A* path planning algorithm and a simple discretized path planner which gave simple commands such as turn left or right, etc (Gao, 2017). The network was trained on images of the surroundings of the robot and of the intention of the robot using shared weights, and was implemented with great success. Within pre-mapped environments it seems, it is very possible to implement end to end learning with path planning. This has also been successfully implemented on full scale vehicles using four surround view cameras, GPS coordinates and a GPS mapping system (Hecker, 2018).

CHAPTER III

METHODS

The methods for accomplishing this thesis can be split into three major parts: data collection, training the network, and testing the network. Data collection was performed by steering the turtlebot down a pathway with an XBox controller, while a data collection program collected images from the camera and saved each with the given velocity command at that time. The images were saved in a folder and were related to the steering angle and speed commands via a csv file. The dataset totaled over 16,000 images and was 1.6 GB in size. This was used to train the network, which was inspired by the architecture put forward by Nvidia in their paper "End to End Learning for Self Driving Cars." The network saved architecture in the form of json and weights as h5. The model architecture and weights were then used to steer the turtlebot based on images it was subscribed to through ROS.

3.1 Turtlebot Setup

The turtlebot was controlled using velocity commands remotely. The host computer was connected to the turtlebot to provide velocity inputs, while the main computer was connected through a ROS network to the host, and was directly connected to the camera for data collection and use in implementing the trained network. The turtlebot had a Kinect camera mounted in the middle of it which was used to provide the images to train and test the network.

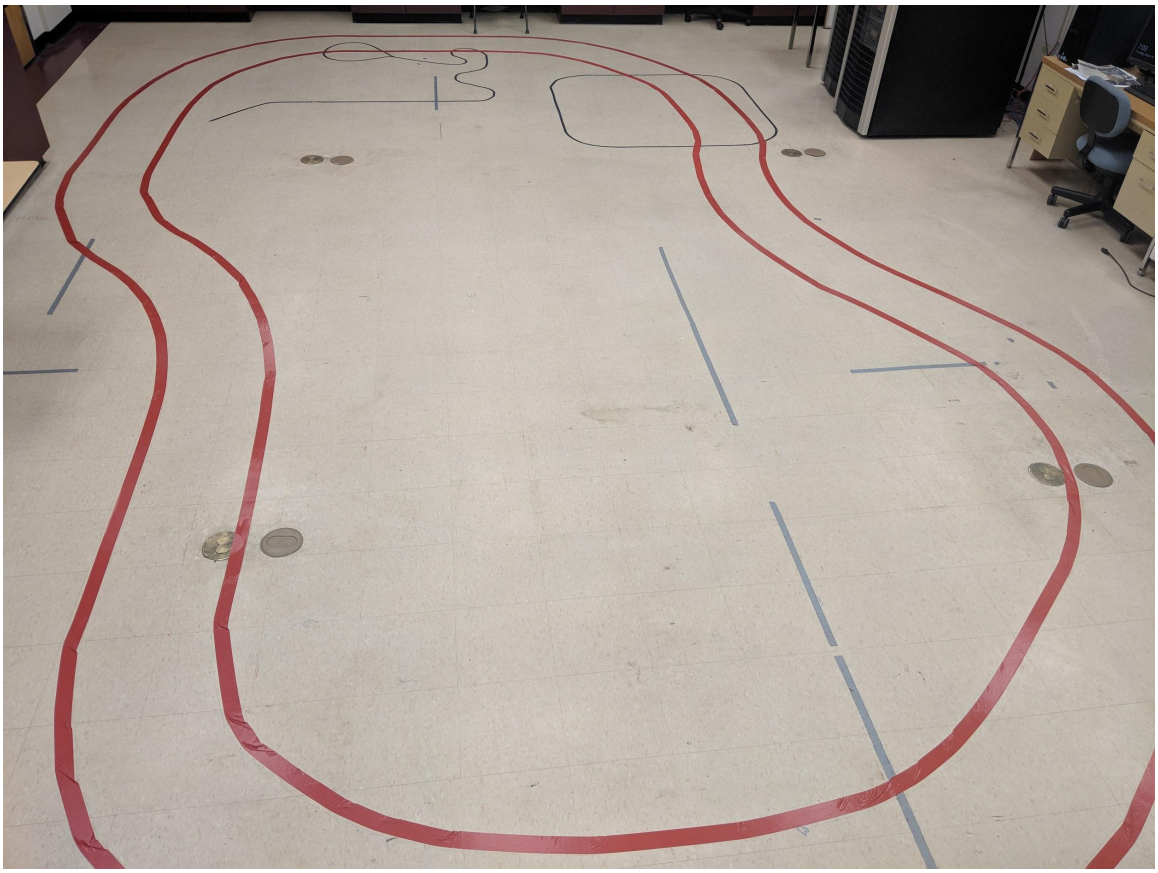

Figure 5. The track used to test the network.

