Learning Highway Ramp Merging Via Reinforcement Learning with Temporally-Extended Actions

Samuel Triest[1], Adam Villaflor[2] and John M. Dolan[2]

***Abstract*— Several key scenarios, such as intersection navigation, lane changing, and ramp merging, are active areas of research in autonomous driving. In order to properly navigate these scenarios, autonomous vehicles must implicitly negotiate with human drivers. Prior work in driving behaviors presents reinforcement learning as a promising technique, as it can leverage data as well as the underlying decision-making structure of driving with interaction. We apply a hierarchical approach to decision-making, where we train a high-level policy using reinforcement learning, and execute the policy's output on a low-level controller. This hierarchical structure helps increase the policy's overall safety, and allows the learning component to be agnostic to the low-level control scheme. We validate our approach on a simulation using real-world highway data and find improved results compared to prior work in ramp merging.**

I. INTRODUCTION

Autonomous driving-related technology has been a major research topic for the last several decades. Although there exist a number of commercially available advanced driving assistance systems (ADAS) for highway driving in today's vehicles, there remain a number of scenarios that are difficult for ADAS. Such scenarios, which include highway ramp merging, are challenging for ADAS to properly negotiate because of the need to interact with other vehicles. Designing agents that can successfully execute these merging maneuvers in a wide variety of driving conditions is a major challenge, as the ego-vehicle must be able to interpret the intent of other vehicles and react accordingly. Furthermore, the long adoption window of autonomous vehicles means that autonomous vehicles will share the road with human drivers for some time. This necessitates that autonomous agents be able to interact with drivers in a human-like way.

Given the above challenges, there is recent work in data-driven approaches to designing autonomous vehicle behaviors. Representative data-driven techniques for designing these agents include probabilistic graphical models [1], [2], reinforcement learning [3], [4], and imitation learning via inverse reinforcement learning [5].

Many reinforcement learning-based approaches rely on learning the merging policy end-to-end. That is, they train a policy to output driving controls (such as acceleration and heading) given an observation. Unlike previous methods, we propose a hierarchical structure in which the trained policy instead outputs high-level actions given an observation, which is then executed by some rule-based controller.

[1]Samuel Triest is with the Computer Science Department of University of Rochester, Rochester, NY 14627 USA, striest@u.rochester.edu

[2]Adam Villaflor and John M. Dolan are with The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA

II. RELATED WORK

Work in designing merge behaviors for autonomous vehicles has been considered by a number of researchers, and several representative solutions exist as a result.

In the 2007 DARPA Urban Grand Challenge, Urmson et al. [6] use a slot-based approach, where a number of potential merge slots (i.e., between car 1 and car 2, between car 2 and car 3) are identified. The feasibility of each slot is evaluated using a rule-based planner, and the ego-vehicle uses a motion planner to attempt to merge into the best slot.

More recently, a number of learning-based approaches to ramp-merging have been considered. Dong et al. [1], [2], [7] use probabilistic graphical models (PGMs) to estimate the host vehicle's intent to yield. Given the intent to yield from the PGM, the merging vehicle can use traditional control techniques to execute the merge. The design of the PGM relies on observed velocity data from a fixed number of past timesteps, as well as the time-to-arrival to a fixed merge point. By encoding the relationship between these observations and intent to yield in a PGM, the authors are able to predict a given host vehicle's intent to yield to a given merge vehicle [1]. The authors extend this framework to apply to multiple host and merge vehicles without the assumption of a fixed merge point [2], [7].

Hu et al. [3] and Bouton et al. [4] use reinforcement learning to solve the merging problem. Hu et al. use an actor-critic-based approach, where a centralized and decentralized critic are used together to encourage vehicles to maximize their own performance while still acting cooperatively. Bouton et al. employ a single-agent approach based on Deep Q-Learning (DQN) [8] with an additional belief on the cooperativeness of the drivers in the scenario. This belief is used as additional input to their policy. Both RL-based approaches use a low-level action space of acceleration changes (Δa) discretized into the set $\{-1, -0.5, 0, +0.5, +1\} m/s^2$ as well as two additional actions to set the acceleration (a) to $0 m/s^2$ or $-4 m/s^2$. Both approaches assume that merging happens at a fixed point, and that the ego-vehicle follows a predetermined path to the merge point.

