Marrying Motion Forecasting and Offline Model-Based Reinforcement Learning for Self-Driving Cars

Swapnil Pande and Xindi Wu
Robotics Institute
Carnegie Mellon University United States
{swapnilp, xindiw}@andrew.cmu.edu

Abstract:
Many of the current state-of-the-art reinforcement learning (RL) algorithms for self-driving cars require online training, which cannot take advantage of the millions of miles of logged driving data available. Model-based offline RL(MBRL) has the promise of learning effective driving behaviors from this data. However, the simple MLP models that have traditionally been used in offline MBRL fail to perform well for the self-driving task, partially due to the difficulty of forecasting the complex behaviors of the other dynamic actors in the environment. To perform offline MBRL for self-driving cars, we propose to use an uncertainty-aware dynamics model that decouples the dynamics for the ego-vehicle and other vehicles, and apply prior work in motion forecasting to do other-vehicle prediction. We show that this algorithm is capable of learning simple driving behavior and demonstrate its shortcomings for more complex driving scenarios.

Keywords: Offline Reinforcement Learning, Motion Forecasting, Autonomous Driving

1 Introduction

Over the past decade, there has been immense interest in both industry and academia for developing algorithms for autonomously driving vehicles. However, developing an algorithm that requires no human intervention has proven to be quite difficult. Many of the successful approaches use a hand-crafted set of modules, each performing a subtask of the driving problem, such as localization, mapping, localization, and control [1]. These approaches require extensive manual tuning of parameters to move closer to the goal of fully autonomous driving.

To circumvent the need for tuning each module, there are many recent works that propose imitation learning as a viable solution to map directly from sensor inputs to driving commands. However, even these systems become difficult to scale due to the need for the extensive expert demonstrations of the rare scenarios that a car might encounter. Reinforcement learning has the potential to alleviate this data issue, as the policy is no longer directly optimized to mimic the expert demonstrations. Additionally, the formulation of RL as a sequential decision making process is better suited for the task of driving compared to treating each time step as independent as imitation learning does.

However, a common problem with reinforcement learning is that it cannot be trained online in the real world due to its high sample complexity. Deploying a random policy to a car would be too expensive, both in terms of time and in potential accidents. As a result, most learning approaches focus on learning in a simulator and later transferring their policies to the real world. While this may be an effective solution, it cannot take advantage of the million miles of driving logs that self-driving car companies have available. These logs contain many useful scenarios and examples of how to react in certain scenarios. Due to the availability of this data, we believe the self-driving car challenge can be posed as an offline reinforcement learning challenge, in which we attempt to train a policy using only an offline dataset of interactions with the environment. Furthermore, we focus on the idea of model-based offline reinforcement learning, in which we learn a dynamics model of

the environment to generate rollouts to optimize a policy. This allows us to "imagine" new scenarios that did not exist in the original dataset.

This approach poses an additional challenge: building a model of such a high-dimensional and cluttered environment is very difficult. In the self-driving car environment, there are multiple actors all interacting with each other and the static objects. Modeling the dynamics of all of these agents in the observation space of the policy would likely result in a poor model as it needs to predict the behavior of the other agents as well as how our observation frame changes. For example, it would be very difficult to predict the position of another car in a front-facing image taken from our car as both cars are moving independently. We propose to simplify this problem by decomposing the model of the environment into a model of the ego-vehicle dynamics and a model of all of the other actors in the environment. The model of the ego-vehicle dynamics simply predicts the vehicle's evolution given the actions generated by the policy. To model the motion of all of the other vehicles, we can apply the recent advances in the literature on motion forecasting. This decomposition allows us to independently reason about our vehicle and other vehicles, and combine the predictions to generate an observation for the policy. Additionally, we can also more explicitly represent the process of "imagining" new trajectories for the other vehicles, allowing us to generate additional training data not already in the dataset.

In this work, we propose a method for performing offline model-based reinforcement learning for self-driving cars. Particularly, we propose a novel model architecture that decomposes the dynamics into the dynamics of the ego-vehicle and the dynamics of other vehicles. To model the dynamics of other vehicles, we apply recent techniques from motion forecasting. We believe that this model architecture will better allow us to reason about the behavior of other actors, so that we can learn policies that effectively and safely interact with other vehicles on the road. We do not focus on the task of learning from raw sensor inputs such as lidar or RGB images. Instead, we assume that, in addition to raw sensor data, we are given access to a low-dimensional representation of the state space, which can be computed from raw sensor inputs.

