Lateral and Longitudinal Motion Control of Autonomous Vehicles using Deep Learning

Shobit Sharma
Mechanical Engineering
Kettering University
Flint, MI 48504
shar0287@kettering.edu

Girma Tewolde
Electrical and Computer Engineering
Kettering University
Flint, MI 48504
gtewolde@kettering.edu

Jaerock Kwon
Electrical and Computer Engineering
Kettering University
Flint, MI 48504
jkwon@kettering.edu

***Abstract*—The Current trend of the automotive industry combined with active research by the major tech companies has proven that self-driving vehicles are the future. The biggest challenge for self-driving cars is autonomous lateral and longitudinal control. An end-to-end model seems very promising in providing a complete software stack for autonomous driving. The work described in this paper focuses on how a deep learning technique is utilized for implementing both lateral and longitudinal control of vehicles. The open racing car simulator (TORCS) is used for developing and testing the implementation. Two separate neural networks were trained that can predict the vehicle speed and steering based on the road trajectory. Such an approach serves as a foundation towards building a system that utilizes artificial intelligence to analyze the environment and determine what the vehicle speed should be rather than following a set of predetermined rules.**

I. INTRODUCTION

In the past decade, the automotive industry has seen significant technological advancements in the field of autonomous vehicles. Joint ventures between car manufacturers and tech giants like Google and Uber are advancing the development of the next-generation autonomous vehicle solutions. Manufacturers such as Tesla and General Motors, to name a few, have already demonstrated some level of autonomy in their premium vehicles. Features like Lane Departure Warning Systems [1], Lane Keep Assist [2] and, Adaptive Cruise Control [3] are now common.

Among the most important functionalities of self-driving vehicles are autonomous lateral and longitudinal controls. The conventional technique for autonomous motion control relies on image processing. The task is divided into detecting lanes, finding lane centers, path planning, path following, and applying a control logic to control the speed and steering of the vehicle. The accuracy of such systems relies significantly on how well tuned the image processing filters are. Operations such as blurring [4], Canny edge-detection [5] and Hough transforms [6] are done on the image to detect lanes and get rid of unwanted information. These techniques are susceptible to variations in lighting and are prone to a lot of false detections. An algorithm tuned to detect lane markings in a place where it is bright and sunny may not do well in a place where it is dark and gloomy. A previous report from NVIDIA [7] demonstrated an end-to-end learning model for autonomous lateral control. This system uses a convolutional neural network [8] to output steering angles based on road images. Essentially, the model is trained to mimic human driving behavior. Since an end-to-end model does not operate on manually defined rules, it is less affected by changes in lighting conditions.

In our previous paper [9] we presented end-to-end learning for lateral control of an autonomous vehicle. In this paper, we extend the work to handle both lateral and longitudinal control of the vehicle. The rest of this paper presents the two alternative approaches explored for the task at hand, demonstrate results in simulation, and provide discussions with final concluding remarks.

II. DRIVING SIMULATOR

For ease of use and safety, we use a simulation environment for training and testing our system. A simulation environment was preferred primarily to save time and labor required for data collection and labeling. The open racing car simulator (TORCS) [10] is a state of the art simulator that allows users to develop their own vehicle controllers. Figure 1 shows a screen shot of TORCS. In simulation, data can be accessed and information about the car such as speed, steering angle, throttle, and brake positions are readily available. A standard install of TORCS, in which a human can control the vehicle inside the simulation is used for data collection. For testing, SCR-server patch must be installed to interface a custom controller to control the vehicle inside simulation. This patch establishes a client-server connection in which a user-defined client can connect to the simulation server and control vehicle.

III. VEHICLE CONTROL OPTIONS

With the two degrees of freedom (speed and steering control) of the vehicle, there are two design approaches that we explored. The following sub-sections discuss each one of them.

Fig. 1. The TORCS driving simulator

A. Single model

A single model was trained to have two outputs: the vehicle speed and steering angle. Various architectures were tried, but none of them provided good results. In all attempts, the model was unavailable to extract a relation between input images and the desired output. In most cases, the model would make the same predictions for different input scenarios. Typically, deep learning [8] algorithms are good at predicting multiple output classes but do not fare too well when they are required to make separate predictions. For such an application, reinforcement learning techniques [11] work better where the network learns over time by making mistakes and being penalized for making errors. If the goal was to predict, for example, whether there was a traffic sign present in the input image, deep learning [12] would have sufficed. This is since the model learns by going through training data and extracting features from the input and mapping them to the output.

