Modeling Human Car-Following Behavior from Demonstration with Recurrent Neural Networks

Iris Jones
School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA, USA
iris.jones@wsu.edu

Megan Walter
Department of Computer and Information Science
University of Oregon
Eugene, OR, USA
mwalter2@uoregon.edu

Rahul Bhadani
Department of Electrical & Computer Engineering
University of Arizona
Tucson, AZ, USA
rahulbhadani@email.arizona.edu

Jonathan Sprinkle
Department of Electrical & Computer Engineering
University of Arizona
Tucson, AZ, USA
sprinkjm@email.arizona.edu

***Abstract*—The validity of simulation testing for autonomous vehicles depends on the ability to accurately simulate human driving behavior. This project seeks to train a model on an individual's driving data, and to test the ability of the model to predict trajectories that replicate the driver's style by using the model in a realistic simulated environment. Specifically, we deployed Recurrent Neural Network (RNN) modeling techniques to create a black-box model of an individual's driving behavior. We use our RNN-trained model to simulate a human-driven vehicle in the Robot Operating System (ROS) based CAT Vehicle simulator for autonomous vehicle validation. We hope this work is a step to improve testing environments for validating human behavior replicating car-following models and thereby improve testing environments for autonomous vehicles in general.**

***Index Terms*—recurrent neural network, car-following models, driving behavior, trajectory prediction, simulation**

I. Introduction

Modeling human driving behavior is necessary to test autonomous vehicles applications in simulation. While there are many proficient drivers, it is difficult for a driver to explain why or how they drive precisely enough to replicate it in a model. This makes replicating human driving behavior with rule-based car-following models difficult, and learning from demonstration with a black box model an encouraging approach, as there is no need to rely on the drivers' ideas of how they drive in order to replicate their driving behavior. To determine that a car-following model is accurate and demonstrates the desired attributes of the human driver, testing in realistic simulations is required.

In this paper, we chose to focus on a simple one-dimensional car-following scenario. We train a simple Recurrent Neural Network (RNN) model on data from one driver in one car in car-following situations, and then test the model in simulation with the Robot Operating System (ROS) [1] based CAT Vehicle simulator [2].

This work was supported by National Science Foundation under award number 1950359 REU Site: CAT Vehicle.

First we introduce RNNs in Section II. Section III reviews current car-following models trained from demonstration. Section IV outlines the problem definition. Section V explains our methodology for cleaning and preparing the data set, the layers and hyperparameters of our model, and the method with which we use the model in ROS. Section VI displays our results on how the model preformed, both with metrics and in simulation. Finally, Section VII presents our conclusions

II. Recurrent Neural Networks

Recurrent neural networks (RNN) are powerful in their ability to address sequential data, making RNNs ideal for working with time series data. The network establishes correlations between each data point and its recent predecessors. Connections between the nodes in the network form directed cycles allowing the network to display temporal dynamic behavior. That is to say, a data point recorded at time t is affected by a data point recorded at time $t-1$ [3]. Long short-term memory (LSTM) RNN frameworks in particular have been effective in topics such as speech recognition [4], automatic music composition [5], and more recently, predicting trajectories of autonomous vehicles [6].

The LSTM architecture was initially introduced by Hochreiter and Schmidhuber in 1997. Conventional RNNs have issues with gradient vanishing and gradient exploding. This creates difficulties when trying to base predictions off of long-term histories. LSTM models are able to avoid these issues [7].

III. Literature review of car-following models

We focus on car-following models that learn their behavior from real world data or human demonstration, as opposed to rule-based models.

A. Recurrent Neural Networks

Recurrent Neural Networks (RNN) have been trained on human driving data to propagate trajectories. In 2017, Morton

[6] used RNNs with Long Short-Term Memory (LSTM) [7] cells to map information in the history about the ego and lead vehicle such as headway distance, speed difference, and the ego car's speed and acceleration, to produce a probability distribution of the ego's acceleration. Ten second trajectories were split into two seconds of data to initialize the model, and then the acceleration was fed back in to determine the ego vehicle's trajectory. They found the RNNs replicated the qualitative behavior of drivers well, but that LSTM relied primarily on more recent information. Aside from car-following models, RNNs have been used to predict human-driven car trajectories at intersections, ranking the most likely solutions [8]. Additionally, RNNs have been shown to characterize driving behavior better than hand-picked feature engineering methods [9] and that LSTMs in particular were successful at identifying drivers by classifying driving maneuvers [10]. Although we are not classifying, our data was collected with a single driver, which gives hope that the RNN would be able to pick up on the particular driver's style.

B. *Inverse Reinforcement Learning*

In 2015, Kuefler [11] used feature based Inverse Reinforcement Learning to learn driving styles from demonstration as represented by a linear cost function. This was a step away from time consuming manual tuning of parameters and towards learning from demonstration, but it still relied on feature selection in order to tell the model what features of the trajectories were desirable to learn about as opposed to the model being able to make those distinctions itself.