2 Related Works

2.1 Learning for Self-Driving Cars

There are many examples of papers that apply imitation learning to learn from sensor inputs using datasets of expert demonstrations [2, 3, 4, 5]. Since pure imitation learning cannot learn to correct mistakes, these methods propose various methods to augment the imitation learning process. For example, [4] trains a priveleged expert from expert demonstrations and then trains a student imitation network that accepts the raw camera input. While training the student, the expert is also queried for all possible driving commands at every state to augment the imitation dataset. [5] augments the imitation learning dataset by adding perturbations to driving trajectories that demonstrate how to recover if the vehicle begins to drift out of lane. While these methods have shown success, the performance of an imitation learning agent is capped to that of the expert demonstrations. Additionally, these methods are difficult to scale as they require expert demonstrations of all of the scenarios the vehicle may encounter.

Recently with the successes of deep reinforcement learning (DRL) for a wide variety of planning and control tasks, DRL, reinforcement learning methods have begun to surpass the performance of imitation agents in standard self-driving benchmarks. Particularly, [6] demonstrates strong performance on the standard driving benchmarks, using a policy trained online using Proximal Policy Optimization (PPO) [7]. In this method, they consider a hand-crafted low-dimensional state space that contains the necessary features for learning effective driving behaviors, and also demonstrate success learning directly from semantically segmented images. Formulating the problem as a sequential decision making process is better suited to self-driving compared to treating each time-step as i.i.d. as imitation learning does. We adopt a slightly modified version of the low-dimensional state-space presented in this work and apply it to learning in the offline setting.

We use the CARLA simulator [8] to generate data and evaluate our policy, which has been used extensively in literature for self-driving cars. Additionally, we benchmark our algorithm using the CARLA benchmark they propose.

2.2 Offline Reinforcement Learning

There has been growing interest in developing and improving reinforcement learning algorithms for the offline setting. A more comprehensive survey of current techniques can be found at [9]. One group of methods for Offline RL focus on improving the stability of off-policy Q-learning by reducing the overestimation of the Q-function in regions out of the data support [10, 11, 12]. For example, [12] performs well by including a constraint that discourages the Q-function from predicting higher values for out-of-distribution states compared to in-distribution states. While these methods have proven to be successful, we focus on model-based offline RL because of its potential to "dream" of rare scenarios not present in the original dataset, such as near collisions with other vehicles and unpredictable pedestrians.

Another class of methods focus on performing Offline RL in a model-based setting with a learning procedure similar to the one presented in [13]. At a high level, these methods first optimize a model $f(s_t, a_t)$ to predict the transition dynamics of the environment. They then train a policy by performing autoregressive rollouts with actions sampled from the policy being optimized. However, similar to the Q-learning algorithms, the model often poorly extrapolates in regions of the state space outside of the data distribution. Therefore, recent works such as [14, 15] present uncertainty-aware dynamics models and introduce a penalty in the reward function conditioned on the state estimation uncertainty during the policy optimization. We focus on applying an algorithm similar to those presented in [14, 15], in which the policy is penalized proportionally to the magnitude of the model's uncertainty in the given state. However, we improve upon their dynamics model structure, by building a model that encodes inductive biases about the structure of the self-driving car problem.

2.3 Motion Forecasting

Motion forecasting has been increasingly essential for the self-driving car field [16, 17], recent progress has shown promising trajectory prediction results given sensor data. [18] provides a survey of traditional approaches. Several benchmarks and datasets are widely used including Argoverse [16], nuScenes [19] and Lyft Prediction [20]. Some methods rasterize the scene for agents as RGB Birds-Eye-View(BEV) images with actor trajectories overlaid onto the image [21, 22], and use different channels for observation timesteps. Some learning-based methods applied long short-term memory units to generate lane-changing trajectories [23], or help process the multi-agent interactions [24]. [25] proposed the combination of LSTM encoder-decoder and the attention mechanism to predict the lane changing intention and the future trajectories.

VectorNet [26] uses a RNN to encode the map which is regarded as a collection of polylines and incorporate vectorized map prior and agent dynamics for motion forecasting. LaneGCN [27] builds a graph of lanes and conducts hierarchical graph convolutions over the vectorized map data to understand the complex topology and actor-map interactions. PRECOG [21] proposed a multi-agent generative motion forecasting method, it aims to capture the future stochasticity in the actor's goals conditioned on the position of all other actors. In our work, we follow the motion forecasting concept used in [21] and combine it with the offline RL setting to learn a driving policy for the agent.