3.2 Track Design

The track was designed within a mechatronics classroom and so was fairly compact. It consisted of a continuous loop with some minor twists and turns. The track was just wide enough for the turtlebot to fit entirely inside and was within the view of the turtlebot at all points while on the track. The lines were made of bright red tape, making it a very distinguishing feature against the dull floors. Yellow was considered, in order to maintain realism with actual roads, but it was decided that red would be a more easily trainable feature. The track used to train the model and the one used to test it were different, however most of the distinguishing features were kept intact. The only difference in the track used

for testing was where within the classroom that the twists and turns were. This was to keep it familiar enough to be still recognizable by the network but different enough to test its effectiveness.

3.3 Design of Neural Network

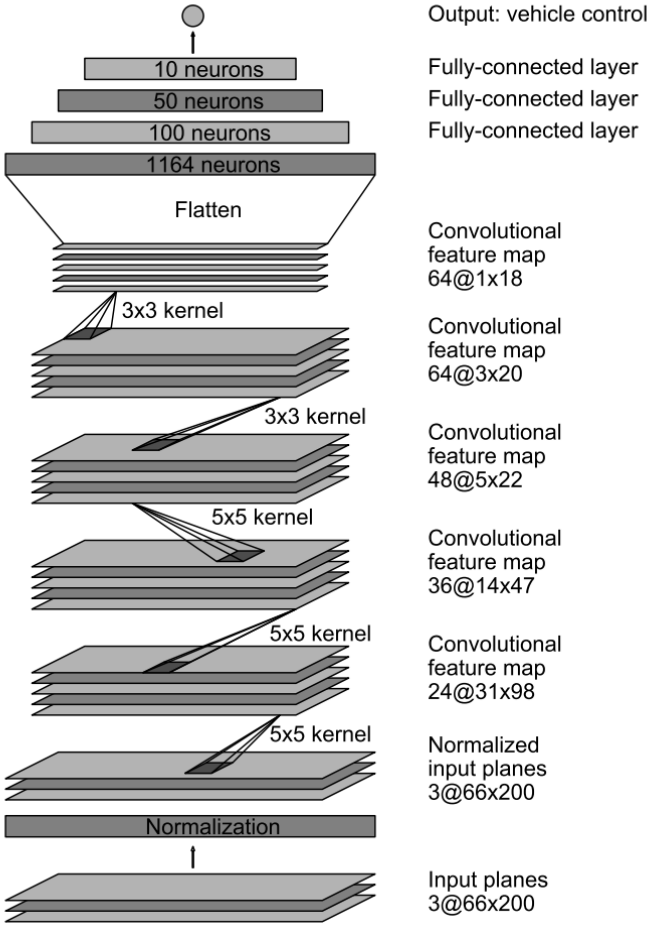

Figure 6. Nvidia Model Architecture. (Bojarski, 2016)

The network created follows the design of Nvidia's network very closely, constructed using Tensorflow with a Keras front-end. Nvidia's model featured a normalization layer, five convolutional layers and 3 fully connected layers. The images were shaped into 66 x 200 arrays and split into RGB planes. There were several

augmentation techniques were used including brightness variation, angle offsetting, and mirroring. These techniques were used to increase the robustness of the data so that the network could adapt to a greater number of scenarios. Several variants of training variables such as batch size, samples per epoch, and number of epochs were tested, with few differences found between each. The final settings used for testing were a batch size of 32, 300 samples per epoch, and 250 epochs.