Hu et al. [3] and Mukadam et al. [9] incorporate low-level controllers into their reinforcement learning algorithms via a masking mechanism. At a given state, a low-level controller is used to determine feasible actions. The policy is not allowed to execute the infeasible actions, thus improving safety and learning speed (though no ablation studies are provided to quantify the improvement in learning speed.)

In a recent survey of RL applications to autonomous

driving, Aradi [10] notes that RL approaches to merging often attempt to learn a direct mapping from observation to vehicle control. This often takes the form of a 1-D action space over longitudinal accelerations [3], [4], but may include steering control [11], as well as lane-change primitives [12].

In addition to using temporally-extended actions to execute merging behaviors, this paper attempts to validate some traditional RL-based approaches on real highway data. These data are collected from the Next Generation Simulation dataset for highway driving (NGSIM) [13], which provides a challenging set of merging scenarios where vehicles do not conform to a single analytical model of driver behavior. We also propose a different method of using reinforcement learning (temporally-extended actions) to execute merge behaviors.

III. BACKGROUND

We introduce background material on partially observable Markov decision processes (POMDPs) and reinforcement learning (RL) to familiarize the reader with our terminology and notation.

A. Partially Observable Markov Decision Processes

A POMDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{O}, R, O, \gamma, d_0)$, where $\mathcal{S}, \mathcal{A}, \mathcal{O}$ are the state space, action space and observation space, respectively, R, O are the reward and observation models, $\mathcal{T}$ is the transition dynamics, and γ is the discount factor. d_0 is the initial state distribution of the POMDP.

A policy π provides a means of selecting actions for a given state s in a traditional MDP. For POMDPs, we assume that the policy does not have access to the true state s, and instead has access to an observation $o \sim \mathcal{O}(s)$. In this work, we consider π to be a stochastic policy. That is, for a given observation o_t, $\pi(o_t)$ is a probability distribution over the elements of $\mathcal{A}$. The POMDP is transitioned to a new state according to:

$$s_{t+1} \sim \mathcal{T}(\cdot, a_t \sim \pi(o_t), s_t)$$

The efficacy of a policy π is determined through the expected discounted sum of rewards, which we express as:

$$J(\pi) = \mathop{\mathbb{E}}_{\substack{s_0 \sim d_0, a_t \sim \pi(s_t, \cdot), \\ s_{t+1} \sim \mathcal{T}(s_t, a_t, \cdot)}} [\Sigma_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1})]$$

B. Reinforcement Learning

Reinforcement learning provides a means of optimizing a policy π to solve

$$\theta^* = \arg\max_{\theta} J(\pi_\theta)$$

where π is parameterized by θ, often as a neural network.

We can optimize a policy π_θ iteratively using the gradient of $J(\pi)$ with respect to θ. This gradient takes the form:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi} [\Sigma_{t=t'}^{T} \nabla_\theta \log \pi(a_t | s_t) Q^\pi(s_t, a_t)] \quad (1)$$

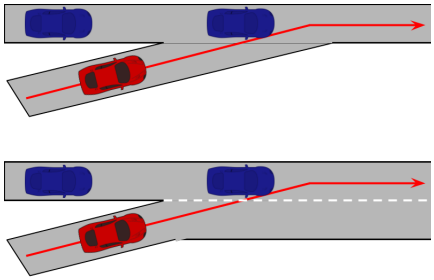

Fig. 1. The merging problem - the ego vehicle (in red) needs to move from the merging lane to the host lane without disrupting the host vehicles (in blue). Two ramp geometries are shown - the top geometry does not contain an auxiliary lane, while the bottom one does.

where $Q^\pi(s_t, a_t)$ denotes the expected discounted sum of rewards from taking action a_t in state s_t. This quantity can be computed via a Monte-Carlo sample of trajectory returns [14], or via some parameterized estimator (such as another neural network) [15].

