Sample-Efficient Learned Self-Driving through a Variational Autoencoder

Manav Gagvani

Thomas Jefferson High School for Science and Technology – Computer Systems Research Lab

Introduction

Each year in Virginia, there are over 110,000 traffic accidents resulting in injury or death. The United Nations further estimates that there are 1.19 million traffic-related deaths worldwide, causing a $1.8 trillion loss to the global economy. A majority of these deaths are a result of human error. The solution to these problems is fully autonomous vehicles (AVs). However, current solutions to self-driving cars are insufficient due to their inefficiency.

Prior Work: Imitation Learning Using CNNs

Nvidia's PilotNet architecture, created in 2016, is a CNN (Convolutional Neural Network) which takes in pictures of a road as input and returns the optimal steering angle as output.

To train such a model, a human driver must drive the car manually to record pairs of associated road images and steering angles. Nvidia's first try required 72 hours of driving data – three day's worth. This is quite inefficient as the model can only work on roads it has been trained on – it does not have the ability to generalize further.

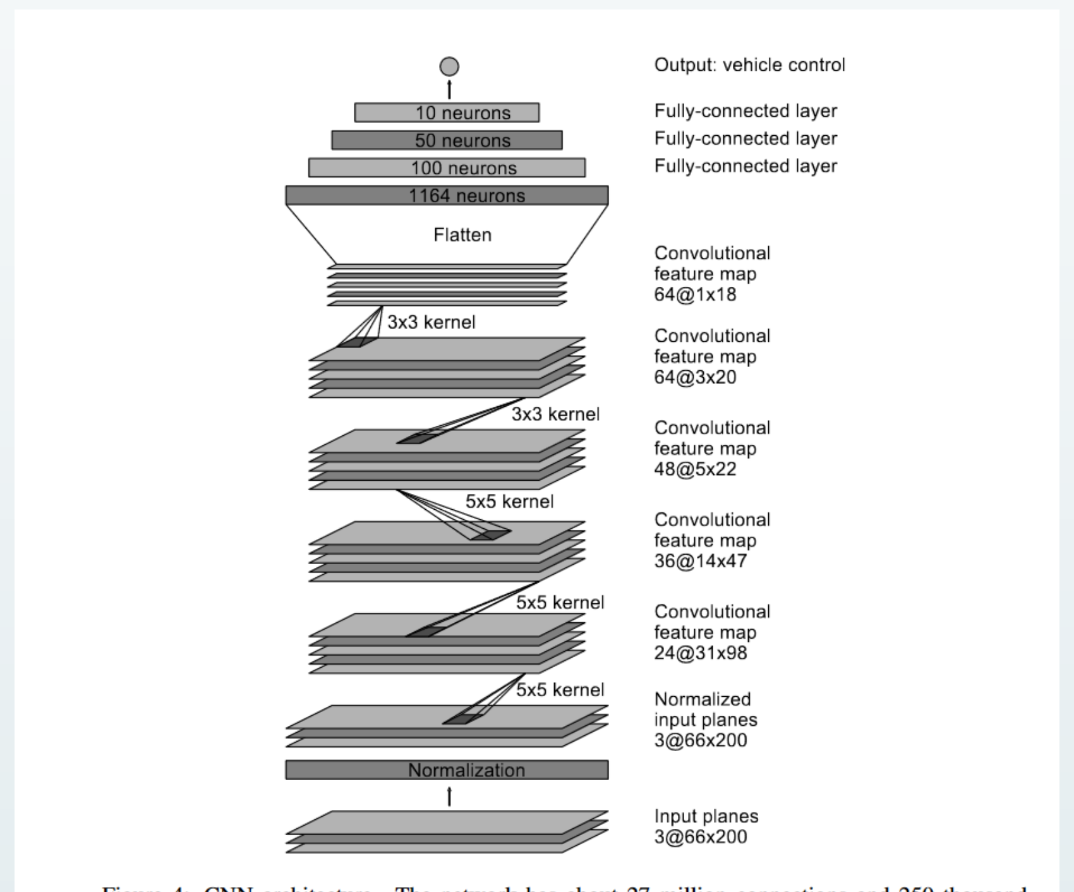


Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

Fig. 1: Diagram of the PilotNet CNN architecture.