3.4 Collection of Data

Data was collected using a ROS network connected within the IRIS Lab, via a data collection program from the Github repository "End-to-end-self-driving-robot" by Fdevmsy (Fdevmsy, 2017). The host was set as the Turtlebot netbook. The purpose of this computer was to receive the velocity commands and translate them to movement through the Turtlebot. Another laptop was connected through the network which was connected to an Xbox controller and the Kinect camera used to gather the image data. The data collection algorithm sampled 20 time frames per second, recording a steering angle and speed for each cataloged image. Figure 7 shows the csv and image folder structure the data was stored in. The csv file stored the image file name, speed and steering angle, respectively.

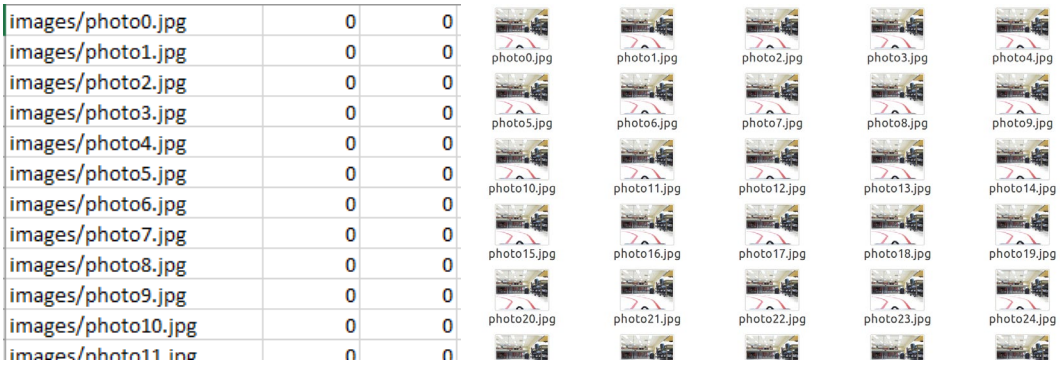

Figure 7. Structure of the csv file and image folder, respectively, that the data was stored in.

3.5 Testing of Network

The network was tested using a setup similar but not identical to the track design shown in figure 5. The Turtlebot was connected to the host netbook which was receiving steering commands through the ROS network, via a program also from Fdevmsy (Fdevmsy, 2017). Another laptop was connected to the Kinect camera, and was giving predictions for the input image frames and publishing them to the velocity command topic. This would allow the Turtlebot to steer itself based on the learned patterns from the trained network.

CHAPTER IV

RESULTS

The network was tested multiple times on the same track and displayed similar behavior despite multiple changes in the preprocessing and training of the data. The program used to predict and publish steering angles gave a very similar output to different situations presented to it as input. Figure 8 shows the predicted steering angles (on the left side of the figure) when given an input of the image on the right side of the figure. This appears to have an error based solely on the magnitude of angle output. The angle (shown in radians) implies that the Turtlebot should be turning at this location, however the correct action to take would be to continue straight (with a turning angle of 0 radians) for at least several more seconds.

Figure 8. Turtlebot Output during Straight Pathway

Figure 9. Turtlebot Output During Curved Pathway

While the incorrect angle prediction is certainly a problem, the main issue is that the network only predicts a single output angle (give or take a few ten thousandths of a radian). This can be shown by comparing the output feed of the program in figures 8 and 9. In this example and any other input provided, the output is effectively the exact same amount. This renders the network unusable, and implies that an error was made in either the data collection, training, or implementation of the network. The network was retrained several times with different parameters and augmentation techniques. The data was changed from being processed as YUV to RGB, cropping was reduced, and other minor augmentations were removed. The number of epochs was increased and samples per epoch were decreased, however no changes in performance of the network took place. The dataset was then analyzed as a possible source of the failure. To analyze the dataset, two time frames were taken which correlate to the beginning and end of a curve. The images

taken during this time frame are shown in figure 10. The curve is the first which takes place in the training dataset and is fairly sharp but still visible throughout the entirety of it.