C. Options Framework and Semi-Markov Decision Processes

The options framework [16] is an extension to traditional Markov decision processes that allows for hierarchical selection of actions. The options framework allows for a set of sub-policies or *options* of the form $< \mathcal{I}_i, \pi_i, \beta_i >$ that interact with a traditional MDP. π_i is a policy that maps states to one-step actions, $\mathcal{I}_i$ is a set of states in which policy π_i can be executed, and β_i is a function $s \rightarrow [0, 1]$ denoting a probability that the policy π_i will terminate in state s. Options can also be Semi-Markov; that is, they can take as input a history h comprised of a fixed number k of previous states.

In addition to the set of options, the options framework allows for a policy μ that selects options π, given a state s. Sutton et al. note that this framework is equivalent to the Semi-Markov decision process (SMDP) and techniques used to solve SMDPs (such as RL [17]) can be applied to the options framework.

IV. PROBLEM FORMULATION

A. The Merging Problem

In this work, we consider the merging problem to be one in which a **merging vehicle** is attempting to enter the lane of a **host vehicle**. The merging problem is illustrated in Figure 1. In general, the host vehicle has right-of-way and the merging vehicle is expected to yield to the host vehicle. A behavior that is able to successfully negotiate the merging problem will allow the merging vehicle to safely enter the host vehicle's lane with minimal disruption to the host vehicle. For this work, we will consider the ego-vehicle to be a merging vehicle.

B. Merging as a POMDP

In order to use reinforcement learning techniques to design a high-level policy, we formulate highway merging as a partially observable Markov decision process (POMDP).

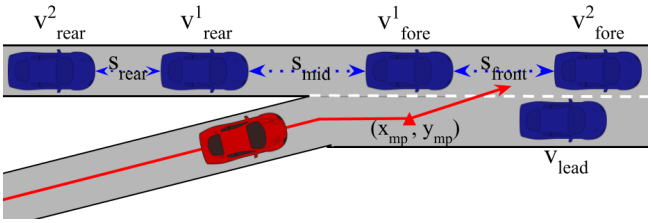

Fig. 2. Diagram of the ego-vehicle's observation space. The ego-vehicle is in red. The vehicles in $\{v_{lead}, v^2_{fore}, v^1_{fore}, v^1_{rear}, v^2_{rear}\}$ are depicted, as well as $\{s_{front}, s_{mid}, s_{rear}\}$ and the merge point. Features are extracted from these entities according to the procedure outlined in section IV.B.2.

1) States: Our state representation consists of the kinematic information of all vehicles in the merging scenario (position, velocity, acceleration, heading). Position, velocity and acceleration for non-ego vehicles are obtained through replaying data from NGSIM. Heading is not directly provided, but is easily computable using historical position information (i.e. $\theta_t = tan^{-1}(\frac{y_t - y_{t-1}}{x_t - x_{t-1}})$).

We define a terminal state as one in which the ego-vehicle is colliding with another vehicle, or if the ego-vehicle successfully drives 50m past the merge point. We choose a point a significant distance beyond the actual merge point to encourage the policy to merge safely (i.e., avoid aggressively entering the host lane and crashing immediately afterwards).

2) Observations: Due to factors such as occlusion and sensor inaccuracies, it is unreasonable to assume that the ego-vehicle can access the true state of the merging POMDP. As such, we limit the ego-vehicle's observation to vehicles in its immediate vicinity. In addition, the observation contains additional information derived from the state s that is relevant to the merging problem.

Instead of global position, the ego-vehicle observes displacements $(\Delta x_{mp}, \Delta y_{mp})$ from a fixed merge point in a lane-relative frame. The ego-vehicle is also only able to observe the information of a limited subset of vehicles - the vehicle immediately in front of it, as well as four vehicles in the host lane, being the two closest vehicles with positive longitudinal displacement, and the two closest vehicles with negative longitudinal displacement. We denote this set as $V = \{v_{lead}, v^2_{front}, v^1_{front}, v^1_{rear}, v^2_{rear}\}$. The ego-vehicle is given a tuple of (x, y, v, t, l, w) for each of the vehicles in V, where x, y are the longitudinal and lateral displacement to the ego-vehicle, v, t are velocity and lane-relative heading, respectively, and l, w are the vehicle's length and width. By using this particular neighbor-choosing scheme, we are also guaranteed to have three potential merging slots defined: $S = \{s_{front}, s_{mid}, s_{rear}\}$. $s_{front}, s_{mid}, s_{rear}$ are defined by the tuples $(v^2_{front}, v^1_{front}), (v^1_{front}, v^1_{rear}), (v^1_{rear}, v^2_{rear})$ respectively. From each slot in S, we extract a tuple (d, v), where d is the length of the slot, and v is the rate at which d is changing. These values are derived from the positions and velocities of the vehicles that define the given slot. We concatenate all of these features together in order to construct our $\mathcal{O}(s)$.