As we were relying on behavioral cloning [9] and expecting the model to define a relation between the input and two separate output entities, the model didn't succeed. If the model is required to make only one prediction, it can extract features and map them to the required output. An important thing to note is that the model does not map lane markings to steering angles. Rather, it just extracts features and learns that if input looks a certain way, the prediction increases or decreases. In our case, as the white lanes provide a significant change in pixel intensity over a gray road, they are one of the most prominently extracted features. It goes without saying that the presence of objects, such as traffic lights, pedestrians, other cars or even the horizon can affect what features are extracted. Therefore, for behavioral cloning [9], unnecessary part of the input image is discarded before feeding to the network [8].

With the addition of another degree of freedom, such relations become harder to establish. However, this does not imply that deep learning [12] cannot be used to control lateral and longitudinal motion with a single model. Due to complications involved, it was decided to go for two separate models and run them in parallel on the same computer.

B. Two separate models

Two separate models were trained to predict speed and steering angle. They were run in parallel on different threads, and the predictions are used to control the vehicle.

1) Steering model

Table 1 shows the network [9] architecture to control the vehicles lateral motion.

Table I: Lateral control architecture

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 70, 160, 3)	0
conv2d_1 (Conv2D)	(None, 70, 160, 24)	1824
max_pooling2d_1 (MaxPooling2D)	(None, 69, 159, 24)	0
conv2d_2 (Conv2D)	(None, 69, 159, 36)	21636
max_pooling2d_2 (MaxPooling2D)	(None, 34, 79, 36)	0
conv2d_3 (Conv2D)	(None, 34, 79, 48)	43248
max_pooling2d_3 (MaxPooling2D)	(None, 17, 39, 48)	0
conv2d_4 (Conv2D)	(None, 17, 39,64)	76864
max_pooling2d_4 (MaxPooling2D)	(None, 8, 19, 64)	0
conv2d_5 (Conv2D)	(None, 8, 19, 64)	102464
max_pooling2d_5 (MaxPooling2D)	(None, 4, 9, 64)	0
flatten_1 (Flatten)	(None, 2304)	0
dropout_1 (Dropout)	(None, 2304)	0
dense_1 (Dense)	(None, 256)	590080
dropout_2 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 1)	65

The selected architecture has 877,833 parameters or adjustable weights for an input image of size 70 * 160 * 3. This is about ten times more than the number of weights in the architecture used for the speed model (see Table 2 below). A heavier architecture was selected because of two reasons: the model was required to control the vehicle in a much more complex track, and a GPU was available to run the network on. Also, the input image size was larger than that from the speed control challenge. This was done to ensure that the input image has enough resolution for the model to extract features and map them to steering angles for more convoluted road trajectories. Each convolution [13] was done with a 5*5 kernel [13] followed by a 2*2 max-pooling [14].

2) *Speed model*

As can be seen in Table 2, the model used for controlling vehicle speed was lighter in architecture with only 77,525 parameters for an input image size of 64 * 64 * 3. This is because the model was expected to merely predict whether the vehicle speed should increase or decrease based on the curvature of the road ahead. For all convolutions, a 4*4 kernel [13] was used followed by a 2*2 max-pooling [14]. A smaller kernel size was selected in order to be able to retain more features from the smaller input image. A larger kernel convolves over a bigger section of the input and generates a feature map of lower resolution. As the input image was very small, it was decided to reduce the kernel size to be able to extract a feature map of higher resolution [13].

Table 2: Speed control architecture

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 64, 64, 3)	0
conv2d_1 (Conv2D)	(None, 64, 64, 3)	12
conv2d_2 (Conv2D)	(None, 64, 64, 8)	392
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 8)	0
conv2d_3 (Conv2D)	(None, 32, 32, 16)	2064
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 16)	0
conv2d_4 (Conv2D)	(None, 16, 16, 24)	6168
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 24)	0
conv2d_5 (Conv2D)	(None, 8, 8, 24)	9240
max_pooling2d_4 (MaxPooling2D)	(None, 4, 4, 24)	0
flatten_1 (Flatten)	(None, 384)	0
dropout_1 (Dropout)	(None, 384)	0
dense_1 (Dense)	(None, 128)	48280
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 32)	2080
dense_4 (Dense)	(None, 1)	33
lambda_1 (Lambda)	(None, 64, 64, 3)	0
conv2d_1 (Conv2D)	(None, 64, 64, 3)	12
conv2d_2 (Conv2D)	(None, 64, 64, 8)	392