Figure 10. The Beginning and End of Curve 1

The steering angle for such a curve would be sinusoidal in shape, with the maximum steering angle occurring in the middle of the turn and resolving to zero at both ends where the turn subsides. This example of an ideal steering profile can be seen in figure 11. The actual steering profile for this curve can be shown by analyzing the steering angles between the two time frames of the images in figure 10. This result is displayed in figure 12, and is far from the ideal sinusoidal shape for this type of curve. This difference between ideal and actual shape of the data is what is most likely causing the prediction error within the network. This data is very unclear as to what is the optimal steering angle for this curve, showing in the middle a dip to zero where the steering angle should be highest and many uneven spikes within the entire curve. This lack of a definite trend is what causes confusion while training. In order to assign value to the image of a curve, the data must be

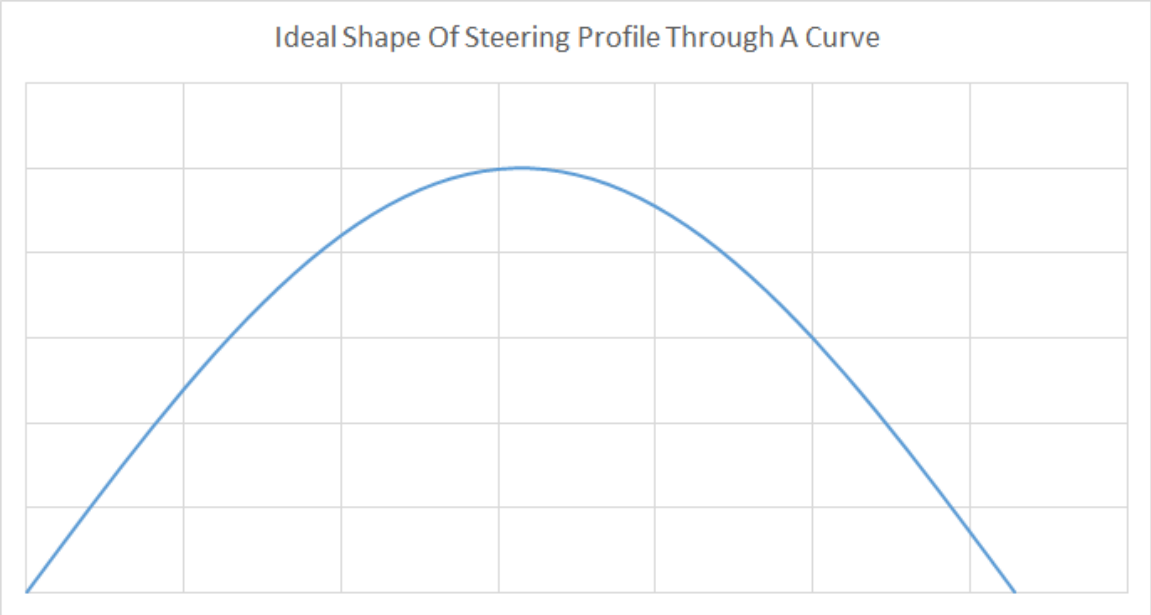

Figure 11. The Ideal Steering Profile for a Right Angle Curve

consistently suggesting some value that is optimal at that curve. If the data is too noisy and does not show any pattern between similar input data values, a single value which eliminates the error may be chosen by the network as optimal.