3) Actions: Unlike prior RL-based approaches to merging, we use a high-level action space in our POMDP formulation. Our action space is the set $a \epsilon \{v_{lead}, v^2_{fore}, v^1_{fore}, v^1_{rear}\}$, where each element of the set corresponds to a distance-keeping behavior given by an underlying rule-based controller $\pi_C(\overrightarrow{o}, a)$. Distance-keeping to each of these vehicles (except v_{lead}) corresponds to choosing a slot to merge into. We can incorporate these actions into our action space by using π_C as a Semi-Markov option $< \mathcal{I}, \pi_C, \beta >$ where:

$$\mathcal{I} = \mathcal{S}$$

$$\pi(h, u) = \begin{cases} 1 & \text{if } u = \pi_C(h_N) \\ 0 & \text{otherwise} \end{cases}$$

.

$$\beta(h) = \begin{cases} 1 & \text{if } |h| = k \\ 0 & \text{otherwise} \end{cases}$$

where u denotes a low-level action, and N denotes the current history length. Note that although histories of observations h are used to formally define the Semi-Markov option for π_C, π_C is only dependent on the current observation $\overrightarrow{o} = h_N$. As noted by Bradke et al., if the transition times for all actions are the same, an SMDP can be treated as a traditional MDP. The various distance-keeping targets can be viewed as a non-learned set of options. Details of the interaction between the high-level action a and low-level controller C are given in Section V.

4) Reward: Our reward function is designed as follows:

$$r(s_t) = 10 * success(s_t) + 0.05 * (x_t - x_{t-1}), \quad (2)$$

where $success$ is 1 if the ego-vehicle is in a success state, and is -1 if the ego-vehicle is in a colliding state (both functions are 0 otherwise). We define a success state as the ego-vehicle being 50m past the merge point in the host lane. x denotes the longitudinal position of the ego-vehicle.

The first term encourages the agent to successfully merge without collision, while the last term encourages the agent to make forward progress. Since the simulation terminates when the ego-vehicle collides with another vehicle, colliding has an additional implicit penalty of preventing the agent from accruing any future reward.

V. METHODOLOGY

A. Hierarchical Control

We split the merging process into two distinct layers: high-level actions, and low-level control. The two layers interact in that the high-level actions specify behaviors that are executed by the low-level control.

Ultimately, our decision-making system takes the overall form:

$$\overrightarrow{u_t} = C_\psi(\overrightarrow{o_t} | \pi_\theta(\overrightarrow{o_t}))$$

where $\overrightarrow{o_t}$ is the vehicle's observation at time t, π_θ is a function that produces high-level actions, and C_ψ is a

function that produces low-level control given a high-level action, and is parameterized by ψ (note that in our work, C is a controller with adjustable parameters ψ and not a deep neural network). Of particular importance to the success of this procedure is that the high-level agent and low-level controller run at different frequencies f_π and f_C, where f_π is much lower than f_C. Since the high-level agent is designed to produce behaviors instead of control, it can be run at a much lower frequency while the actual control is generated at f_C via the low-level controller, conditioned on the high-level action.

This overall procedure is presented in Algorithm 1. It is assumed that the algorithm runs at f_C. Essentially, at every timestep, the low-level controller outputs a control action conditioned on the high-level action. Every f_C timesteps, the high-level agent is able to choose a new high-level action.