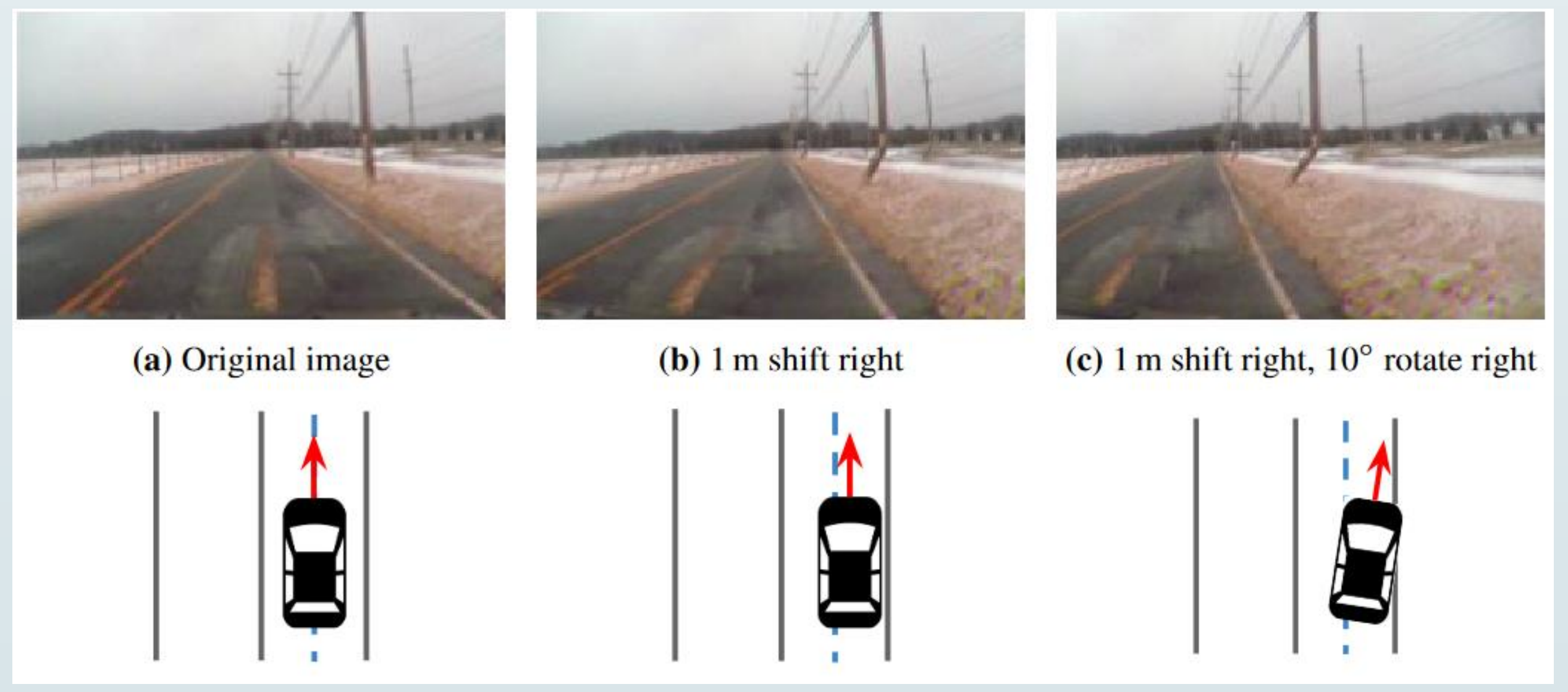


Fig. 2: Example of how the dataset was artificially augmented to increase the robustness of the system. The CNN is trained on images from the car centered on the road, but minor errors can lead to translation or rotational error relative to this point.

Prior Work: Reactive Methods

Reactive Methods are a class of controller for self-driving vehicles which use distance data to come up with a mathematical estimate of the "best course of action". Distance information is most commonly provided by LiDAR (Light Detection And Ranging) sensors. LiDAR sensors measure the time it takes for a beam of light to travel to an obstacle and get reflected back to the sensor in order to measure distance.

There are several variations of the basic Reactive Method, which is to turn away from the closest obstacle. These include Follow the Gap (FTG), which intelligently incorporates the physical characteristics of a car, and Disparity Extender, which is especially suited for cars in a racing scenario. These approaches traditionally incorporate motion data from an IMU (Inertial Measurement Unit) in addition to a map created through individual LiDAR scans.