Based on the training data, the model outputs a speed prediction, and the accuracy of predictions is largely dependent on the training sets quality. During initial data collection runs, vehicle speed was recorded in multiples of five, i.e. speed measurements were rounded off to the nearest multiples of five. This was done to ensure that the expected output from the model was consistent. In another attempt, a simple closed loop controller was used to control vehicle speed during data collection. All these efforts were made with the intention of stabilizing the target speed for the network [8]. However, this affected performance and defeated the purpose of having artificial intelligence predict vehicle speed. The resultant networks would end up predicting the same or almost the same speed for all scenarios irrespective of changes in the road curvature. The vehicle would not increase or decrease its speed based on the road's trajectory ahead. Eventually, raw speed measurements were recorded with a human driver in control and this data was used for training. Initially, the trained models were not acceptable as the predictions they made were not consistent. They would extract a relationship between road trajectory and target speed, but it was not consistent for all portions of the track. To combat this issue, more training data was recorded and eventually, a state was reached wherein the model could successfully predict speed for all portions of the track.

IV. Software Architecture

A crucial aspect while designing the software flow was to ensure that there are no bottlenecks. To achieve this, multi-threading was used, and there was a total of five threads running. Table 3 shows information on the threads running.

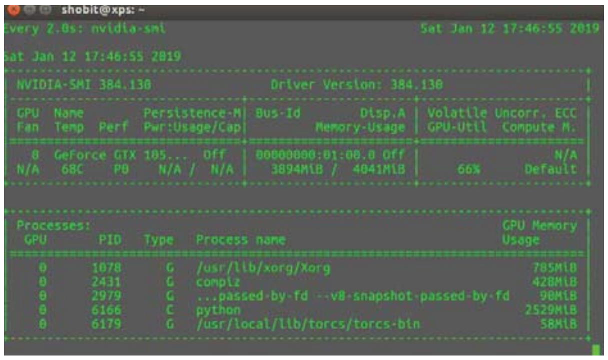
Table III: Threading information

Thread	FPS
Screenshot	10
Steering	95
Speed	105
TORCS	350
Main	NA

The main thread was merely used to start other threads and no processing was done in it. After all other threads had been initialized; the main thread was not killed. This was done to be able to catch interrupts and shut down the other threads. Failure to do this prevents the threads running models from releasing GPU resources which eventually results in exhausting GPU RAM, resulting in out of memory errors. Since both the models rely on images from the TORCS simulator for input and provide predictions that go to TORCS, separate threads were used for getting screenshots and passing predictions to TORCS. This avoids doing the same thing twice and speeds up the rate at which the algorithms run. As both models use images of different size, the input coming from the screenshot thread is resized to the correct one in their respective threads. Figure 2 shows the resource utilization on the GPU. The two models running simultaneously consume approximately 2.5 GB of GPU RAM.

V. Data Acquisition

Figure 3 shows the layout of the track used for collecting training data. The vehicle speed, steering angle, and road images were recorded during data acquisition. As can be seen from the figure, the track consisted of a combination of straight sections and turns of varying radii. The same dataset was used for training both speed and steering models.



```
shobit@xps: ~
Every 2.0s: nvidia-smi                                Sat Jan 12 17:46:55 2019

Sat Jan 12 17:46:55 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.130                Driver Version: 384.130                   |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 105...  Off  | 00000000:01:00.0 Off |                  N/A |
| N/A   68C    P0    N/A /  N/A |   3894MiB /  4041MiB |     66%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1078      G   /usr/lib/xorg/Xorg                            785MiB |
|    0      2431      G   compiz                                        428MiB |
|    0      2979      G   ...passed-by-fd --v8-snapshot-passed-by-fd     98MiB |
|    0      6166      C   python                                       2529MiB |
|    0      6179      G   /usr/local/lib/torcs/torcs-bin                 58MiB |
+-----------------------------------------------------------------------------+
```

Fig. 2: GPU utilization

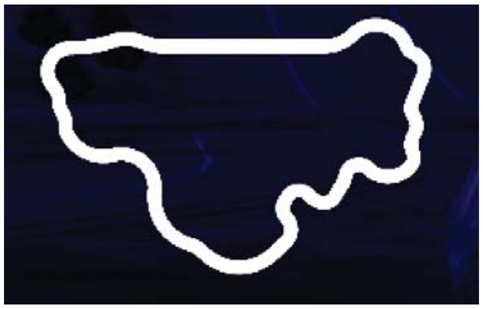


Figure 3: TORCS track (e-road) used for data acquisition

VI. Results

The models trained can successfully control the car and complete entire laps without going over the lane markings. A 100% autonomy was achieved on e-road track. Equation 3 shows the relation used to calculate autonomy. The trained models were also able to complete CG-track 2 completely with 100% autonomy. Figure 4 shows the layout of CG-track 2.