3 Methods

3.1 Problem Formulation

We consider the task of self-driving as a Partially-Observable Markov Decision process (POMDP) defined as the tuple $(S, O, A, r, P, \rho_0, \gamma)$, where S is the state space, O is the observation space, $r : S \times A$ is the reward function, $P : S \times A \times S$ is the state transition probability, ρ_0 is the initial state distribution, and γ is the discount factor. Our objective is to find a policy $\pi(a_t|o_t)$ such that:

$$\pi = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] \tag{1}$$

The offline RL problem extends this setting by assuming we are given a dataset $\mathcal{D} = \{s_i, o_i, a_i, r_i, s_{i+1}\}_{i=1}^{N}$, generated by a set of data generation policies π_D. We assume we have

no knowledge about these data collection policies. Our goal is to find the policy in Equation 1 using only the experience in dataset $\mathcal{D}$.

A full diagram of model can be found in Figure 2. We propose a model-based learning approach, in which we first learn an uncertainty aware model of the transition dynamics $P(s_{t+1}|s_t, a_t)$ based on the offline dataset $\mathcal{D}$. Using these transition dynamics, we then optimize the policy π by performing 5-step rollouts with actions sampled from the policy. We penalize the policy for entering states with high uncertainty in a manner similar to MOPO, presented in [15].

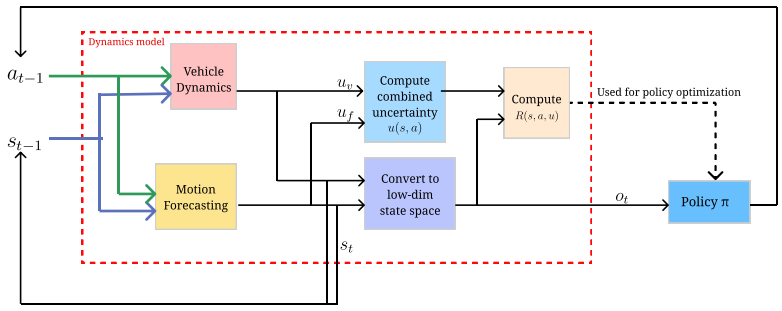

Figure 1: Overview of the proposed structure of our model

3.2 Self Driving Car Environment

We formulate the self-driving car problem as follows: our goal is to travel from a source waypoint to a destination waypoint, while avoiding infractions such as colliding with other vehicles, pedestrians, and static objects as well as following driving rules such as staying within lanes. We assume that we have knowledge of the route to follow, described as a series of dense waypoints (4.5 meters apart) between the source and destination waypoints. Additionally, action inputs are mapped through PID controllers to smooth the motion of the vehicle. This problem formulation is inspired from the work presented in [6].

3.3 Dynamics Model

The objective of the dynamics model is to model the state transition probability $P(s_{t+1}|s_t, a_t)$. A naïve approach to selecting the representation of S would be making it the same as observation space for the vehicle O. However, this is a difficult representation to learn, as the feature evolution in the observation is a function of both our vehicle dynamics, as well as the intentions of the other dynamic actors in the environment. For example, if the observation space was a forward-facing image, we would have to predict the transformation of the camera frame due to our action, as well as the relative transformation of the other vehicles due to their actions. This representation would be difficult to learn and couples the uncertainty in the motion of our vehicle with the motion of the other vehicle.

Instead we select a state representation that naturally decouples into the dynamics of our vehicle and the dynamics of all of the other nearby vehicles. Specifically, S contains the coordinates (relative to a map frame) of the other vehicles and detailed features about our vehicle's state. With this representation, the evolution of the other vehicles can be predicted using prior work in motion forecasting, while the evolution of our vehicle can be predicted using a separate dynamics model. Once these are separately computed, they can be combined to reconstruct an observation for the policy. This simplifies the prediction problem and incorporates prior work in motion prediction to improve our modelling performance.