Reinforcement Learning In Simulation

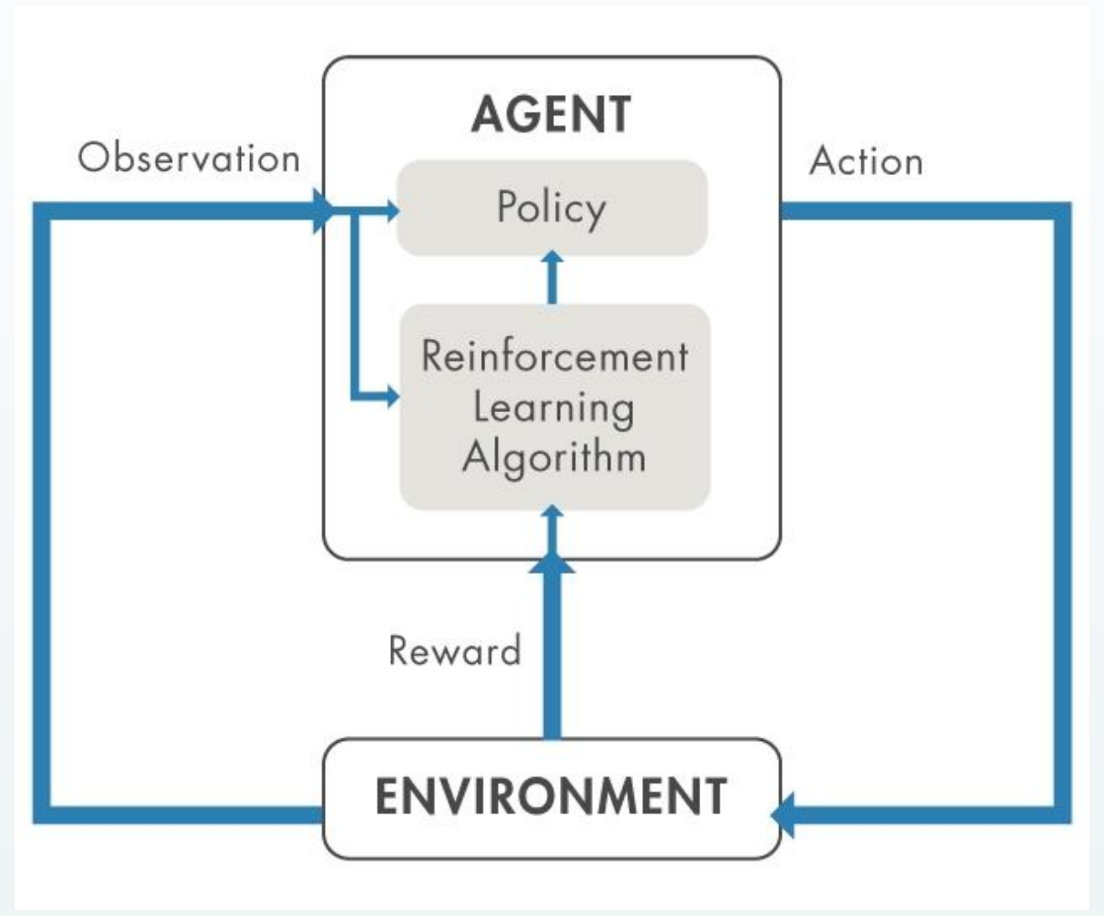


Fig. 3: Diagram of the reinforcement learning process. Agents receive a reward by performing actions in an environment. This informs their policy, a strategy for getting the highest reward by choosing the optimal actions.

Problem: "Standard" reinforcement learning algorithms take a significant time to train. A widely used reinforcement learning algorithm, PPO (Proximal Policy Optimization), took 48 hours to train a simulated car which was being simulated at 4x real-time. *This would not be practical to fine-tune on a car in real life.*