C. *Generative Adversarial Imitation Learning*

In 2017, Kuefler [12] expanded on their working with imitation learning with Generative Adversarial Imitation Learning, or GAIL, which attempts to learn a surrogate reward function from the data by utilizing the feedback from a discriminator which seeks to differentiate between the true data and the generated data. This allows GAIL to learn the driving behavior from the data, without the need for hand picked features or domain knowledge. GAIL tended to provide more stable long term trajectories than previous approaches.

Several modifications to GAIL have been proposed to improve its performance in modeling human driving behavior, by adding the capacity to model multi-agent behavior, incorporate some domain knowledge, and capture latent variables that classify driving styles [13]. The simulation used to test their methods sampled real data, choosing one car from the sample to control with the policy. The policy outputs acceleration and turn rate values in response to observed features, creating the car's trajectory. All cars not controlled by the policy followed their set path in the data, which means they were not reacting to the decisions made by the ego vehicle, limiting the simulations ability to reflect the real world and potentially leading to more crashes than would otherwise occur if other drivers could react.

IV. Problem Definition

We want to create car-following trajectories that resemble the driving style of the driver who created our real world data. We want to predict the car's speed at the next time step given the car's speed, acceleration, and distance from the lead vehicle for the last window of time. In the model implementation, we seek to predict Δv, i.e. change in the velocity from previous time-step to current time-step to determine the next commanded velocity by adding predicted Δv to velocity v from the previous time-step, i.e.

$$\begin{aligned}\Delta v(t) &= f_R(\mathbf{v}_{t-k-1,t-1}, \mathbf{d}_{t-k-1,t-1}, \mathbf{a}_{t-k-1,t-1}) \\ v_{cmd}(t) &= \Delta v(t) + v(t-1)\end{aligned} \tag{1}$$

where $\Delta v(t)$ is predicted by our black-box RNN model f_R; $\mathbf{v}$, $\mathbf{d}$, and $\mathbf{a}$ are velocity vector, distance vector and acceleration vector respectively over given time horizon (or history length) k. v_{cmd} is the commanded velocity for the current time-step. Therefore, speed, acceleration, and lead distance for the last window of time are the input to our model, and the change in speed, Δv, is the output. The ROS simulation is to be designed in such a way that the trained RNN model will be fed an initial window of values, and then the prediction will be used to command speed to the simulated robotic vehicle, and acceleration and lead distance will be updated according to the environment and be fed back in, creating the trajectory.

V. Methodology

A. *Dataset and Preparation*

Our data set consists of 68 trips, the majority of trips conducted in speedway conditions by a single driver, and each trip varying in duration. After filtering out the trips that were under 3 minutes (as they tend to have no movement in them), there were 55 trips. A typical remaining trip lasts around 10-20 minutes. The data contains many messages, but we chose to use lead distance (the distance between the car and the object in front of it), speed, and acceleration in the x direction (forward and backward) as our features. The value to predict, Δv, was calculated from the difference in speed. Since the sensors do not all sample at the same instance or rate, we interpolated and re-sampled at a rate of 50 hertz, which results in time steps of 0.02 seconds, where each timestamp has a value for all the feature variables. A time-step of 0.02 seconds is reasonable as that is a close approximation of the average time between sensor readings for the features we used. Lead distance is a discontinuous variable so we only interpolated and re-sampled on continuous sections, and we did not make predictions across discontinuous sections. Discontinuous lead sections where determined by jumps in value over a threshold of 3 meters. When we later decided to only use data where the lead car was present, a time gap over one second was also considered discontinuous. The remaining features were continuous, and when interpolating and re-sampling them, their rate was matched to that of the lead. The interpolation method used was the cubic spline interpolator from the scipy package. In order to insure we did not predict over time gaps in the data, gaps over 0.05 seconds where used to divide the trips into sections, and state action pairs where only created within and not across sections.

A single state action pair consists of a state, which represents the states of the feature variables for the last time window, and an action, the change in speed the car makes at the current time interval. The action, Δv, when added to the last speed in the state history results in the speed observed in the data at the next time step. Δv was chosen as the action rather than speed because it was easier to use to command the speed in the simulation. We also converted the data to be in the same units as the simulation uses, specifically we transformed speed into meter per second from kilometers per hour to avoid conversions when using the model. We chose a history length of 100 points, resulting in a time window of two seconds, consistent with the length of history found useful in [6]. Figures 1, 2 and 3 show the history of an example state graphed for each feature, and the red point on Figure 3 represents the action, Δv, as the speed after adding it to the previous speed. These graphs represent the histories before scaling. All data in the state action pairs is scaled with the min max scaling formula. For lead, the maximum value used for scaling was the maximum value still in range, as the lead sensor records 252 meters when there is no lead vehicle in front and the true range is around 110 meters. All the states are stored in one array, and the matching actions are stored in a parallel array. The state array dimensions are (number_of_points, history_length, n_features) and the matching action array has dimension (number_of_points,). See Figure 4 for an example of a single state in array format. Note time is not an explicit variable in our data, but rather implicitly implied in the histories of each state as each point in the state is 0.02 seconds apart. After all the state action pairs were created, test data was taken in multiple groupings, sampled across various sections of the data, totaling 15% of the data. Another 15% of the data was randomly selected and reserved from the remaining state action pairs to serve as validation data during training.