3.3.1 Ego Vehicle Dynamics

The dynamics of the ego vehicle are described by four key features. The first two features are the current steer angle ($\tilde{s}$) and the current speed ($\tilde{v}$) of the vehicle. The next two features are related to the waypoints. The distance from trajectory ($\tilde{n}$) describes the perpendicular distance between the vehicle and the trajectory defined by the waypoints. Finally, we add an orientation error ($\tilde{w}$), which

denotes the angular error between the vehicle's current heading and the next waypoint. A visualization of the distance to trajectory and orientation error can be found in Figure 4 in the appendix. To predict these waypoint features, the dynamics model additionally requires information about the position of the waypoints. Therefore, we also include the cartesian coordinates of the next 3 waypoints in the vehicle frame: $\tilde{\mathbf{e}}_\mathbf{t} = \{(x_{t,w1}, y_{t,w1}), (x_{t,w2}, y_{t,w2}), (x_{t,w3}, y_{t,w3})\}$. It is important to note that the dynamics model cannot predict these since they are generated by a high-level planner and are not a function of the state. Therefore, we assume these are given from the dataset. Finally, we stack a history of the previous two states as the input. In total, the input space for the vehicle dynamics is $(\tilde{n}_{t-1}, \tilde{n}_t, \tilde{w}_{t-1}, \tilde{w}_t, \tilde{s}_{t-1}, \tilde{s}_t, \tilde{v}_{t-1}, \tilde{v}_t, \tilde{\mathbf{e}}_{t-1}, \tilde{\mathbf{e}}_t)$ and the output space is $(\tilde{n}_{t+1}, \tilde{w}_{t+1}, \tilde{s}_{t+1}, \tilde{v}_{t+1})$.

We parametrize the dynamics model as an ensemble of multi-layer perceptrons. We opt for an MLP, instead of a simplified model such as a bicycle model, as an MLP can capture higher order dynamics that a bicycle model may not be able to represent. At each time step, we sample the output of one of members of the ensemble as the next state prediction. Additionally, we estimate the uncertainty in the dynamics prediction as proportional to the maximum discrepancy in state prediction between members of the ensemble, as proposed in [15]. Specifically, we define the uncertainty as follows:

$$u_v(s, a) = \max_{i,j} \|f_i(s, a) - f_j(s, a)\|_2 \tag{2}$$

where i and j index the members of the ensemble.

The dynamics models are trained by minimizing the Huber loss between the 1-step predictions and the ground-truth in the dataset. We train each member in the ensemble with a different shuffling of the dataset.

3.3.2 Motion Forecasting for Other Vehicle Prediction

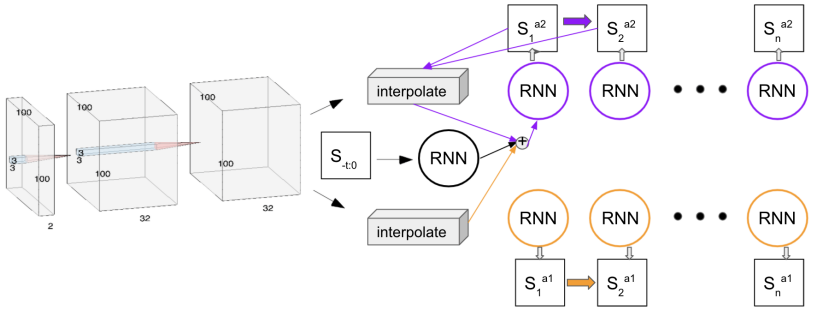

Figure 2: Overview of the proposed structure of the motion forecasting model. Ego-vehicle LIDAR features are passed through a CNN, and interpolated into the frame for each agent. The history of states for all of the agents are fed through RNNs and combined with LIDAR features to generate predictions for the next states for each agent.

To predict the behavior of other vehicles, we apply motion forecasting on vehicles near the ego vehicle to predict their future trajectories. For the vehicles of interest $\{V_1, \ldots, V_M\}$, given the location information of each vehicle V_i, where $X_{V_i} = (x_i^t, y_i^t)$ for time steps $t = \{1, \ldots, obs\}$, we aim to predict the future coordinates $Y_{V_i} = (x_i^t, y_i^t)$ for T steps $t = \{obs + 1, \ldots, obs + T\}$. A diagram of the model is presented in Figure 2, which is based on the approach proposed proposed in [21]. We use a probabilistic generative model to model the multi-agent system with A agents. Each agent's state is described by the environment perception information $\phi \doteq \{\mathbf{S}_{-obs:0}, \boldsymbol{\chi}\}$, where obs denotes the number of time steps of position history and $\boldsymbol{\chi} \in \mathbb{R}^{100\times100\times2}$ represents LIDAR observation generated from the ego-vehicle. It is important to note that S includes the position of all nearby vehicles.