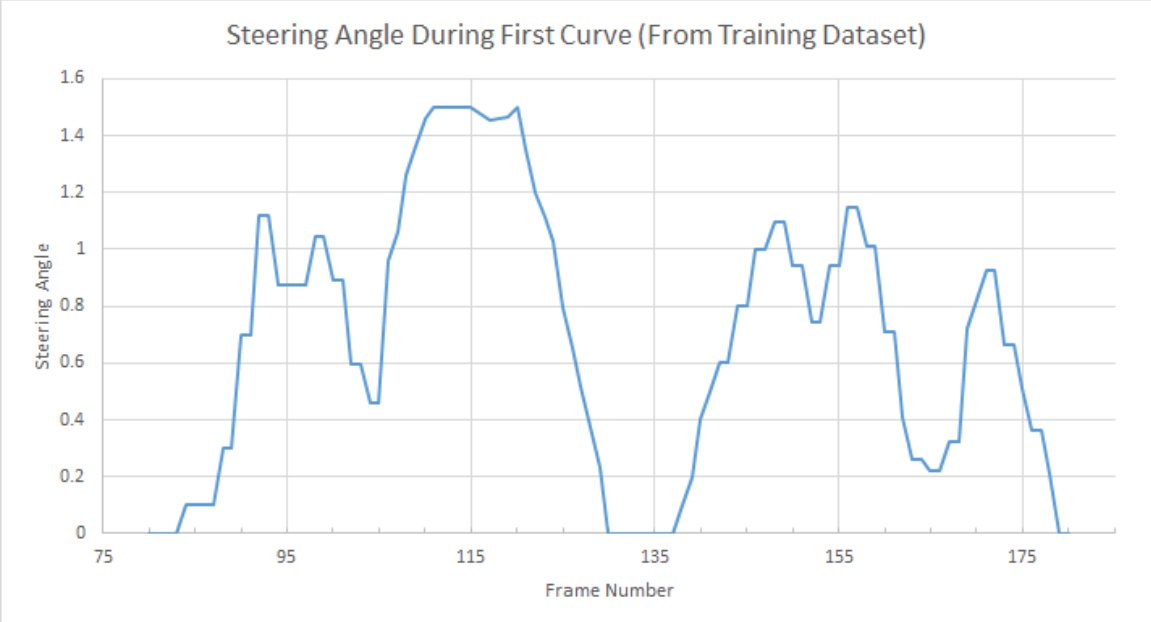

Figure 12. Actual Steering Values Between Beginning and End of First Curve

CHAPTER V

CONCLUSION

5.1 Validation of Hypothesis

The hypothesis of this thesis was that it was possible to use end to end learning to "steer a robot without intervention down a constructed pathway." While this experiment was unsuccessful in its attempt to achieve this task, the hypothesis is not entirely invalidated. The possibility of this deep learning method to steer a robot has already been demonstrated on full scale vehicles and remains a possibility given a better was to collect data.

5.2 Criteria for Success Evaluation

The criteria will be examined individually to determine if each was successful, then as a whole to determine the success of the project.

1. *There must be a network designed for and trained by image data of the created pathway.*

This criterion has been met. The network was successful in training on image data and predicting output steering angles for images as input.

2. *The network must be able to accurately predict the steering patterns when compared to a human driver.*

This was not achieved. The accuracy to human driving pattern were not achieved.

3. *The robot must be able to make it through the path without infractions or straying away from the path.*

This was also not achieved. The robot was not able to traverse any part of the constructed path successfully due to the issue discussed in the results section.

5.3 Future Recommendations

There are many improvements which could be made to this project to create a functionally steering robot.

First, the data should be collected in a smoother fashion. The Turtlebot velocity command input is very crude and does not replicate the smoothness of a steering wheel in a car, but improvements could be made to the programs which are used to translate Xbox controller commands to the steering commands received by the Turtlebot.

Second, the track could be altered to accommodate the lack of sensitivity of the steering. To achieve this, the track would need to be much larger and the turns much wider. It would still require a very steady hand to operate and the data would need to be checked before being assumed to be smooth enough.

Third, a smoothing algorithm could be applied to the data which averages the values of a given number of time frames out to create a more accurate representation of the necessary steering commands.

Fourth, full scale tests could be performed on cars. If one can gain access to the steering angle of the vehicle, it would be significantly easier and more substantial to use this data instead of that of a Turtlebot, which bears little resemblance to real life scenarios.

REFERENCES

Admin. 2017. Convolutional Neural Networks Tutorial in TensorFlow. *Adventures in Machine Learning.* https://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-tensorflow/

Bhatia, Pratyush. 2003. Vehicle Technologies to Improve Performance and Safety. *University of California Transportation Center*, Berkeley, CA. https://escholarship.org/uc/item/4zw4m05k

Bojarski, Mariusz, et al. 2016. End to End Learning for Self-Driving Cars. *ArXiv.* Cornell University, Ithaca, NY. https://arxiv.org/pdf/1604.07316v1.pdf

DeepAI. No Date, a. Neural Network. *DeepAI*. https://deepai.org/machine-learning-glossary-and-terms/neural-network

DeepAI. No Date, b. Deep Learning. *DeepAI*. https://deepai.org/machine-learning-glossary-and-terms/deep-learning

Eckhardt Optics. No Date. LIDAR – Light Detection and Ranging. *Eckhardt Optics*, White Bear Lake, MN. https://eckop.com/lidar-light-detection-and-ranging/

Fdevmsy. 2017. End-to-end-self-driving-robot. *Github Repository.* https://github.com/Fdevmsy/End-to-end-self-driving-robot

Federal Highway Administration. 2018. Traffic Volume Trends, January-December 2017. *U.S. Department of Transportation*, Washington, DC. https://www.fhwa.dot.gov/policyinformation/travel_monitoring/tvt.cfm