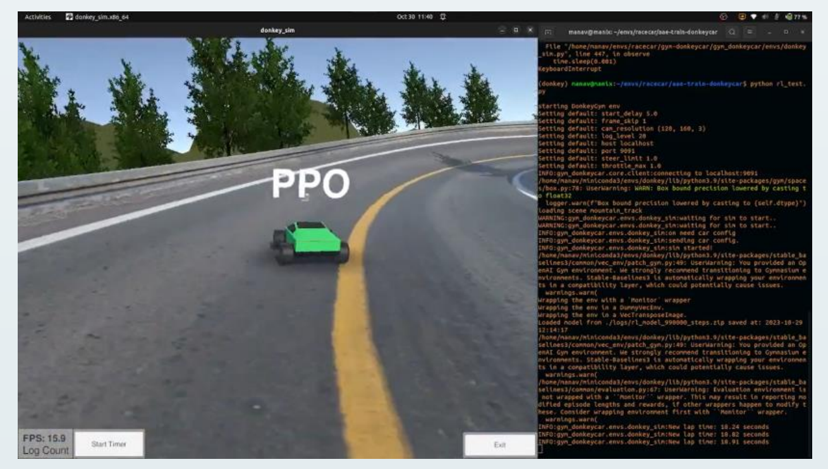
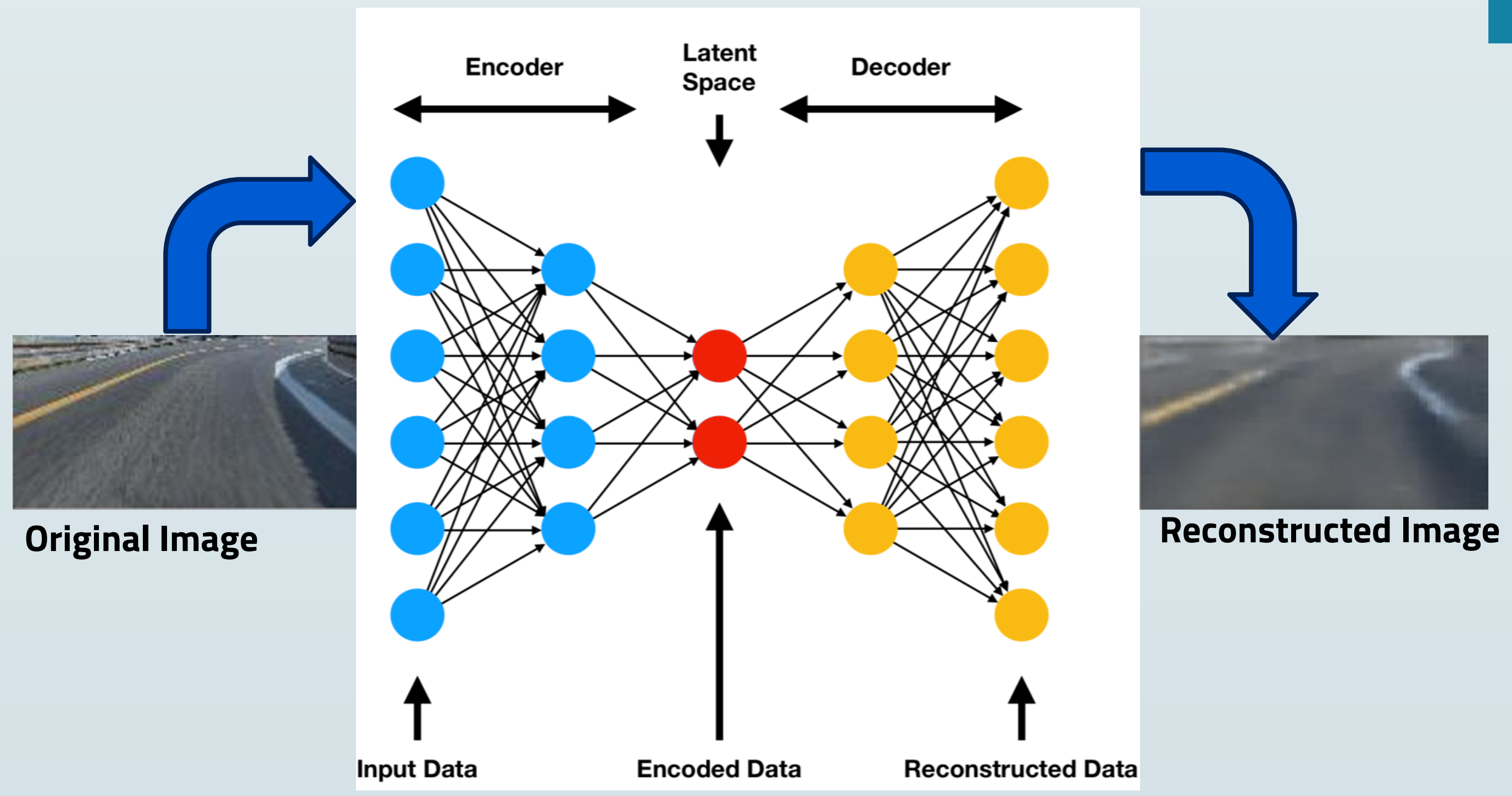


Fig. 4: A self-driving car in the process of training using reinforcement learning in simulation.