Given this state ϕ, our model $S_{t+1} \sim q(\cdot|S_{t-obs:t}, \phi)$ generates predictions of next state for each vehicle. In order to handle uncertainty in the goals of each agent, the model predicts a mean μ and standard deviation σ for a Gaussian over next states. We sample from this Gaussian by sampling a latent $\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$, and mapping that through the function $S_t^a = f(\mathbf{Z}; \mu, \sigma) = S_{t-1}^a + (S_{t-1}^a - S_{t-2}^a) + \mu + \sigma * \mathbf{Z}$. As an application of the reparametrization trick (referred to in [21] as a

pushforward function), this is differentiable. We optimize the following joint distribution under the dataset.

$$q(S|\phi) = \prod_{t=1}^{T} q(S_{t+1}|S_{t-obs:t}, \phi). \tag{3}$$

Finally, we quantify the uncertainty in the forecasting prediction at time t as follows:

$$u_f(s) = \sum_{a=1}^{A} \sigma_t^a(s) \tag{4}$$

where A is the number of vehicles we are modeling.

3.3.3 Combining Features

We combine the features predicted by the vehicle dynamics and the motion forecasting to create the observation for the policy. The observation contains all of the features predicted by the vehicle dynamics model $(n_{t+1}, w_{t+1}, s_{t+1}, v_{t+1})$. Additionally, we include another feature $\tilde{o}$ that represents the distance of the nearest vehicle in front of our vehicle, which is calculated from the predicted position of vehicles from the motion forecasting model. We set $\tilde{o} = 1$ if the distance of the vehicle in front is greater than or equal to a threshold distance.

3.4 Policy Optimization

The policy π is parametrized by a multi-layer perceptron. The action space for the policy is the target velocity and target steer angle for the PID controller. The reward function is defined as follows:

$$\mathcal{R} = \alpha * \tilde{v} - \beta * \tilde{n} - I(s) - \eta * u_v(s, a) - \omega * u_f(s, a) \tag{5}$$

where $\tilde{v}$ is the velocity of the ego vehicle, $\tilde{n}$ is the distance to trajectory feature, $I(s)$ is an infraction penalty, and u_v and u_f are the uncertainty of the state, action pair. We consider two infractions in the infraction penalty: a collision with another vehicle (distance between vehicles below threshold) and an off-route penalty (distance to trajectory greater than threshold). α, β, η, and ω are constant hyperparameters representing the weight of each term in the reward function.

To optimize the policy using the dynamics model, we initialize the dynamics model to a state from the dataset and perform 5-step rollouts with actions sampled from the policy. The policy is optimized using proximal policy optimization [7].

4 Experimental Results

4.1 Dataset

We generate a dataset of driving trajectories in Town01 in Carla 9.10. 50 other vehicles are also intialized in the environment. The data generating policy is a predefined CARLA autopilot policy, that uses heuristics about the state to generate driving policies. We additionally add Gaussian noise to the autopilot policy to improve the performance of the dynamics model. The dataset contains a total of 60,000 environment interactions.

4.2 Benchmarks

We evaluate the performance of our policy using a slightly modified version of the original CARLA benchmark [8]. The CARLA benchmark has 4 challenge levels, varying from driving straight to navigating through a town with other dynamic agents. In this benchmark, the episode only ended when the agent reached the goal waypoint. We simplify this benchmark to include three difficulty levels. The first (straight) requires the agent to follow a straight route with no other vehicles in the environment. Next, the second level (turn), requires the agent to follow a route containing a turn. Finally, the third level (dynamic straight), requires the agent to follow a straight route with other vehicles in the environment.

We compare our policy against Learning by Cheating [4], a state of the art imitation learning algorithm. For the "Straight" and "One Turn" benchmarks, we train a single policy for both benchmarks. Additionally, we do not include the motion forecasting model for these as there are no other vehicles in the environment. We train a separate policy for the "Dynamic Straight" benchmark, with the motion forecasting model included.

