AutoEnv Project Report

Zhihao Zhang
advised by Prof. Changliu Liu

***Abstract*—Due to the requirement for robust self-driving algorithms and the risk for testing vehicles in real world scenario, autonomous driving simulation has been in the highest demand in recent years. Meanwhile, multiple of autonomous driving algorithms have emerged[1], which include trajectory prediction, vehicle control and planning algorithms. As the research field is flourishing rapidly, we don't have a code base integrating all these together. To this end, we have built a compact platform named AutoEnv in Python which gives us an integrated pipeline from algorithm implementation, simulation to evaluation. Most of the algorithms are implemented in PyTorch[2], but we have TensorFlow[3] support as well. Both of them are wonderful deep learning libraries. In this paper, we will briefly introduce key features and components of this platform and further evaluate several algorithms on trajectory prediction using this platform. AutoEnv's source code can be found at https://github.com/intelligent-control-lab/Autoenv.**

***Keywords* – Autonomous driving, Simulator, Trajectory prediction**

I. Introduction

We notice that it is not practical to test the performance of autonomous driving algorithms in the real world scenario, since on-road testing is not able to satisfy safety and efficiency requirement. On the other hand, simulation is a much better solution. Meanwhile, multiple of autonomous driving algorithms have emerged[1], which include loads of trajectory, vehicle control and planning algorithms. As autonomous driving algorithms are flourishing rapidly, we surely need a code base to integrate all these together. To this end, we aim to build a compact platform named AutoEnv in Python which gives us an integrated pipeline from algorithm implementation, simulation to evaluation. Most of the algorithms are implemented in PyTorch[2], but we have support for TensorFlow[3] as well, both of them are wonderful deep learning libraries. In this paper, we briefly introduce key features and components of this platform and further evaluate several algorithms on trajectory prediction.

We want our vehicle to be intelligent enough to do self driving tasks. Meanwhile, we shouldn't sacrifice the safety guarantees to achieve this. To balance this, our algorithm should use as much as information it could receive to do robust control of our ego vehicle. It is a common case where many vehicles will do simultaneous control in the same environment. In order to satisfy the safety constraint in such scenario, it is important for our ego vehicle to predict trajectories of other vehicles around. Hence, our primary goal for building this environment is to test different trajectory prediction models. This work is built based on a previous Julia version Autonomous Driving Simulator[4]. I will mainly introduce our work in a chronological order for both simulator and algorithms we used.

II. Simulator

A. Data processor

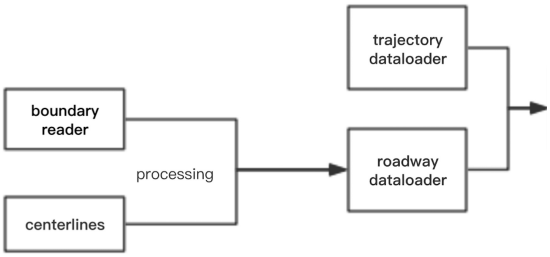


Fig. 1: Flow chart for data processor, our data processor has two branch, one is for lane processing and one for trajectory processing

lane processing, we have two raw csv files writing boundary and center line information respectively. We have thousands of curvature point identify the reference point along the road collected by a vehicle using the same frequency, but we need to do a further smoothing process to interpolate these points in order to form our integrated lane information

trajectory processing, we have a csv file including all the information related to vehicles, each entity is identified by time stamp and its vehicle id with one entity per row. Attributes could include global location, lane id, vehicle width and etc. We down sampled these vehicles by a 10 Hz frequency, followed by integrating the trajectory information with lane information to form our final version data.