$$a = \left(1 - \frac{N*2}{T}\right) * 100 \qquad (3)$$

Where *a* represents the degree of autonomy, N is the number of lane changes the vehicle made while completing the track, and T is the total time elapsed in seconds. If the vehicle changes lanes during the testing phase, it is considered undesirable since the training dataset did not include intentional lane changes.

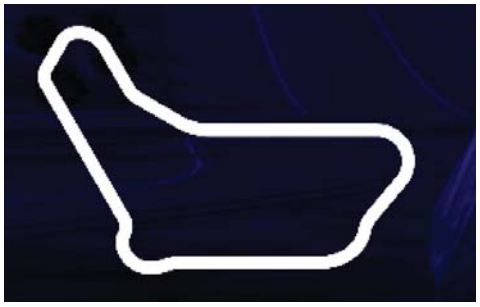


Figure 4: CG -track 2

Due to the limited nature of the training data that was used, the speed and steering models were not able to achieve the 100% autonomy in other TORCS [10] tracks. In some tracks, the road conditions are very different from what the models have been trained with. The presence of cracks on the road can be misleading and sometimes cause unnecessary steering maneuvers. As the models were able to complete laps on CG-track 2, it can be assumed that there is no over-fitting for the e-road track on which data was collected.

An important observation while testing the two models on different tracks was that the speed model worked very well. Speed predictions would go up if the road ahead looked clear of any turn and in case of the car crossing the lanes, speed would go down by a lot. In some cases, the speed prediction even lowered down to 20 kph. This is remarkable considering that all training data was collected with speeds in the range of 40 – 70 kph.

VII. Conclusion

Overall, with the limited training data presented in this paper, the system performed well on the two tracks it was tested on. There is still room for improvement. Work will continue to improve the system by collecting more data from different tracks, especially consisting of different road conditions. This is expected to achieve successful control of both lateral and longitudinal motion in all TORCS tracks.

References

[1] J. He, H. Rong, J. Gong and W. Huang, "A Lane Detection Method for Lane Departure Warning System," 2010 International Conference on Optoelectronics and Image Processing, Haiko, 2010, pp. 28-31.

[2] A. Mammeri, G. Lu and A. Boukerche, "Design of lane keeping assist system for autonomous vehicles," 2015 7th International Conference on New Technologies, Mobility and Security (NTMS), Paris, 2015, pp. 1-5.

[3] N. Benalie, W. Pananurak, S. Thanok and M. Parnichkun, "Improvement of adaptive cruise control system based on speed characteristics and time headway," 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, St. Louis, MO, 2009, pp. 2403-2408.

[4] J. Flusser, S. Farokhi, C. Höschl, T. Suk, B. Zitová and M. Pedone, "Recognition of Images Degraded by Gaussian Blur," in IEEE Transactions on Image Processing, vol. 25, no. 2, pp. 790-806, Feb. 2016.

[5] J. Canny, "A Computational Approach to Edge Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986.

[6] Dagao Duan, Meng Xie, Qian Mo, Zhongming Han and Yueliang Wan, "An improved Hough transform for line detection," 2010 International Conference on Computer Application and System Modeling (ICCASM 2010), Taiyuan, 2010, pp. V2-354-V2-357.

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

[8] T. Guo, J. Dong, H. Li and Y. Gao, "Simple convolutional neural network on image classification," 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)(, Beijing, 2017, pp. 721-724.

[9] S. Sharma, J. Kwon, and G. Tewolde, "Behavioral Cloning for Lateral Motion Control of Autonomous Vehicles using Deep Learning," the 18th Annual IEEE International Conference on Electro Information Technology (EIT 2018).

[10] The Open Racing Car Simulator, [Online]. Available: http://torcs. sourceforge.net/

[11] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J-M. Allen, V-D. Lam, A. Bewley, A. Shah, "Learning to Drive in a Day," Proceedings of the International Conference on Robotics and Automation (ICRA 2019).

[12] X. Du, Y. Cai, S. Wang and L. Zhang, "Overview of deep learning," 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), Wuhan, 2016, pp. 159-164.

[13] https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

[14] https://keras.io/layers/core/