Idea: Reduce the "observation space" of the reinforcement learning model, e.g., the dimensionality of the input it has to work with. Images are large – a 640 by 480 image is almost a million individual numbers. Images can be encoded into a compressed representation using an *autoencoder.*



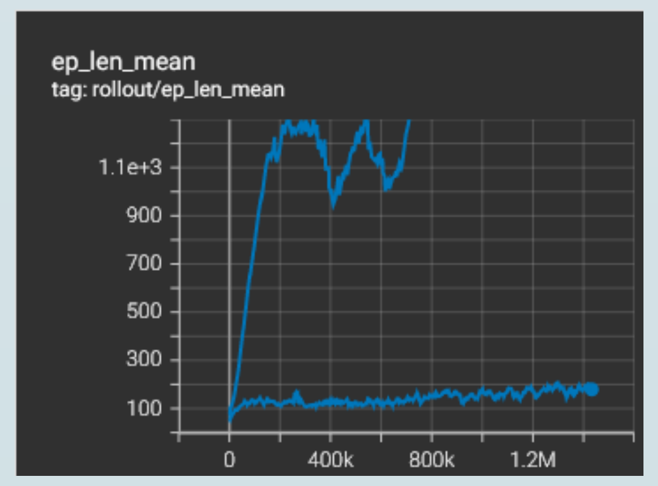


Fig. 5: Episode length shows the number of seconds before a crash with and without the autoencoder. Using an autoencoder reduced the amount of crashes as compared to raw image input given the same amount of training time. The average lap time decreased from 18 seconds to 15.5 seconds.

Physical 1/10 Scale Car

Given that it would be impractical to test algorithms on a full-scale car, researchers from the University of Pennsylvania, University of Texas at Austin, and MIT have used 1/10 scale R/C (remote-controlled) cars as bases to build miniature self-driving cars. These scaled self-driving cars contain many of the same components real self-driving cars do, but at a smaller scale, allowing for more rapid and inexpensive prototyping of both AV hardware and software. Common configurations of these miniature AVs contain cameras, LiDAR sensors and IMUs for sensing, a microprocessor for doing computation and some form of programmatic motor controller to drive the steering servo motor and throttle motor.