B. Feature extractor and interactive Interface

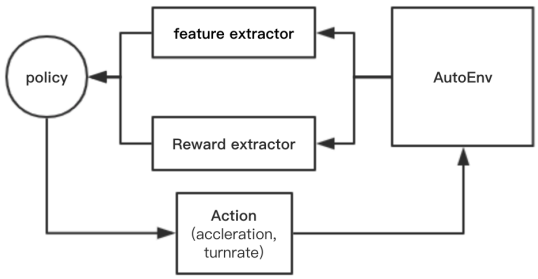


Fig. 2: Flow chart for feature extractor and interactive interface, AutoEnv module represent our interface interact with our agent, who is controlled by a policy.

feature extractor, after generating all the informative trajectory data we want, we need to use several feature extractors to pull all the features we want out of those data. More specifically, we have a hierarchical extractor structure with one top level extractor and several sub extractors. Namely, car lidar feature extractor who is in charge of providing the distance to other nearby vehicles measured by radar ranging, core feature extractor who provides some basic information about our vehicle and the lane(eg. car width, length, lane curvature), fore feature extractor providing the relative information with respect to the front vehicle(eg. relative speed, relative acceleration rate), temporal feature extractor providing some first order or second order derivatives of ego vehicle's velocity and angular velocity, well behaved feature extractor to provide some behavioral feature(eg. out of the lane, is colliding or not). For each time step our interactive interface will extract those features from previously processed data and feed it to our agent.

interactive interface, for the interface, our simulator follows a close loop interaction structure. At each time step, our environment(AutoEnv) will first extract all the features and send them to our agent with the reward for that state. Our agent has a control module, which is represented by a policy, whose input is the feature of that state and output is the action. The generated action will then be fed back to the interface. The interface will then update its state according to the action and provide our agent with features and reward for next time step. Given a policy, we can thus acquire a roll-out trajectory as our predicted trajectory through the interaction between our agent and the environment.

III. Methodology

Our model mainly consists of two part, an offline training part and an online adaption part. Our main contribution is proposing the online adaption algorithm to achieve a more accurate prediction. Previous state-of-the-art method make the assumption that each individual vehicle's policy model share the same weights, which might not be satisfied in the real world scenario. We present a more realistic model which take the difference of individual vehicle's policy into account. On the other hand, we don't want to train our model from scratch. To achieve an efficient and intuitive model, we combine offline training with online adaption. More specifically, we use the parameters trained in the offline phase as the initial parameters for our policy in online adaption. As the vehicles are interacting with the real world, our online adaption algorithm will adjust each vehicles' policy according to their history trajectories and actions.

For time step t in the online phase we can denote our policy as π_t. The way we do prediction at time step t is to use our policy π_t at that time step to roll-out a trajectory whose horizon is 50 steps, notice that the prediction sample frequency is 10Hz across all steps.

A. Offline Policy Training

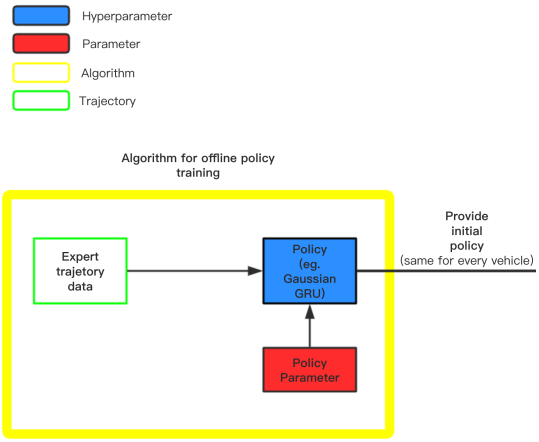


Fig. 3: Offline policy training phase, aim at providing a shared initial policy for each individual vehicle

For offline training part, we should train a policy in a multi-agent scenario. In the case of lacking expert data, we can define a reward function and use state-of-the-art multi-agent reinforcement learning algorithms(eg. MA-DDPG) to optimize our policy. If we could get access to some expert data, then we could further incorporate some imitation learning algorithms(eg. PS-GAIL) to help us learn the policy. For experiments, we choose PS-GAIL[4] as our offline training algorithm since we could get access to NGSIM highway expert driving data[5] and do imitation learning accordingly.