Fukushima, K. 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*. https://link.springer.com/article/10.1007%2FBF00344251

Gao, Wei, et al. 2017. Intention-Net: Integrating Planning and Deep Learning for Goal-Directed Autonomous Navigation. *ArXiv.* Cornell University, Ithaca, NY. https://arxiv.org/pdf/1710.05627.pdf

Glasmachers, Tobias. 2017. Limits of End to End Learning. *ArXiv,* Cornell University, Ithaca, NY. https://arxiv.org/abs/1704.08305

Goodfellow, Ian, et al. 2016. Deep Learning. *MIT Press,* Cambridge, MA. http://www.deeplearningbook.org/

Hannson, Erik, et al. 2011. Relationship between commuting and health outcomes in a cross-sectional population survey in southern Sweden. *BMC Public Health*, London, UK. https://bmcpublichealth.biomedcentral.com/articles/10.1186/1471-2458-11-834.

Hecker, Simon. 2018. End-to-End Learning of Driving Models with Surround-View Cameras and Route Planners. *European Conference on Computer Vision.* https://arxiv.org/pdf/1803.10158.pdf

Ibru, Bernard, et al. 2017. Optimizing Color Detection with Robotic Vision Sensors for Lane Following and Traffic Sign Recognition in Small Scale Autonomous Test Vehicles. *SAE World Congress*. Detroit, Michigan: SAE International. doi:10.4271/2017-01-0096.Copyright.

Ivakhnenko, A. G. 1973. Cybernetic Predicting Devices. *CCM Information Corporation*, Sacramento, CA. https://books.google.com/books?id=FhwVNQAACAAJ

Juergen Schmidhuber. 2015. Deep Learning. *Scholarpedia*, 10(11):32832. http://www.scholarpedia.org/article/Deep_Learning

LeCun, Yann, et al. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1, pp. 541–551. http://yann.lecun.com/exdb/publis/pdf/lecun-89e.pdf

LeCunn, Yann, et al. 2005. Off-Road Obstacle Avoidance through End-to-End Learning. *Neural Information Processing Systems*. http://yann.lecun.com/exdb/publis/pdf/lecun-dave-05.pdf

Ljubo Vlacic, et al. 2001 a. Intelligent Vehicle Technologies, p. 132. *Elsevier*, Oxford, UK. https://books.google.com/books?hl=en&lr=&id=vREURy3rzlUC&oi=fnd&pg=PP1&dq=computer+vision+for+self+driving+vehicles&ots=6k9SE1GZjf&sig=1XGlbV9kZLeoZwhswyaXJno05-A#v=onepage&q&f=false

Ljubo Vlacic, et al. 2001 b. Intelligent Vehicle Technologies, p. 183. *Elsevier*, Oxford, UK. https://books.google.com/books?hl=en&lr=&id=vREURy3rzlUC&oi=fnd&pg=PP1&dq=computer+vision+for+self+driving+vehicles&ots=6k9SE1GZjf&sig=1XGlbV9kZLeoZwhswyaXJno05-A#v=onepage&q&f=false

Najm, W. et al. 1995. Synthesis Report: Examination of Vehicular Crashes and

Potential ITS Countermeasures. *U.S. Department of Transportation*, Washington, DC. https://web.archive.org/web/20031011103348/http://www.uctc.net/papers/622.pdf

NHTSA. 2005. 2004 Data Summary. *U.S. Department of Transportation*, Washington, DC. https://crashstats.nhtsa.dot.gov/#/

NCHS. 2017. Mortality in the United States, 2017. *Centers for Disease Control and Prevention*. Atlanta, GA. https://www.cdc.gov/nchs/data/databriefs/db328_tables-508.pdf#4

NHTSA. 2019. Automated Vehicles for Safety. *U.S. Department of Transportation*, Washington, DC. https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety#topic-safety-timeline

Schwarz, Brent. 2010. Mapping the World in 3D. *Nature*, New York, NY. https://www.nature.com/articles/nphoton.2010.148.pdf?origin=ppub

Silver, David, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *ArXiv.* Cornell University, Ithaca, NY. https://www.nature.com/articles/nature16961

Stanford. No Date. Convolutional Neural Network. *Stanford*, Stanford, CA. http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