With each iteration of our model, we decided to filter the data to use for training further. Model A's training data has no additional filters, and is considered the baseline. Model B's data was filtered for lead present, meaning only data where the lead vehicle was in sensor range was included. Model C's data was filtered for speed nonzero, meaning only data where the ego car was moving was included. Model D's data was filtered for both lead present and speed nonzero. As can be seen in Table I, these filters significantly reduced the number of state action pairs.

B. Model Layers

Our model is a LSTM Recurrent Neural Network. We chose to use LSTM because it can learn temporal features with shorter histories than a traditional feed-forward network and it showed promise in [6]. Our model has two LSTM layers with Relu activation and 64 and 32 neurons respectively,

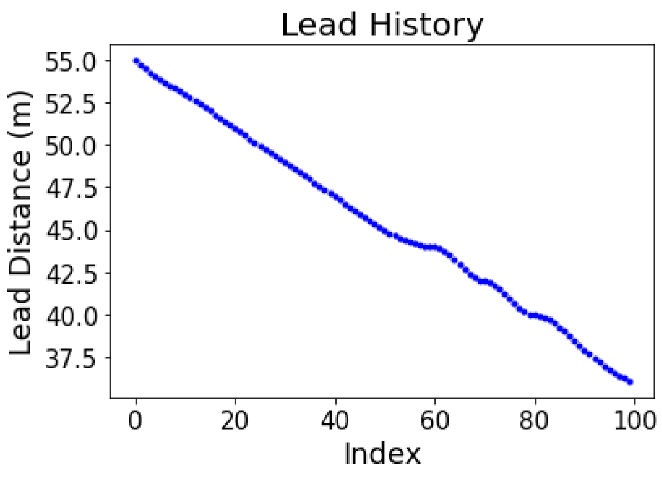


Fig. 1. Example state lead distance history.

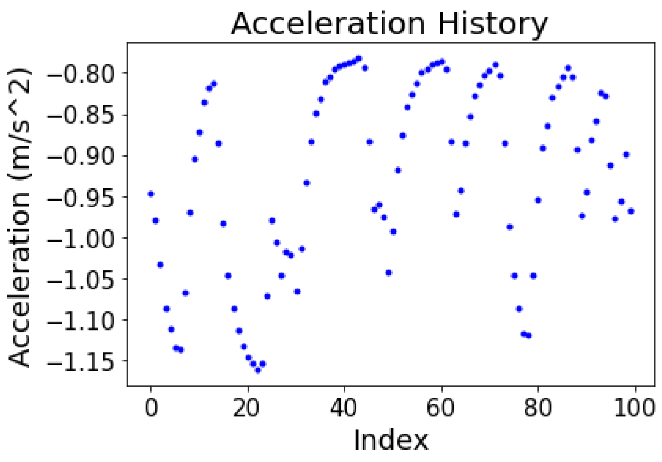


Fig. 2. Example state acceleration history.

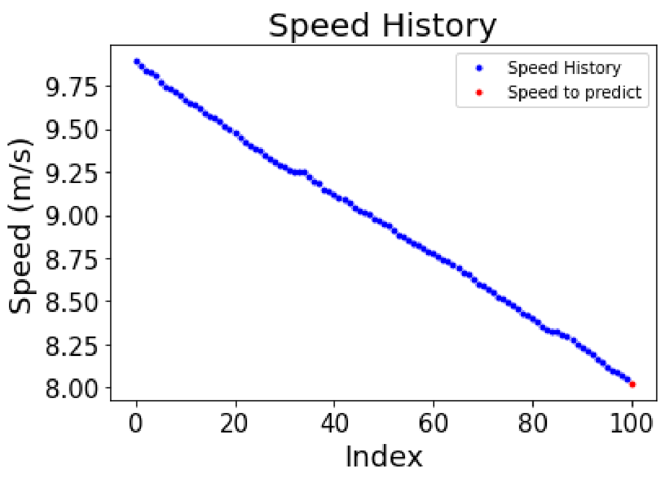
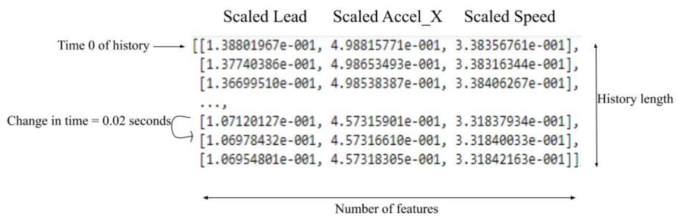


Fig. 3. Example state speed history and action.