B. Online Adaption

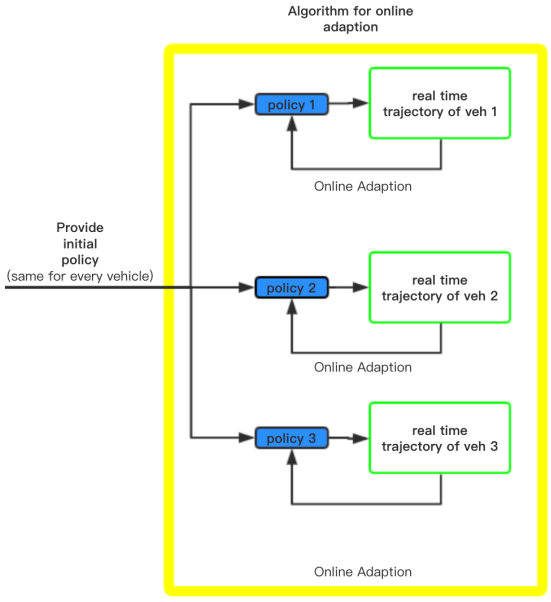


Fig. 4: Online policy adaption phase, our model could adapt each vehicle's policy according to their history trajectory

During the process of online adaption, our vehicles will first receive the same policy trained in the offline phase as their starting policy, then each vehicles will use recursive least square (RLS)[6] online adaption to adjust their own policy through the test time. As shown in Fig. 4, each vehicle is doing real time interactions with the world. Our objective is shown as following

$$\overline{W}(k,i) = \arg\min_{W} \sum_{t=1}^{k} \lambda^{k-t} \left\| u_i(t) - W g(y_i(t)) \right\|^2 \quad (1)$$

where $\lambda \in (0,1]$ is a discount factor, i is the index of the target vehicle, W is the last fully connective layer in the policy network and g is the function that consist of the rest of other layers in the policy network which can be highly none-linear. $u_i(t)$ stands for the ground truth action target vehicle takes at time step t and $y_i(t)$ denotes the observation of target vehicle at time step t. k is the current time stamp and $\overline{W}$ is our optimized result. We can obtain $\overline{W}(k,i)$ recursively as below.

$$\overline{W}(k+1,i) = \overline{W}(k,i) + e_i(k+1) g(y_i(k))^T F_i(k+1) \quad (2)$$

where $F_i(k)$ is the learning gain and $e_i(k+1)$ is the prediction error. For detailed definition please refer to [?]. When doing prediction at time step t, the last layer of our policy π_t is then tuned as $\overline{W}(t)$ which incorporates more about the personal driving habits instead of a sharing policy. Then we will do a 50 steps roll-out prediction using the same $\overline{W}(t)$ since prediction at time step t means we are not interacting with real world and couldn't evaluate the error in time step $t+1$.

IV. Result

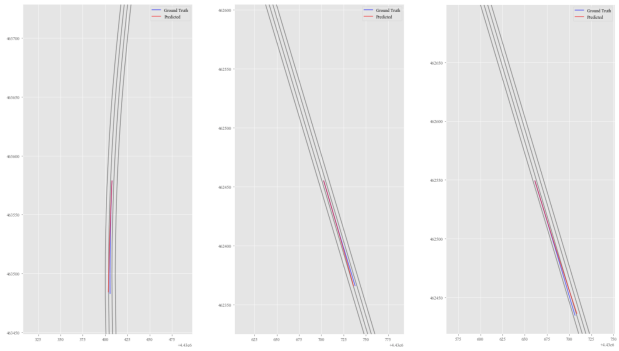


Fig. 5: Visualization of AGen prediction algorithm in test time, blue line is the ground truth trajectory while red line is our predicted trajectory, prediction span is 50 steps

We compared our algorithm with a state-of-the-art multi-agent imitation learning algorithm PS-GAIL. More specifically, we test the performance of four cases which are 1) PS-GAIL with RLS online adaption, 2) PS-GAIL without online adaption, 3) fine-tuned PS-GAIL with RLS online adaption, 4) fine-tuned PS-GAIL without online adaption. These experiments are tested using Holomatic highway driving data set. The training set consists of ∼200,000 trajectories whose average length is ∼500 while the test set consists of ∼15,000 trajectories whose average length is ∼500 as well. We use several statistical metrics to evaluate and compare the performances for all the cases.