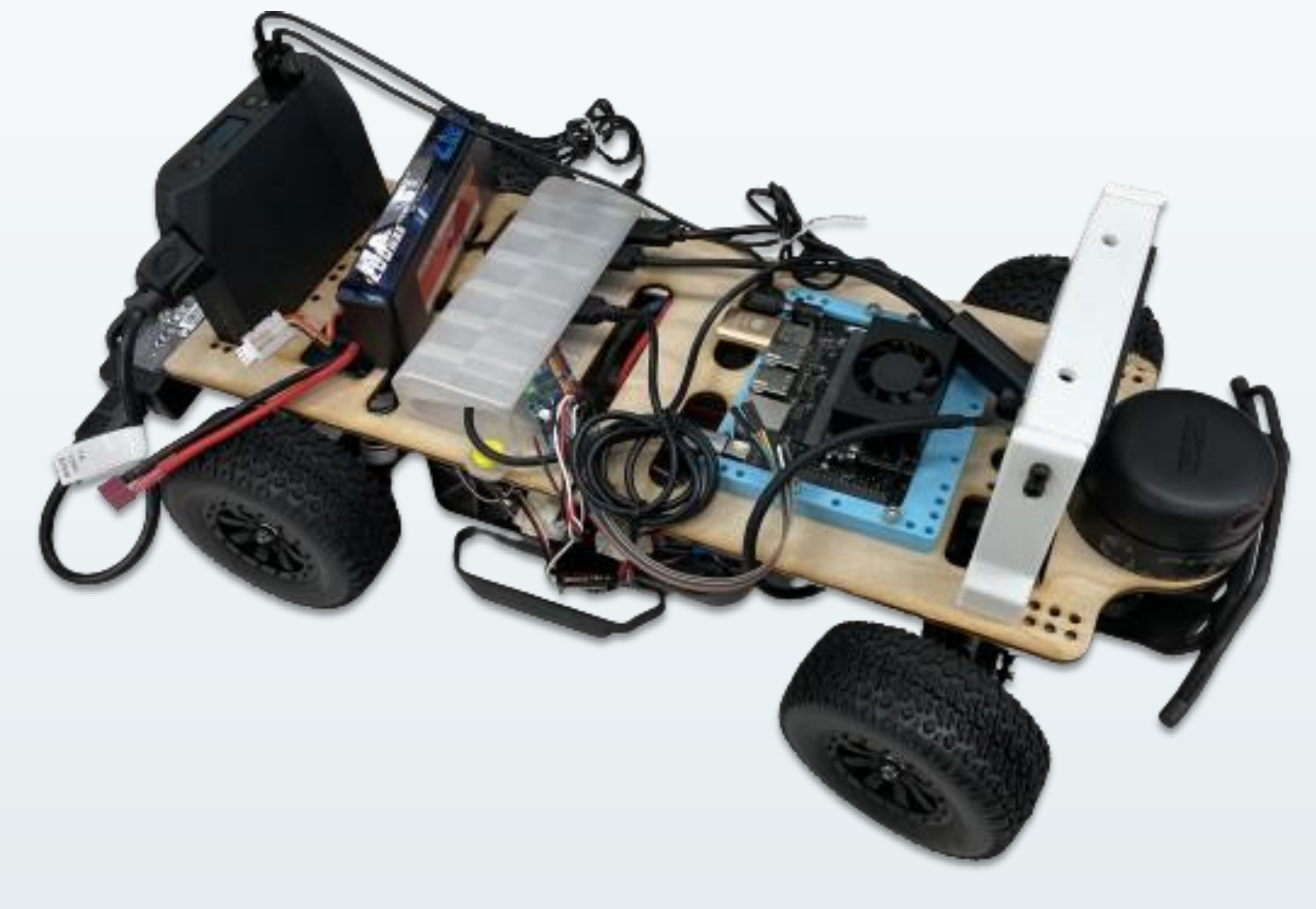


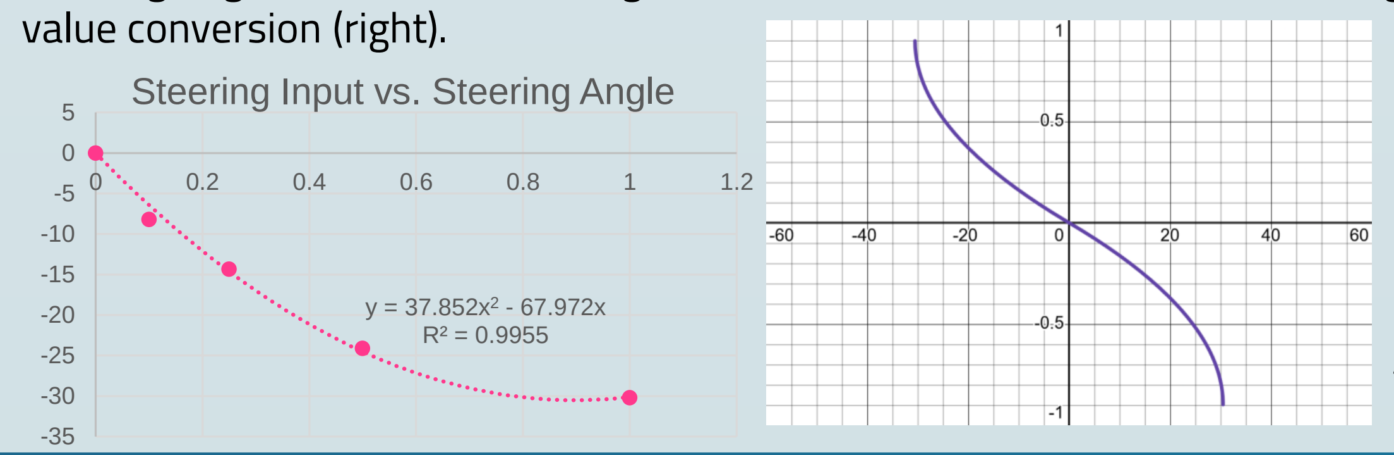
Fig. 6: The miniature self-driving car I built on a 1/10 scale chassis. From left to right, the components mounted are two battery packs, PCA9685 I2C multiplexer, Nvidia Jetson Orin Nano SBC (Single-Board Computer), OAK-D Pro stereo depth camera, and RPLidar A2 rotational LiDAR sensor.

Simulation To Reality

Since algorithms are developed in simulation, they need to be ported Specifically, the real miniature AV does not have perfect LiDAR maps, since it needs to reconstruct it from scans with limited speed and accuracy. Furthermore, camera images are susceptible to motion blur, and glare from reflective surfaces. Finally, the car does not have unlimited battery life, and thus cannot be trained continuously as is done in simulation. Furthermore, virtual cars and real cars are interfaced with differently:

- Steering
  - *Simulated car:* Steering angle according to the kinematic bicycle model
  - *Real car:* Steering value from -1 to 1, with -1 being fully left and 1 being fully right
- Throttle
  - *Simulated car:* Velocity in meters per second
  - *Real car:* Throttle value from -1 to 1, with -1 being full speed backwards and 1 being full speed forwards

Thus, conversions need to be made to account for this discrepancy. To calculate true steering angles, I used a laser rangefinder attached to the wheel to calculate the true steering angle of different steering values (left). This was converted to an angle → value conversion (right).