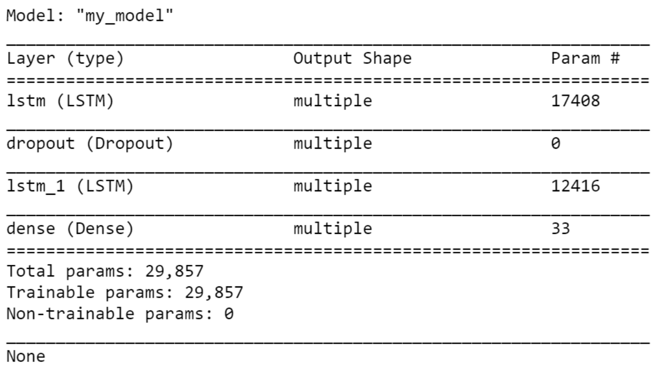
```
[[1.38801967e-001, 4.98815771e-001, 3.38356761e-001],
 [1.37740386e-001, 4.98653493e-001, 3.38316344e-001],
 [1.36699510e-001, 4.98538387e-001, 3.38406267e-001],
 ...,
 [1.07120127e-001, 4.57315901e-001, 3.31837934e-001],
 [1.06978432e-001, 4.57316610e-001, 3.31840033e-001],
 [1.06954801e-001, 4.57318305e-001, 3.31842163e-001]]
```

Fig. 4. Example of a single state.

TABLE I
DATA USED BY EACH MODEL

Model	Description	No. State Action Pairs
A	Baseline	2665657
B	Lead present	1181875
C	Speed nonzero	1686755
D	lead present and speed nonzero	749246

separated by a dropout layer with a dropout rate of 0.10 for regularization. Early tests showed that two layers of LSTM outperformed a single layer with more neurons. The final layer is a Dense layer with one neuron in order to produce the single value prediction. See Figure 5 for the model summary provided by Tensorflow.



```
Model: "my_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  multiple                  17408
_________________________________________________________________
dropout (Dropout)            multiple                  0
_________________________________________________________________
lstm_1 (LSTM)                multiple                  12416
_________________________________________________________________
dense (Dense)                multiple                  33
=================================================================
Total params: 29,857
Trainable params: 29,857
Non-trainable params: 0
_________________________________________________________________
None
```

Fig. 5. Model Summary.

C. Hyperparameters

Parameters were initially chosen based on success of past models and then tuned based on what was working best on our trial runs with small data sets and limited features. We used the standard Adam optimizer provided by Tensorflow with default values (learning rate is 0.001). A callback function was used such that training was cut short if validation loss did not improve after 3 epochs. We saw marginal improvement if any after 10 epochs, and therefore stopped training at 10 epochs (if the callback didn't stop it sooner).

We tried both mean squared error (MSE) and Huber as the loss function. Both loss functions were reasonably good at ignoring outliers. In our initial tests, Huber looked to do a better job of replicating the distribution of Δv. Our data for Δv is centered at zero, and Huber did a better job of centering near zero than MSE on our initial tests. However, on the larger data set with all our features, MSE and Huber both seem to produce results off center by similar distances, just in opposite directions. Huber tends to skew a little negative, whereas MSE tends to skew positive (although it did skew negative sometimes in our initial tests). As we will see later, if there are too many negative Δv predictions, the car can start to drive backwards which is not desirable or a characteristic exhibited in the training data at all. Even though Huber tends to skew negative, it was closer in absolute value to the zero centered distribution, so we used Huber as the loss function for the models presented in this paper as A-D.

The Huber loss function is given by

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2, & |y - f(x)| \le \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, & otherwise \end{cases} \tag{2}$$

where the $error = y - f(x)$ is the difference between the truth and the prediction. The Huber loss function is quadratic for small values of error, and linear for large values of error, as distinguished by δ [14]. We used the default Tensorflow value for δ of 1.0. The scaling factor of $\frac{1}{2}$ is not included in [14], however it is in the original definition [15] and in the definition used by Tensorflow [16].

D. Transfer Learning

To test our model in a different environment, we used ROS to create simulations. Specifically, we used ROS Melodic with rosversion 1.14.6 as well as its Gazebo package which visualized our tests. Additionally, we were able to take realistic vehicle dynamics into account by using the CAT Vehicle simulator.

Our simulation consisted of an ego car, which we controlled through our model in a ROS node, and a lead vehicle. We obtained the lead distance and velocity in real time through a set of subscribers in the ROS node that recorded the values of the command velocity and LIDAR sensor of the ego car. This data, along with the ego car's acceleration calculated from the velocity, acted as the inputs on which the model made its predictions.

The lead vehicle's velocity was determined by a trip that the data-collection car had taken. We recreated that trip's velocity by publishing the relevant speed messages to the lead vehicle. In addition, we also ran separate tests for each of our models with reduced lead vehicle velocity so that the lead vehicle would remain in range of the ego car for a longer period and allow us to further observe the effects of lead distance on the models' predictions.