A. Evaluation Metrics

The trajectory denoted by a sequence of two dimensional location points predicted by the model will be compared with the ground truth value in the expert driving data. Given two location point $v_t^{(i)}$ and $\overline{v}_t^{(i)}$, where the former stands for the ground truth value of time step $t+i$ and latter denotes the predicted value of $t+i$ at time step t. We can calculate the root mean square error for the prediction as following:

$$error(t,i) = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (v_t^{(i)} - \overline{v}_t^{(i)})^2} \quad (3)$$

B. Overall root mean square error along prediction span

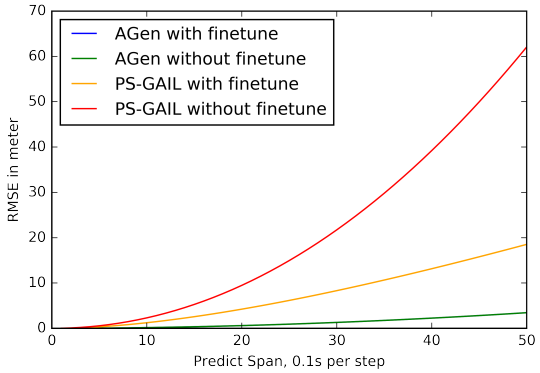


Fig. 6: Overall RMSE loss for all four cases

Fig.6 shows the overall root mean square error for all four cases. We draw this graph using $\frac{1}{T}\sum_{i=1}^{T} error(t,i)$ for prediction step i. It is quite intuitive that PS-GAIL without fine-tune has the worst performance, fine-tuning policy parameters on Holomatic training data has a better performance. When AGen is used, the prediction error rapidly decreases. When AGen is combined with fine-tuned policy, there isn't an obvious improvement in the overall RMSE, as AGen is overfitting the local policy of the target vehicle and would adjust the policy to a local optimal in one step. Therefore, AGen is actually compensating for the inadequate training of the feature network.

	Overall	Curve	Lane-changing	Straight
PS-GAIL	20.960075	21.085213	21.766454	20.895329
fine-tuned PS-GAIL	7.335238	6.843423	5.615716	7.583835
AGen	**1.223645**	**1.495403**	4.364190	**1.077413**
fine-tuned AGen	1.224736	1.498173	**4.360599**	1.077699

TABLE I: Categorical RMSE over all prediction span, unit: meter

C. Root mean square error pyramid for AGen with fine-tune

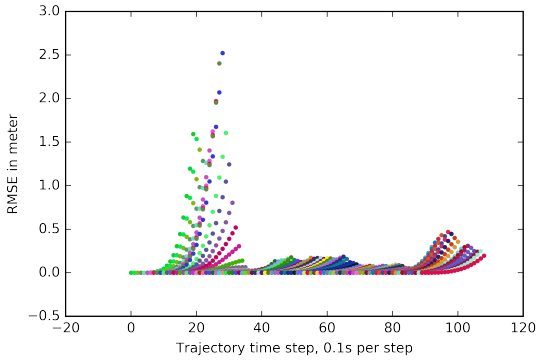


Fig. 7: This graph shows the root mean square error of the prediction over 110 steps using AGen with fine-tuned feature network, for each step, we do a 50 steps trajectory roll-out prediction and calculate its RMSE with regard to the ground truth value accordingly

Adaption mainly let our prediction model adjust or overfit to vehicle's local driving policy in order to make more accurate prediction for that vehicle at next time step. So the RMSE loss will decrease as the adaption step goes. There are some waves shown in this graph which could be interpreted as some small changes in drivers policy or behavior.

D. Categorical root mean square error

We classify each of our ground truth trajectory into three category, namely straight trajectory, curve trajectory and lane-changing trajectory. For each algorithm, we have drawn the mean/std graph as shown in Fig. 8. It is quite intuitive that straight trajectories always have a lower RMSE which is followed by curve category while lane-changing has the worst performance. This is not the case for Fig. 8(d). We find that PS-GAIL with fine-tuned feature network tends to predict turning behavior since it has a highest straight RMSE. Feature network fine-tune has not changed much of the AGen performance on all three categories which also indicates the ability of AGen to locally overfit, this can be seen from TABLE 1 as well.