4.3 Ego Vehicle Dynamics Learning

We first evaluate the performance of our vehicle dynamics model on a hold-out validation dataset, as shown in Figure 5 in the appendix. We perform an ablation on the features in the dynamics model. Specifically, we compare the performance of the full state space to a state space with the history of the $t-1$ step removed and another with the waypoint positions removed. The results show that these features are critical for effectively learning the dynamics of the vehicle. The stack of observations allows the model to estimate higher order derivatives in the vehicle dynamics. The waypoint positions help to improve the prediction of the next orientation and distance to trajectory features. The next orientation and distance to trajectory are a function of the position of the waypoints as well as the position of the vehicle. If the waypoints are static, the model should not need information about the position of the waypoint to predict how the feature will evolve as it is then only a function of the vehicle position. However, during a time step, the vehicle may "cross" a waypoint, causing that waypoint to be removed from the trajectory. As a result, the model would no longer know the position of the next waypoint, leading to erroneous predictions. In these cases, the knowledge of the position of the waypoint is important to correctly predict the distance to trajectory and waypoint error.

4.4 Motion Forecasting

We visualize examples of the motion forecasting predictions in Figure 3. Additional images as well as performance metrics are presented in Figure 6 and Table 2 in the appendix. The figures demonstrate that the model accurately captures the true trajectory within the distribution of predicted trajectories. However, we also see in 3b that some of the trajectories are still infeasible, where the vehicle is predicted to drive into an obstacle. This means that we may not be able to trust the model predictions over long horizons. The model does perform well for 5-step rollouts, which is the rollout length used for policy optimization.

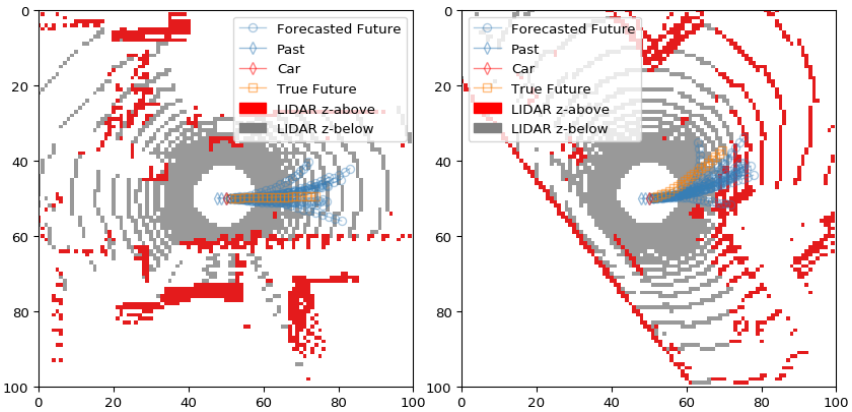

Figure 3: Image of distribution of possible vehicle trajectories (blue) compared to ground-truth (orange). (a) contains a straight driving trajectory and (b) contains a turning trajectory.

4.5 Policy Learning

The results of the policy training are presented in Table 1. Additionally, videos of the driving behaviors are available at the link below[1]. We find that the policy is able to perform well on the

[1] https://drive.google.com/drive/folders/1KT87gr5IOGO5CBE4VnXqW0XifFBFFWFr?usp=sharing

Task	LBC [4]	Ours
Straight	**25**	24
One Turn	**25**	4
Dynamic Straight	**N/A**	1

Table 1: Performance of our policy evaluated in Town01 on the CARLA benchmark (Success Episodes out of 25)

Straight benchmark, achieving a nearly perfect score. However, the car does not perform well on the One Turn and Dynamic Straight benchmarks.

In the One Turn benchmark, we find that the car's velocity entering turns is too high, causing the car to enter the oncoming traffic lane as it takes the turn. As it enters the opposing lane, the car actually stops completely, causing the episode to time out. In the case of the Dynamic Straight benchmark, we find that the car does not stop behind vehicles, instead continuing forward and colliding with them.

We believe that both of these failure cases are a result of the short episode rollouts, which cause poor value function estimation. In the turning case, for example, the policy is likely learning to accelerate into a turn as this locally maximizes rewards due to the high velocity. Normally in the online setting, the value function estimate at the terminal state would correct this behavior, as policy rollouts would demonstrate that these poor states result in low future reward through the turn. However, in the offline setting, we start policy rollouts only from states present in the dataset, meaning we rarely encounter these poor turn states resulting in a poor value estimate in these states. Similarly, in the case of the Dynamic Straight benchmark, the policy is always initialized at a safe distance away from vehicles in front. In the 5-step rollout, the policy likely never encounters a collision state. Therefore, the policy learns to maximizes it's reward by maintaining its velocity as it drives toward the car, without considering the resulting poor long-term reward.