As our model outputted scaled values, we used the min max formula with the provided min and max values of the training data to unscale the output of the model back into the correct scale. Those unscaled predictions were then added to the ego car's current velocity and published through a publisher in the ROS node to the ego vehicle as the command speed, and then were fed back into the model as an input (after scaling again). Figure 6 shows a visualization of the simulations set up in

Gazebo and Figure 7 shows the flow of data in the simulation.

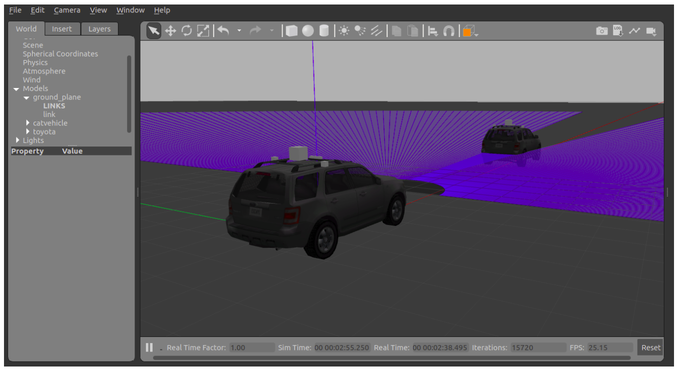


Fig. 6. Ego car with laser sensor and lead vehicle in Gazebo.

VI. Results

A. Model results from training

Our initial model, Model A, appeared to have a good loss as seen in Table II, however it didn't seem to capture any car-following behavior in simulation, as will be shown in the next subsection. In order to capture more of the car-following behavior, we though it would improve the model to train only on data where the lead car was actually present, because you cannot exhibit car following behavior when there is no car in front of you, which we did for Model B. The trade off here is that the model will only be applicable when a car is in sensor range, and will have unknown behavior if the lead car leaves sensor range. As is shown in Table. II, this model had the best test loss. (Training and validation loss are measured on the last epoch in Table II).

TABLE II
Model Loss Evaluation

Model	loss		
	Train	***Validation***	***Test***
A	$2.81e^{-7}$	$9.12e^{-7}$	$2.85e^{-7}$
B	$2.70e^{-6}$	$1.33e^{-6}$	$1.98e^{-7}$
C	$5.18e^{-7}$	$1.89e^{-7}$	$7.82e^{-7}$
D	$7.90e^{-7}$	$2.44e^{-6}$	$5.94e^{-7}$

The graphs of a very small portion of the test data can lend some insight for each model. Figures 8 through 11 graph some of the predictions made on the test data against the true actions in the test data by model. The red dots and blue dots at the same x-tick are the prediction and true speed respectively for the same history. Since the test data was taken in small sections, the blue dots before each red dot are also the history (or portion of the history) that was used to make those predictions. We will refer back to these graphs as we talk more about each model's performance.

We can see in Figure 8, the predictions made by Model A tend to be slightly lower than the target value, although this is exaggerated visually by the tight range of the y-axis scale. Figure 9 shows Model B's predictions are closer to the truth, although the scale here plays apart in that visual. (Note that this graph spans two test sections, the true history does not ask the model to predict over that jump.) Ultimately, as we will see in the next subsection, Model B still did not exhibit the car-following behavior we were after.

We restricted the data further by ignoring any history that has a speed of zero in it. We speculated this would help because when a car is stopped at a stop light or parked for the beginning of the recording, the car's actions are not correlated to the lead vehicle's distance. It is possible we would be excluding data where the car was stopped because of the car in front of it as well, but the majority of nonzero data seemed to be from the beginning of trips before the car had started driving. Model C trained on data restricted to speed nonzero only, while Model D trained on data restricted for both lead present and speed nonzero. Training the Models C and D on these reduced data sets lead to a larger loss, see Table. II. However, this is reasonable because such a large portion of the data contained zero speeds, and predicting no change in speed when the car has not been moving is relatively easy. Looking at the graphs of a potion of the test predictions in Figure 10 and Figure 11 doesn't display any noticeable difference in the quality of predictions from each other, even from the previous Model B. Ultimately, the simulation does the best job of determining whether or not the model displays car following behavior. As we will discuss, despite Models C and D seeming similar when looking at these graphs and at the test loss, they behaved quite differently in simulation.

B. Model results in simulation

As reducing the speed of the lead vehicle to one sixth of its original speed allowed for us to take a closer look at the effect of the lead vehicle on the ego car, all the results discussed in this section will be referring to tests that used this reduced lead vehicle speed.

From our tests with the lead vehicle's velocity reduced to one sixth of the original trip velocity, our baseline Model A predicted primarily small negative values resulting in an overall small and negative velocity (Figure 12). There was a slight correlation between the lead distance and the ego car's command speed, as the lead distance increased, the command speed did as well. However, the velocity of the ego car remained between -0.25 m/s and 0 m/s meaning that the change in velocity was negligible when compared to the speed of the lead vehicle which stayed between 0 m/s and 2.5 m/s. Visually, this was represented by the ego car rolling backwards at an extremely slow pace. In addition, after the lead vehicle had moved out of range of the ego car's laser sensor, the command speed converged at around -0.16 m/s and continued that way for 50 seconds until the end of the simulation.