Algorithm 1: End-to-end Control via HRL

Input: Merging POMDP M with transition and observation models $\mathcal{T}_M, \mathcal{O}_M$ respectively, high-level agent π and its frequency f_π, low-level controller C and its frequency f_C

Output: Acceleration and steering a_t and θ_t

while *not done* **do**
- $o_t \sim \mathcal{O}_M(s_t)$
- **if** $t \,\%\, f_C == 0$ **then**
  - $v_{target} = \pi(o_t)$
- **end**
- $\overrightarrow{u_t} = C(o_t | v_{target})$
- $s_t \sim \mathcal{T}_M(\cdot, \overrightarrow{u_t}, s_t)$
- $t = t + 1$

end

B. High-level Decision-Making

We implement our high-level decision-making agent π_θ as a neural network parameterized by θ, which we train using reinforcement learning.

Our action space follows from Dong et al. [1], [2], [7] in which the output of π_θ will be a discrete action $a_t \epsilon \{v_{lead}, v^2_{fore}, v^1_{fore}, v^1_{rear}\}$ corresponding to a vehicle to which to distance-keep.

C. Low-level Control

We implement our low-level controller as a simple proportional controller on the acceleration and heading of the autonomous vehicle. Given a vehicle to distance-keep to from π_θ, our controller C outputs acceleration and heading commands for the ego-vehicle according to Algorithm 2.

An example controller is presented in Algorithm 2. At a high level, the controller causes the ego-vehicle to distance-keep to a point slightly behind the target vehicle in the longitudinal direction. We calculate this value as $x_{target} - x_{ego} - x_{min}$, where $x_{target} - x_{ego}$ is the longitudinal distance between the vehicles. By subtracting a safety distance x_{min}, our controller produces accelerations to distance-keep x_{min} behind the target vehicle. The controller also causes the ego-vehicle to move toward the target lane centerline in the lateral direction. Similarly, our controller drives $y_{ego} - y_{lane}$ to 0, where y_{lane} will be either the on-ramp or the target lane, depending if the vehicle has passed an arbitrary merge point. Since the target lane changes when the vehicle passes the merge point, the controller is able to execute the merge. Accelerations are clipped within $\pm 4.25 m/s$ to ensure that controller actions are executable by real vehicles, and headings are clipped based on turn rate to ensure smooth lane transitions.

Algorithm 2: Low-level controller = $C(s_{ego}, y_{host}, y_{merge}, s_{target})$

Input: State of ego vehicle $(x_{ego}, y_{ego}, \theta_{ego})$, target vehicle $(x_{target}, y_{target}, \theta_{target})$ and lateral distances to centerlines of host and merge lanes y_{host}, y_{merge}).
Hyperparameters for minimum safe longitudinal distance x_{min}, limits on turn rate ω_{max} and acceleration a_{min}, a_{max}, merge point location (x_{mp}, y_{mp}) and control frequency dt.

Output: Acceleration and steering a_t and θ_t

$$a_{t+1} = K_a(x_{target} - x_{ego} - x_{min})$$
$$\theta_{t+1} = \begin{cases} K_\theta(y_{ego} - y_{merge}) & \text{if } x_{ego} < x_{mp} \\ K_\theta(y_{ego} - y_{host}) & \text{if } x_{ego} > x_{mp} \end{cases}$$
$$a_{t+1} = clip(a_{t+1}, a_{min}, a_{max})$$
$$\theta_{t+1} = clip(\theta_{t+1}, \theta_t - \omega_{max} * dt, \theta_t + \omega_{max} * dt)$$
return a_{t+1}, θ_{t+1}

D. Algorithm

We use Advantage Actor-Critic (A2C) [18] to train our high-level policy. We also implemented several Q-learning-based approaches, and found that they significantly underperformed the policy gradient-based approaches.

VI. IMPLEMENTATION

A. Environment

The merging scenarios for training were obtained from the NGSIM dataset [13]. NGSIM contains two sets of human driving trajectories - one is extracted from US Highway 101 and the other from Interstate 80. Both datasets contain an on-ramp to extract merging scenarios from. Additionally, both datasets were collected around rush hour, and contain various degrees of traffic density. The varying degree of traffic density ensures that the scenarios provide a variety of challenging merge scenarios for the ego-vehicle to negotiate. Data are recorded at 10Hz.

Individual merging scenarios were extracted from NGSIM by identifying all vehicles that start in the merge lane. A vehicle is chosen from this set to be the ego-vehicle. Vehicles that do not enter either the host lane or merging lane are removed from the scenario. Additionally, vehicles behind the

ego-vehicle in the merge lane are removed from the scenario. The remaining vehicles are replayed directly from the data. In total, there are 402 different merging scenarios, of which we hold out 127 (32%) for testing.