E. M Stability

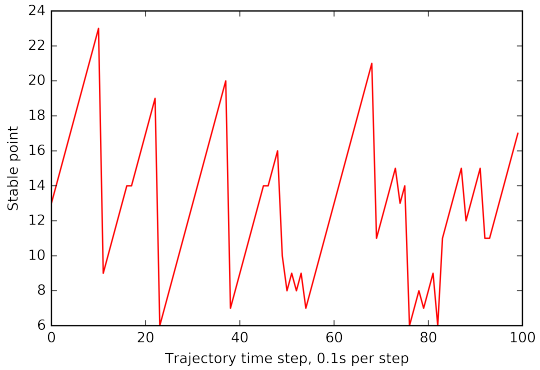


Fig. 9: This graph shows the M stability for 100 steps trajectory prediction

Why linear increasing? The error of predicted trajectories have variance, if one of the predicted trajectory is better than most of other neighbor trajectory, than there will be an increasing. As showed in Fig. 10, circle represents predicted trajectory steps, blue circle represents common trajectories and green circle indicates better predicted trajectories. Blue arrow indicates normal behavior, where neighboring predictions should be accurate while green arrow indicates abnormal predictions. Therefore, we can see M in the fourth step is 1 and in the fifth step is 2 as the blue arrow showed, which follows a linear increasing trend.

Why sudden dropping? Whenever there is a better predicted trajectory(trajectory in green) appeared in this error matrix, the trajectory predicted in the next time step will have a worse RMSE value compared to the current one. Therefore, value M will head back to a small value again, which corresponding to a sudden drop in the graph.

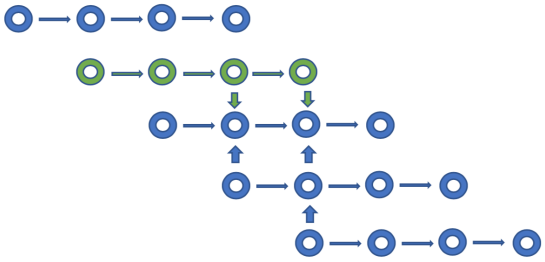
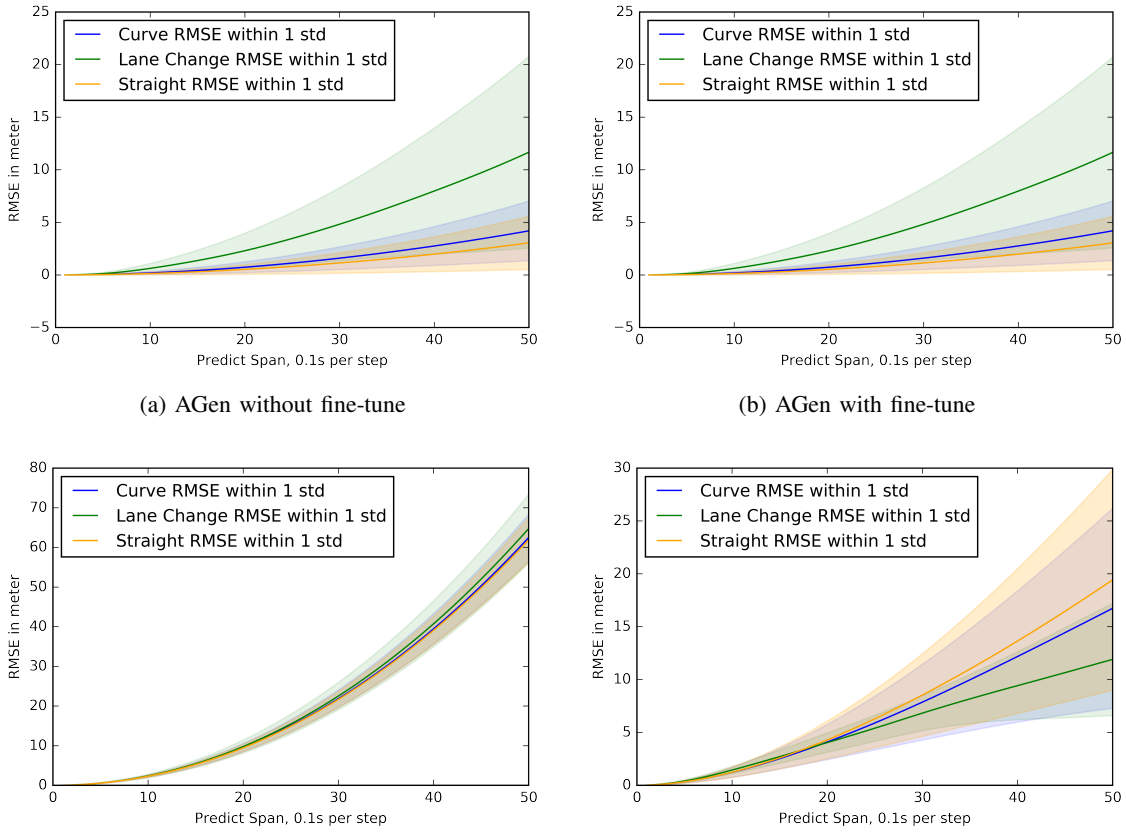