After adjusting the model so that it only trained on data when the lead vehicle was present (Model B), we found that, similar to Model A, the predictions led to a small, negative velocity. Model B performed slightly better in terms of the magnitude of the velocity in that the velocity remained

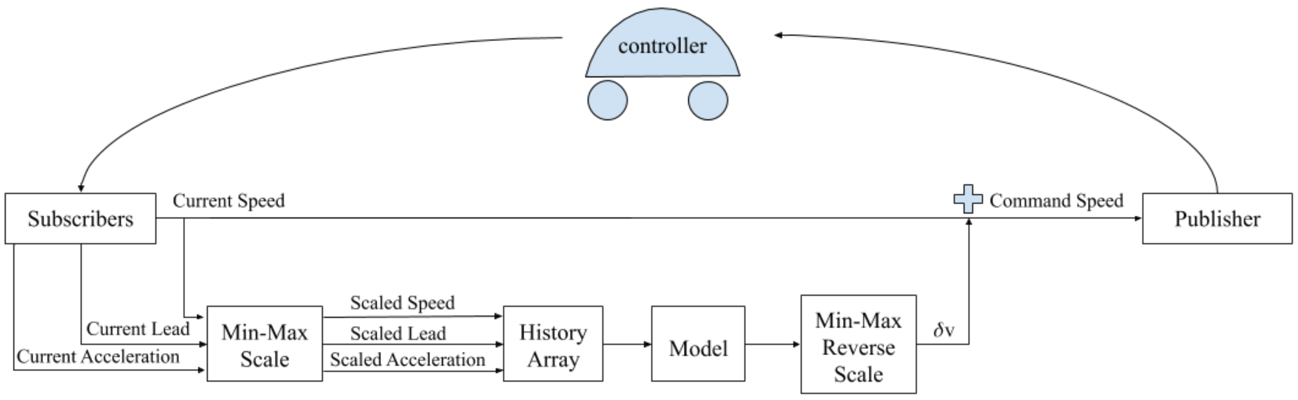


Fig. 7. Data flow of model in the simulation.