$y = 37.852x^2 - 67.972x$
$R^2 = 0.9955$

Fig. 7: Conversions and data collected to convert angles to steering values.

Implementation of Simulated Algorithms in Real Life

After the appropriate conversions and translations were made to make previous simulations match reality, various algorithms, which are different approaches to the self-driving problem, were implemented.

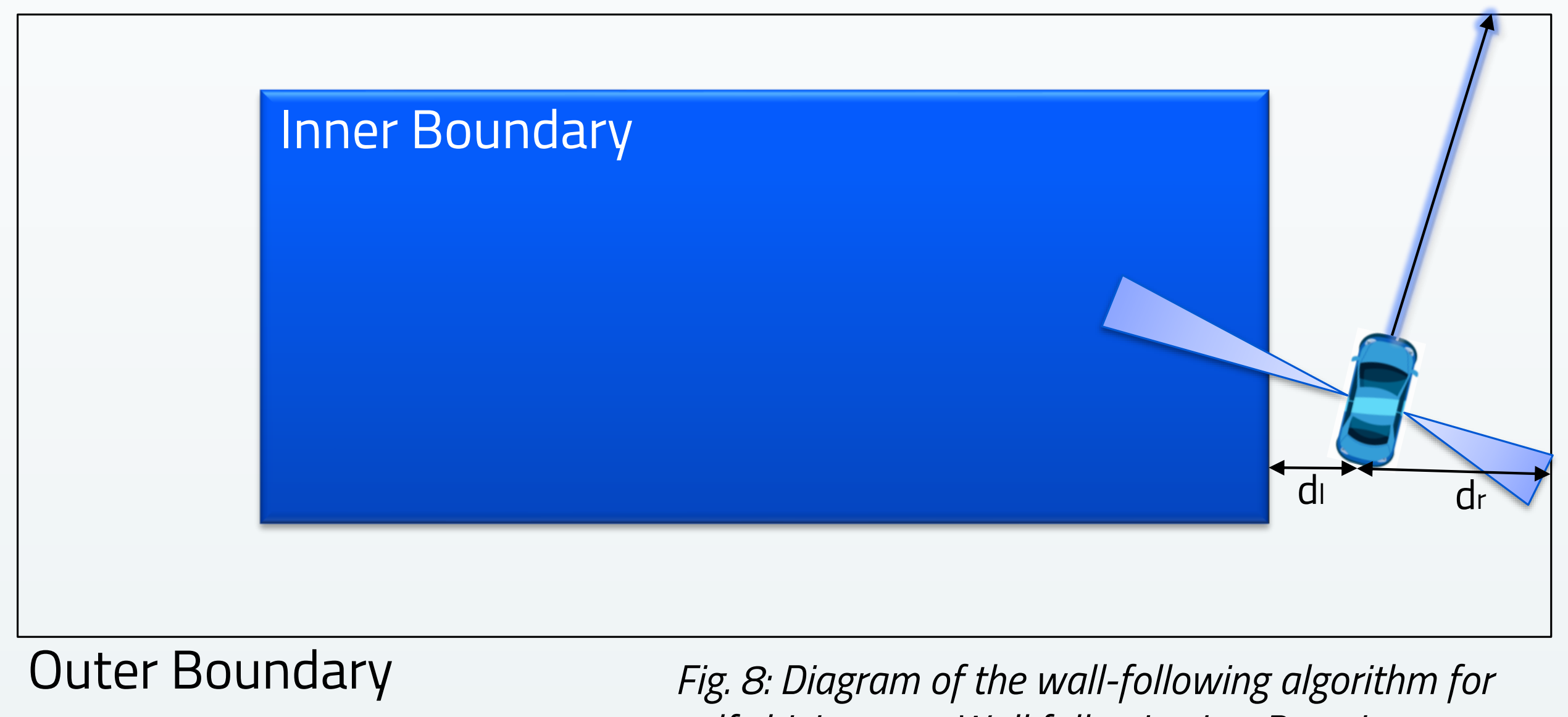
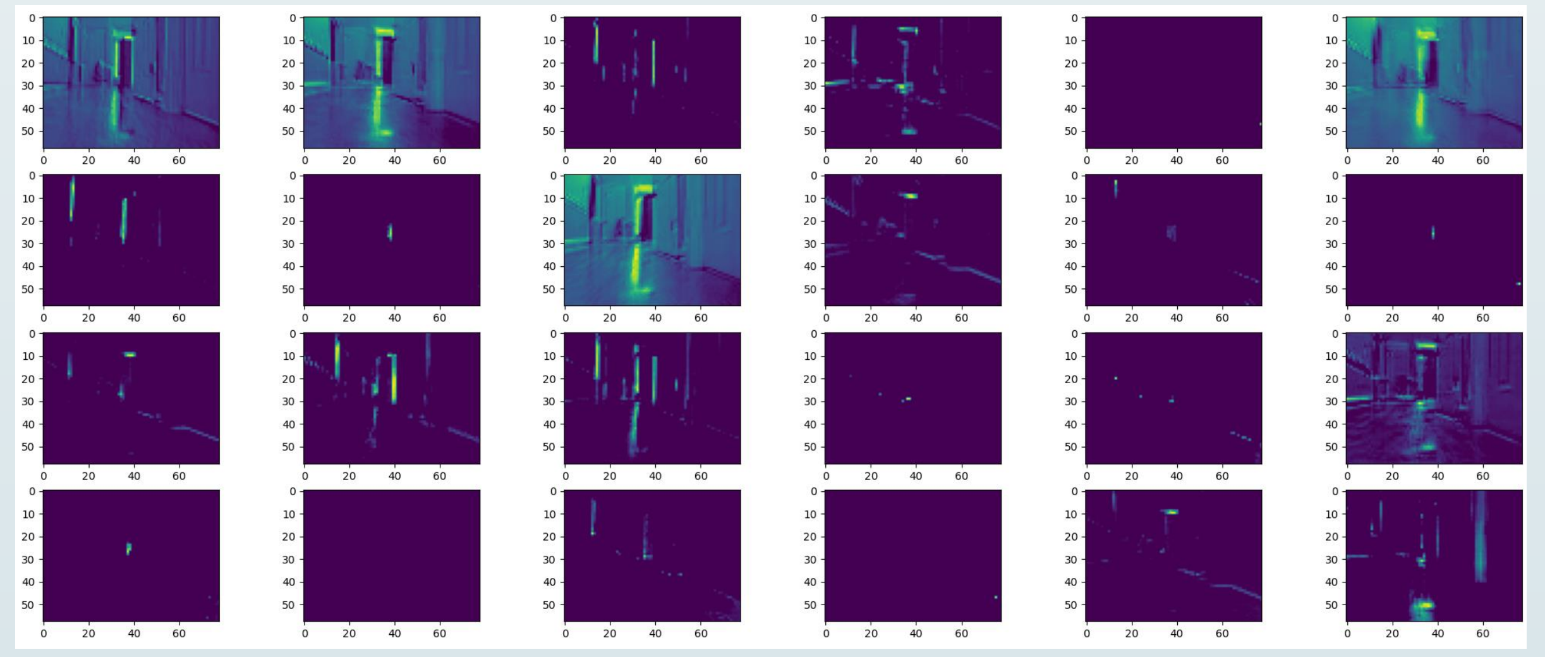