Vázquez, Favio. 2017. Deep Learning Made Easy with Deep Cognition. *Medium.* https://becominghuman.ai/deep-learning-made-easy-with-deep-cognition-403fbe445351

APPENDIX A

Data Recording Program

```
from __future__ import print_function
import sys
import rospy
import cv2
from std_msgs.msg import String
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import roslib
import tf.transformations
from geometry_msgs.msg import Twist
import time
import os

class Record_data:
    def __init__(self):
        self.imgList= list()
        self.steeringList = list()
        self.count = 0
        self.steering_data = [0, 0]

    def launch(self):

        self.bridge = CvBridge()
        img_topic = "/camera/image/image_raw"
        rospy.Subscriber(img_topic, Image, self.callback1)
        velocity_topic = "cmd_vel_mux/input/teleop"
        rospy.Subscriber(velocity_topic, Twist, self.callback2)

    def callback1(self, data):

        # Convert image to OpenCV format
        try:
            cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
        except CvBridgeError as e:
            print(e)

        global image_data
        image_data = cv_image


    def callback2(self, msg):
        self.steering_data = [msg.linear.x, msg.angular.z]


    def take_picture(self):

        # Save an image
        img_title = 'photo' + str(self.count) + '.jpg'
        cv2.imwrite("dataset/images/"+img_title, image_data)
        self.imgList.append(img_title)

        self.count = self.count + 1
        rospy.loginfo(self.count)


    def record_steering(self):
            rospy.loginfo(self.steering_data)
            self.steeringList.append(self.steering_data)

if __name__ == '__main__':

    # The dataset will be saved to "dataset/images".
    if not os.path.exists("dataset/images"):
        os.makedirs("dataset/images")
    # Initialize
    rospy.init_node('record_data', anonymous=False)

    data = Record_data()
    data.launch()

    # break when ctrl + c is pressed
    while not rospy.is_shutdown():
            time.sleep(0.05)
            # save both data every one second.
            data.take_picture()
            data.record_steering()

    # save to csv file
    dataset = list()
    rospy.loginfo("Saving dataset...")
    for img_path, (speed, angle) in zip(data.imgList, data.steeringList):
        newString = "images/" + str(img_path) + ", " + str(speed) + ", " + str(angle)
        dataset.append(newString)
    # rospy.loginfo(dataset)
    # rospy.loginfo(imgList)
```