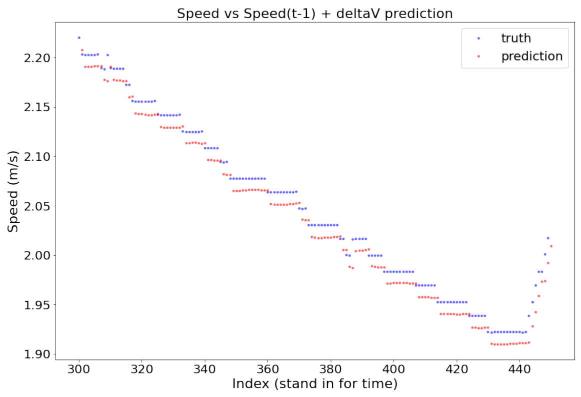


Fig. 8. Model A, sample of predictions on test set

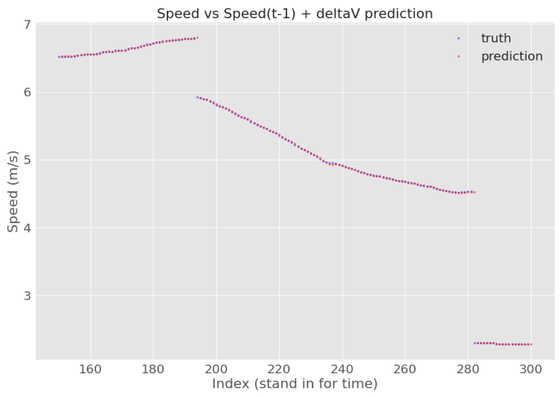


Fig. 10. Model C, sample of predictions on test set

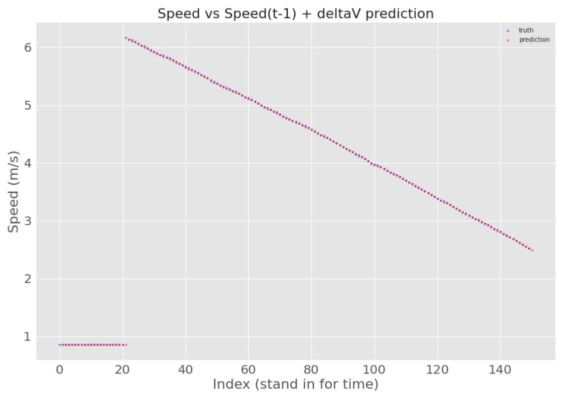


Fig. 9. Model B, sample of predictions on test set

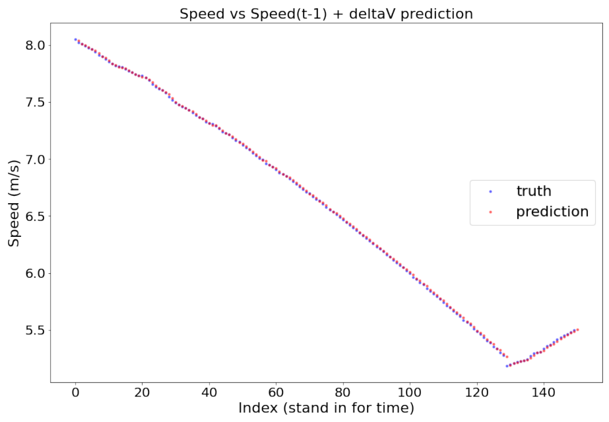


Fig. 11. Model D, sample of predictions on test set

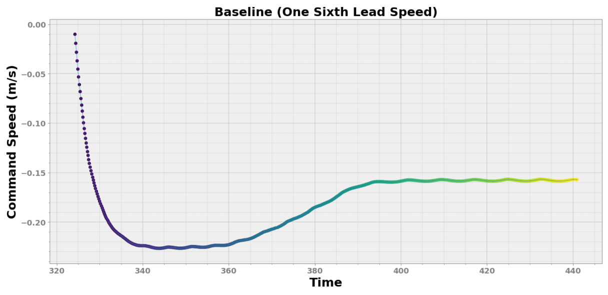


Fig. 12. Model A, command speed predicted by model and published to ego car. Lead speed divided by six.

between 0 m/s and -0.85 m/s. We found that as the lead vehicle drove further away, the velocity of the ego car grew larger and more negative. Again, once the lead vehicle drove far enough that the ego car could no longer sense it, the velocity converged around -0.8 m/s.

The car controlled by the Model C, the model trained with only non-zero values for speed, on the other hand, remained positive throughout the entire simulation. The behavior of Model C very much resembled that of the baseline model with its positive values being the exception. The ego car's velocity remained between 0 and 0.25 m/s with the velocity having a slight correlation to the lead distance. Just as the previous models, once the ego car could no longer sense the lead vehicle, the commanded velocity converged around 0.22 m/s and remained that way for 50 seconds, until the end of the simulation.

Finally, we tested our Model D, which trained on data that always had a nonzero velocity and a lead vehicle present. In general, this model performed far better than the others and was able to follow the lead vehicle for 5 minutes. The velocity of the ego car remained between -0.2 m/s and 2.5 m/s which nearly matched the range of the lead vehicle's velocity between 0 m/s and 2.6 m/s. The connection between the command velocity shown in Figure 13, and the lead distance shown in Figure 14 becomes clear as the command speed closely follows the same pattern as the lead distance, albeit scaled down. These results strongly indicate that the model put a lot of weight on lead distance as a factor in predicting the next velocity. Furthermore, before the lead vehicle began to move, the velocity published to the ego car was negative, but as the lead vehicle began to drive, the ego car began to exhibit car-following behavior. A possible reason for this could be that the lead vehicle started too close to the ego car and the model attempted to maintain an ideal distance. It is also important to note that though the ego car was able to follow the lead car at one-sixth speed, when the lead vehicle went faster (full speed, half speed, third speed), the ego car was not able to catch up in time to exhibit the same results as one-sixth speed.

The majority of our models converged on unrealistically small values in simulation despite showing promising results after evaluating them with test data. This may be due to compounding error. During the validation process, the history

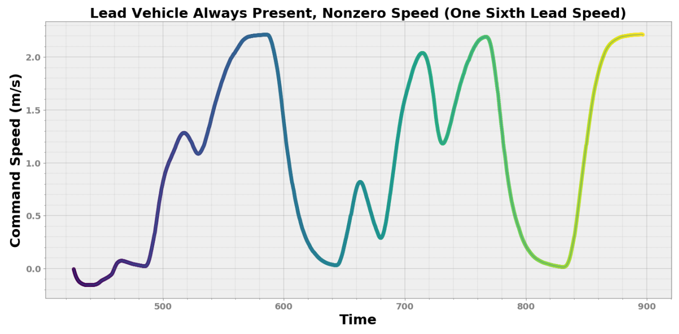


Fig. 13. Model D, command speed predicted by model and published to ego car. Lead speed divided by six.

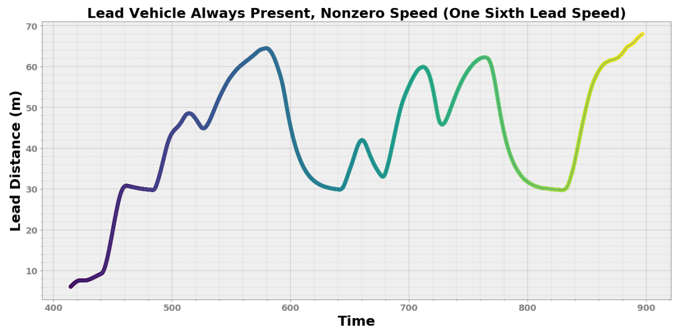


Fig. 14. Model D. Lead distance recorded by ego car. Lead speed divided by six.

that the model took into account was from an actual trip that the car took so if a prediction was too low or too high, it would not directly affect the next prediction. The lack of correction may have been the reason the results of the simulations did not reflect the loss values found during the model loss evaluation.