Fig. 10: Schematic diagram to explain the jagged shape of M-stability result

V. Future Work

A. Metrics design

How to define a metric for autonomous driving task is still ambiguous in literature. Usually, our ground truth trajectory is a sequence of deterministic points, however the ground truth trajectory may not be the only reasonable prediction given information of the current state. To this end, we might want to



(a) AGen without fine-tune

(b) AGen with fine-tune

(c) PS-GAIL without fine-tune

(d) PS-GAIL with fine-tune

Fig. 8: Categorical root mean square error along prediction span, we classify our trajectory based on the curvature of ground truth trajectory reference points. Namely, straight trajectory, curve trajectory and lane changing trajectory

use a probabilistic prediction model instead of a deterministic one. Then the problem is how to properly evaluate a distribution against a deterministic ground truth value. This problem surely need further consideration.

B. Label bias

For the training and testing set, if we look at the portion of each categorical trajectories, we will find ∼80% of all the ground truth trajectories are straight line trajectory, which can be a bit misleading with regard to the prediction error. Since constant velocity model could achieve a good prediction score which is not what we want. Therefore, in future experiments, we might need to balance the number of each categorical trajectory to correct this label bias phenomenon.

C. Being flexible

We want our platform to be flexible enough to test all kinds of algorithms. For feature extraction part, we provide several interfaces which can be modified independently. What you need to do is to simply add a python file writing a sub extractor class and register it with the higher level extractor. You can define any feature you want given the raw information of the trajectory data. For algorithm part, you can implement your own algorithm to do control or prediction. Currently, the simulator could only accept control input which can be used in generative prediction model as well. For the metrics part, like what we said in section A, it is hard to design a persuasive evaluation metric. We only have support for root mean square error for now, but will test and add more in the future.

APPENDIX A
CODE STRUCTURE

```
./Autoenv
├── adaption.py
├── algorithms
├── doc
├── envs
├── feature_extractor
├── preprocessing
└── src
```

Fig. 11: The tree structure of our code base

algorithms: We put all model related file in this directory, e.g. AGen, GAIL and TRPO[7]. In the future, you can put all your algorithms in this folder.

envs: This directory contains all the environment interface classes, which include single agent environment and multi-agent environment. Of course you can design your own

environment and put the file in this folder.

feature extractor: We put all kinds of feature extractor classes here which have been mentioned before. You can add whatever feature you want that can be extracted from raw data here.

preprocessing: Currently we have put all data preprocessing files here, but these are related to what data format you are using.

src: This folder contains all source code dealing with trajectory integration and smoothing.

doc: Contains the manual instruction of AutoEnv.

adaption.py: the main file to run for AGen

ACKNOWLEDGMENT

The author would like to thank his advisor Prof. Changliu Liu and all the lab members in Intelligent Control Lab for their suggestions and help. Sincerely enjoying the research life in CMU with such a group of talented people and a great advisor.

REFERENCES

[1] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[2] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Pytorch," *Computer software. Vers. 0.3*, vol. 1, 2017.

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[4] R. P. Bhattacharyya, D. J. Phillips, B. Wulfe, J. Morton, A. Kuefler, and M. J. Kochenderfer, "Multi-agent imitation learning for driving simulation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 1534–1539.

[5] J. Colyar and J. Halkias, "Us highway 101 dataset," in *Tech. Rep. FHWAHRT-07-030*, Jan. 2007.

[6] M. H. Hayes, "9.4: Recursive least squares," in *Statistical Digital Signal Processing and Modeling*, 1996, p. 541, ISBN: 0-471-59431-8.

[7] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.