APPENDIX B

Neural Network

```
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras.layers import Lambda, Conv2D, MaxPooling2D, Dropout, Dense, Flatten
import csv
import cv2
import matplotlib.pyplot as plt
import random
import matplotlib.image as mpimg


data_dir = '/home/keith/Thesis/dataset'
labels_file = '/home/keith/Thesis/dataset/dataset.csv'

def load_data(labels_file, test_size):

    labels = pd.read_csv(labels_file, names=['center', 'speed', 'steering'])
    X = labels['center'].values
    y = labels['steering'].values
    X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=test_size, random_state=0)

    return X_train, X_valid, y_train, y_valid


def load_image(data_dir, image_file):

    return mpimg.imread(os.path.join(data_dir, image_file.strip()))


data = load_data(labels_file, 0.2)


IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS = 66, 200, 3
INPUT_SHAPE = (IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)

def preprocess(img):

    img = cv2.resize(img, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
    return img

def aug_image(data_dir, center, steering_angle):

    return load_image(data_dir, center), steering_angle

def random_flip(image, steering_angle):

    if np.random.rand() < 0.5:
        image = cv2.flip(image, 1)
        steering_angle = -steering_angle
    return image, steering_angle


def random_shadow(image):

    bright_factor = 0.3
    x = random.randint(0, image.shape[1])
    y = random.randint(0, image.shape[0])c
    width = random.randint(image.shape[1], image.shape[1])
    if (x + width > image.shape[1]):
        x = image.shape[1] - x
    height = random.randint(image.shape[0], image.shape[0])
    if (y + height > image.shape[0]):
        y = image.shape[0] - y
    image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    image[y:y + height, x:x + width, 2] = image[y:y + height, x:x + width, 2] * bright_factor
    return cv2.cvtColor(image, cv2.COLOR_HSV2RGB)

def random_brightness(image):

    hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    ratio = 1.0 + (np.random.rand() - 0.5)
    hsv[:, :, 2] = hsv[:, :, 2] * ratio
    return cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)

def augument(data_dir, center, steering_angle):

    image, steering_angle = aug_image(data_dir, center, steering_angle)
    image, steering_angle = random_flip(image, steering_angle)
    image = random_shadow(image)
    image = random_brightness(image)
    return image, steering_angle
```

```
def NVIDIA_model():
    model = Sequential()
    model.add(Lambda(lambda x: x / 127.5 - 1.0, input_shape=INPUT_SHAPE))
    model.add(Conv2D(24, 5, 5, activation='relu', subsample=(2, 2)))
    model.add(Conv2D(36, 5, 5, activation='relu', subsample=(2, 2)))
    model.add(Conv2D(48, 5, 5, activation='relu', subsample=(2, 2)))
    model.add(Conv2D(64, 3, 3, activation='relu'))
    model.add(Conv2D(64, 3, 3, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1))
    model.summary()
    return model




batch_size = 32
samples_per_epoch = 300
nb_epoch = 250


def batcher(data_dir, image_paths, steering_angles, batch_size, training_flag):

    images = np.empty([batch_size, IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS])
    steers = np.empty(batch_size)
    while True:
        i = 0
        for index in np.random.permutation(image_paths.shape[0]):
            center = image_paths[index]
            steering_angle = steering_angles[index]
            if training_flag and np.random.rand() < 0.6:
                image, steering_angle = augument(data_dir, center, steering_angle)
            else:
                image = load_image(data_dir, center)
            images[i] = preprocess(image)
            steers[i] = steering_angle
            i += 1
            if i == batch_size:
                break
        yield images, steers


def train_model(model, X_train, X_valid, y_train, y_valid):
    checkpoint = ModelCheckpoint('model-{val_loss:03f}.h5',
                                 monitor='val_loss',
                                 verbose=0,
                                 save_best_only=True,
                                 mode='auto')
    model.compile(loss='mse', optimizer=Adam(lr=.0001))
    model.fit_generator(batcher(data_dir, X_train, y_train, batch_size, True),
                        samples_per_epoch,
                        nb_epoch,
                        max_q_size=1,
                        validation_data=batcher(data_dir, X_valid, y_valid, batch_size, False),
                        nb_val_samples=len(X_valid),
                        callbacks=[checkpoint],
                        verbose=1)
    json_string = model.to_json()
    with open('./model.json', 'w') as f:
        f.write(json_string)

model = NVIDIA_model()
train_model(model, *data)
```

APPENDIX C

Runner

```
import rospy
from SteeringNode import SteeringNode
import argparse
import json
from scipy import misc
from keras.optimizers import SGD
from keras.models import model_from_json, load_model
import utils
import numpy as np
import thread
import tensorflow as tf
from geometry_msgs.msg import Twist
import time
def process(model, img):
    if img is not None:
        # print(img)
        # print(img.shape)
        img = misc.imresize(img[:, :, :], (66, 200, 3))
        # print(img.shape)
        img = utils.rgb2yuv(img)
        # print(img.shape)
        img = np.array([img])
        # print(img.shape)
        # print('\n\n')
        # steering_angle = model.predict(img[None, :, :, :])[0][0]
        steering_angle = float(model.predict(img, batch_size=1))
        print(steering_angle)
        pub_steering(steering_angle)


def get_model(model_file):

    with open(model_file, 'r') as jfile:

        model = model_from_json(jfile.read())

    # sgd = SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
    # model.compile(sgd, "mse")
    model.compile("adam", "mse")

    weights_file = model_file.replace('json', 'h5')
    model.load_weights(weights_file)

    # return model
    # graph = tf.get_default_graph()
    return model

def pub_steering(steering):
    move_cmd = Twist()
    move_cmd.linear.x = 0.8
    move_cmd.angular.z = steering

    node.pub.publish(move_cmd)

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Model Runner')
    parser.add_argument('model', type=str, help='Path to model definition json. \
                        Model weights should be on the same path.')

    args = parser.parse_args()
    model = get_model(args.model)
    node = SteeringNode()

    # rospy.Timer(rospy.Duration(1), process(model, node.img))

    # rospy.spin()

    while not rospy.is_shutdown():
            time.sleep(0.01)
            # save both data every one second.
            process(model, node.img)
```