Another discrepancy between the training data and the simulation is that the maximum range of the sensor for lead distance in the training data was around 110 meters, while it was 81 meters for the sensor. Therefore, when the lead car goes out of range in the simulation, the ego car was trained to believe that that the lead car is still within range.

For Model D, though the model had comparatively better results, it still was only able to follow the lead vehicle at low speeds that would not be helpful in recreating human-driving behavior. It is possible that additional data to train on would increase the efficacy of the model.

VII. Conclusion

In this paper we trained an RNN car-following model and demonstrated the insights ROS can provide for testing car-following models. We showed that narrowing the data to instances where the lead vehicle was in range and the car was moving improved the model's ability to react to the behavior of the lead vehicle.

While we did show some car-following behavior in our last model, the car did not reach high speeds and was not able to stay in range of the lead vehicle for very long. Until the model is able to more fully replicate the car-following behavior in a general sense, it is not worth trying to compare the style of how the model follows to how the original driver follows. The model needs to exhibit good control of car following behavior in general before it warrants a comparison on the specifics of style.

Ideas for future work on this model include fine tuning of parameters and adding more features, such as relative speed, that convey information about the lead vehicle or turning angle that would expand it from a one-dimensional to a two-dimensional car-following model. Using more data and pulling histories that do not overlap with each other might also improve the model's performance. Further testing can be done with the simulation where the lead vehicle has a variety of behaviors to see how the car follows, and more model types can be tested such as GAIL. Generative Adversarial Networks may be better at replicating the distribution than our RNN model, which struggled to produce a Δv distribution centered at zero like the original data. The model could also be used on a drive with a real car, predicting but not commanding speed, in order to assess its performance on a real drive in real time.

Acknowledgment

The authors thank Co-Principle Investigator Dr. Ditzler. Jones and Walter thank fellow participants of the Cognitive and Autonomous Test Vehicle Research Experience for Undergraduates for their camaraderie.

References

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[2] R. Bhadani, J. Sprinkle, and M. Bunting, "The CAT Vehicle Testbed: A Simulator with Hardware in the Loop for Autonomous Vehicle Applications," in *Proceedings 2nd International Workshop on Safe Control of Autonomous Vehicles (SCAV 2018), Porto, Portugal, 10th April 2018, Electronic Proceedings in Theoretical Computer Science 269*, vol. 269, pp. 32–47, 2018.

[3] T. I. Poznyak, I. Chairez Oria, and A. S. Poznyak, "Chapter3 - background on dynamic neural networks," in *Ozonation and Biodegradation in Environmental Engineering* (T. I. Poznyak, I. Chairez Oria, and A. S. Poznyak, eds.), pp. 57 – 74, Elsevier, 2019.

[4] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 1764–1772, PMLR, 22–24 Jun 2014.

[5] K. Choi, G. Fazekas, and M. B. Sandler, "Text-based LSTM networks for automatic music composition," *CoRR*, vol. abs/1604.05358, 2016.

[6] J. Morton, T. A. Wheeler, and M. J. Kochenderfer, "Analysis of recurrent neural networks for probabilistic modeling of driver behavior," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1289–1298, 2017.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] A. Zyner, S. Worrall, and E. Nebot, "Naturalistic driver intention and path prediction using recurrent neural networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 4, pp. 1584–1594, 2020.

[9] W. Dong, J. Li, R. Yao, C. Li, T. Yuan, and L. Wang, "Characterizing driving styles with deep learning," *CoRR*, vol. abs/1607.03611, 2016.

[10] Y. Wang and I. W. Ho, "Joint deep neural network modelling and statistical analysis on characterizing driving behaviors," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1–6, 2018.

[11] M. Kuderer, S. Gulati, and W. Burgard, "Learning driving styles for autonomous vehicles from demonstration," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2641–2646, 2015.

[12] A. Kuefler, J. Morton, T. Wheeler, and M. Kochenderfer, "Imitating driver behavior with generative adversarial networks," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 204–211, 2017.

[13] R. Bhattacharyya, B. Wulfe, D. Phillips, A. Kuefler, J. Morton, R. Senanayake, and M. Kochenderfer, "Modeling human driving behavior through generative adversarial imitation learning," 2020.

[14] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning Data Mining, Inference, and Prediction, Second Edition*. Springer New York, 2009.

[15] P. J. Huber, "Robust estimation of a location parameter," *Ann. Math. Statist.*, vol. 35, pp. 73–101, 03 1964.

[16] "tf.keras.losses.huber." https://www.tensorflow.org/api_docs/python/tf/keras/losses/Huber. Accessed: 2020-08-11.