These issues may be fixed by better tuning the penalization weights for the uncertainty estimates, as the dynamics models should have higher uncertainty in states not present in the dataset. Additionally, we can increase the rollout length to collect rewards on a longer horizon, which may require improving the dynamics model to reduce the rate of error growth in the predictions. Finally, we can switch to using a Q-learning policy optimization method, such as Soft Actor-Critic [28] or Conservative Q-Learning [12], which would allow us to include expert trajectories from the dataset in our policy optimization as well.

We also find that the policy sometimes learns to brake in cases where the car is drifting off the road or into the wrong lane, a behavior that is not present in the dataset. This demonstrates the benefit of a reinforcement learning approach over an imitation learning approach, namely that we can learn safe behaviors according to a reward function, allowing us to perform better than the data collection policies. Figures 7,8 in the appendix show a comparison of the actions taken by the learned policy and the actions in the dataset. Overall, we see that the policy's steer angle is similar to those in the dataset, but the target speed is very different.

5 Conclusion

In this work, we propose a method for performing offline model-based reinforcement learning for self-driving cars. We build a dynamics model that decouples ego-vehicle dynamics from other vehicle dynamics, for which we apply prior works in motion forecasting. We demonstrate that the policy is able to learn basic lane-following behaviors, including some safe behaviors not present in the dataset. However, the policy fails to learn more complex behaviors such as turning and collision avoidance, likely due to the short policy training rollouts. In the future, we plan to evaluate and improve the prediction accuracy of our dynamics models to perform longer policy rollouts. Additionally, we would like to explore using Q-learning based policy optimization methods that allow us to incorporate expert trajectories in the policy optimization dataset. Finally, we hope to explore the performance of our dynamics model for learning effective behaviors in scenarios with complex interactions with other vehicles, such as lane merging or 4-way stop intersections.

References

[1] M. Campbell, M. Egerstedt, J. How, and R. Murray. Autonomous driving in urban environments: Approaches, lessons and challenges. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 368:4649–72, 10 2010. doi:10.1098/rsta.2010.0110.

[2] A. Bewley, J. Rigley, Y. Liu, J. Hawke, R. Shen, V.-D. Lam, and A. Kendall. Learning to drive from simulation without real world labels. Dec. 2018.

[3] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy. End-to-end driving via conditional imitation learning. Oct. 2017.

[4] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl. Learning by cheating. Dec. 2019.

[5] M. Bansal, A. Krizhevsky, and A. Ogale. ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst. Dec. 2018.

[6] T. Agarwal. On-policy reinforcement learning for learning to drive in urban settings. Master's thesis, Pittsburgh, PA, August 2020.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. July 2017.

[8] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. Nov. 2017.

[9] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. May 2020.

[10] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. Sept. 2015.

[11] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods, 2018.

[12] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative Q-Learning for offline reinforcement learning. June 2020.

[13] M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-Based policy optimization. June 2019.

[14] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims. MOReL : Model-Based offline reinforcement learning. May 2020.

[15] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. MOPO: Model-based offline policy optimization. May 2020.

[16] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8748–8757, 2019.

[17] W. Luo, B. Yang, and R. Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 3569–3577, 2018.

[18] S. Lefèvre, D. Vasquez, and C. Laugier. A survey on motion prediction and risk assessment for intelligent vehicles. *ROBOMECH journal*, 1(1):1–14, 2014.

[19] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[20] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska. One thousand and one hours: Self-driving motion prediction dataset. https://level5.lyft.com/dataset/, 2020.

[21] N. Rhinehart, R. McAllister, K. Kitani, and S. Levine. Precog: Prediction conditioned on goals in visual multi-agent settings. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2821–2830, 2019.

[22] T. Phan-Minh, E. C. Grigore, F. A. Boulton, O. Beijbom, and E. M. Wolff. Covernet: Multimodal behavior prediction using trajectory sets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14074–14083, 2020.

[23] D.-F. Xie, Z.-Z. Fang, B. Jia, and Z. He. A data-driven lane-changing model based on deep learning. *Transportation research part C: emerging technologies*, 106:41–60, 2019.

[24] L. Hou, L. Xin, S. E. Li, B. Cheng, and W. Wang. Interactive trajectory prediction of surrounding road users for autonomous driving using structural-lstm network. *IEEE Transactions on Intelligent Transportation Systems*, 21(11):4615–4625, 2019.

[25] J. Zhu, S. Qin, W. Wang, and D. Zhao. Probabilistic trajectory prediction for autonomous vehicles with attentive recurrent neural process. *arXiv preprint arXiv:1910.08102*, 2019.