Fig. 8: Diagram of the wall-following algorithm for self-driving cars. Wall following is a Reactive Method which incorporates LiDAR map data to maintain constant distance from two opposing walls.. It does this by holding (d_l – d_r) to 0 through a PID (Proportional-Integral-Derivative) controller.

Fig. 9: Activations from a PilotNet CNN trained on images taken from driving in an indoor environment. The first layer of activations shows the CNN "learned" key features of the environment such as the importance of doors as spatial landmarks. The car was trained by manually driving it with a game controller and recording the images associated with steering and throttle input values. Over 20,000 images were collected and the final trained model had a prediction accuracy of 92% over the entire dataset.



Future Work

A large focus in academia currently is simulation to reality transfer of advanced AV algorithms which incorporate neural networks, such as supervised learning or unsupervised reinforcement learning. Possibilities for future work include implementing reinforcement learning in a safe manner on a real car, incorporating object detections through YOLO or another method, or incorporating V2I (Vehicle-To-Infrastructure) communications to better model real-world AV systems. The technical problem of self-driving cars has largely been solved, except for rare edge cases which are hard to predict.

Acknowledgements

I would like to thank Dr. Gabor, my Senior Research teacher, and Dr. Torbert, who has been immensely helpful in buy parts needed to assemble the car. I would also like to thank Shreyan Dey and Christopher Lim for their help designing and fabricating the mechanical assembly.