[26] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid. Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11525–11533, 2020.

[27] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun. Learning lane graph representations for motion forecasting. In *European Conference on Computer Vision*, pages 541–556. Springer, 2020.

[28] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

6 Appendix

6.1 Description of Low-Dimensional Features

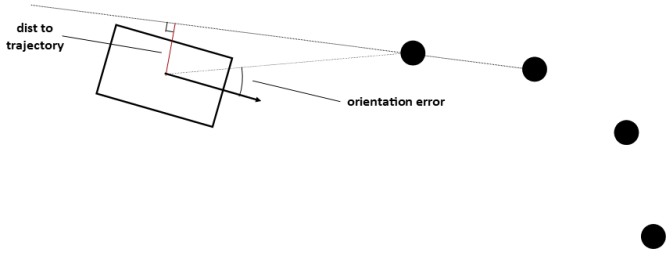

Figure 4: Visualization of Distance to Trajectory and Next Orientation Features

6.2 Ego-Vehicle Dynamics Ablation Results

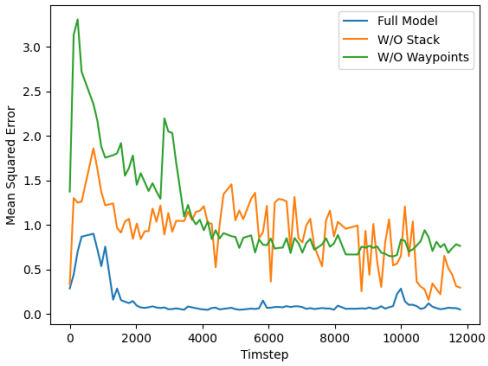

Figure 5: Performance of three vehicle dynamics models on a validation dataset. The full model includes the entire input space as described in the methods. W/O stack removes the $t-1$ history of the state (only includes state at time t). W/O Waypoints removes the positions of the waypoints relative to the vehicle

6.3 Motion Forecasting Performance Evaluation

Evaluation Metric Definitions:

- $H(p,q)$ Test log-likelihood: $H(p,q) = -\mathbb{E}_{\mathbf{S}^* \sim p(\mathbf{S}^*|\phi)} \log q(\mathbf{S}^*|\phi)$, it is given by the forward cross-entropy.
- $\hat{e}$ Extra nats: $\hat{e} \doteq \big[H(p',q) - H(\eta)\big](TAD) \geq 0$, it is the normalized extra nats above 0.
- $\hat{m}$ min Mean Squared Displacement(MSD): for each ground truth trajectory, it computes the minimum mean error between the ground truth trajectory and its associated forecast trajectories.

Results are presented in Table 2 and additional trajectory predictions in Figure 6

$Mean\ H(p,q)$	-83.768 $\pm$ 0.109
$Mean\ \hat{e}$	1.092 $\pm$ 0.003
$Mean\ \hat{m}$	1.333 $\pm$ 0.041

Table 2: Additional Motion Forecasting Performance Metrics

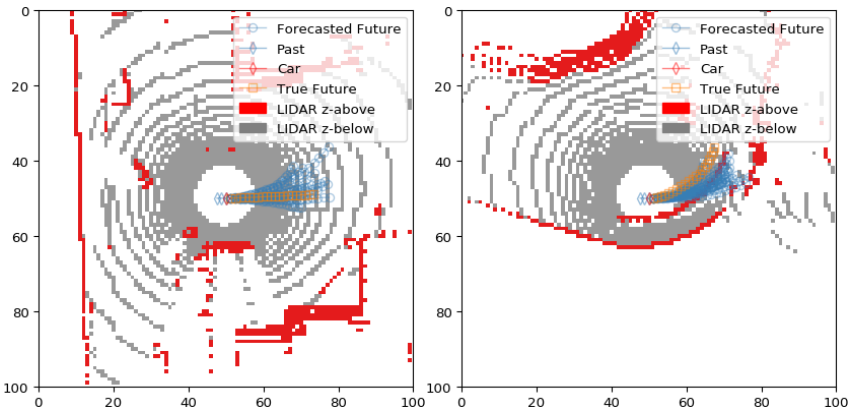

Figure 6: Additional Trajectory Predictions from the Motion Forecasting Model

6.4 Comparison of Policy and Dataset Actions

Figure 7: Comparison of target speed selected by trained policy with target speed in dataset

Figure 8: Comparison of steer angle selected by trained policy with target speed in dataset