B. Control using HRL

We choose to run our high-level agent at 1Hz and low-level controller at 10Hz (to match NGSIM). Both the controller and agent use the observation generated from the environment to compute their corresponding actions. We instantiate the low-level controller and high-level agent and train the high-level agent with the hyperparameters specified in Tables I and II.

TABLE I
HYPERPARAMETERS FOR LOW-LEVEL CONTROLLER

Hyperparamter	Value	Range	Units
Acceleration K_p (K_a)	1.5	± 1.0	
Minimum safe distance (x_{min})	4	± 2.0	m
Min/Max Acceleration (a_{min}, a_{max})	± 4.25	± 0.0	m/s^2
Steering K_p (K_θ)	2.0	± 1.0	
Steer rate threshold (ω_{max})	15	± 10.0	deg/s

TABLE II
HYPERPARAMETERS FOR HIGH-LEVEL AGENT

Hyperparamter	Value
Neural Network Architecture	2 × [Dense(128)]
Training Steps	6×10^6
Activation Function	Tanh
Entropy Regularizer	0.01
Critic network updates/policy update	80
Optimizer	Adam [19]
Learning Rate	5×10^{-4}
Discount Factor	0.99

VII. RESULTS AND DISCUSSION

A. Baselines

We compare our work to MML-PGM by Dong et al. [2]. This method is selected because of its ability to handle multiple merging vehicles at once, which is more in line with the simulation environments that we evaluate our results on. We also compare our approach to the slot-based approach in Urmson et al. [6] to verify that our high-level policy yields improvement over rule-based methods for merging.

We also compare our approach to an approach that learns low-level control (i.e., acceleration commands on a fixed path) on real-world data. We use A2C to train a policy that uses an action space inspired by Hu et al. [3], where the agent chooses an action from the set $\Delta a = \{-1, -0.5, 0, +0.5, +1\} m^2$ with additional actions that instantaneously set the acceleration to either $0 m/s^2$ or $-4 m/s^2$. The low-level policy runs at 5hz. We also incorporate cooperativeness estimates based on the CIDM model proposed by Bouton et al. [4] into the agent's observation space.

Our RL-based approach is also validated with noise in both the observation space and controller space. Observation noise was implemented as Gaussian noise. Controller noise was implemented by uniformly sampling controller parameters from the ranges specified in Table I. In Table III, all policies were trained and tested without noise. In Table IV, all policies were trained as indicated by the method and tested as specified by the column.

We also varied the coefficients in our reward function to explore the effect that each term has on the policy's overall success. The success term provides a sparse reward that is directly related to the task, while the forward progress (FP) term provides a dense reward that does not explicitly reward successful merging. To explore the effect that each term has on the overall success rate, we ran an experiment each where we set the coefficient on one of these terms to 0 and report the results in Table V.

All approaches are validated on every held-out trajectory (n=127) across 10 random seeds. A hyperparameter search was performed individually on each RL-based approach and the maximum performance on the held-out trajectories is reported.

TABLE III
COMPARISON OF HRL TO EXISTING METHODS ON NGSIM (ON-RAMP)

Method	Collision Rate
Low-level RL	60.0%
Slot [6]	12.9%
MML-PGM [2]	9.9%
HRL (Ours)	**4.2%**

TABLE IV
PERFORMANCE OF HRL WITH OBS AND CONTROLLER NOISE ON NGSIM (ON-RAMP)

Method	w/o Noise	w/ Noise (std=1)
HRL	**4.2%**	**4.5%**
HRL + obs noise (std=1)	5.2%	5.4%
HRL + obs noise (std=3)	8.9%	8.7%
HRL w/ C noise	4.9%	5.1%

TABLE V
PERFORMANCE OF HRL WITH VARYING REWARD COEFFICIENTS ON NGSIM (ON-RAMP)

Method	Collision Rate
FP + Success	**4.2%**
FP, Success = 0	8.2%
Success, FP = 0	9.5%

B. Results and Discussion

Overall, we find that learning of high-level actions outperforms other methods in terms of collision rate. Noting Table III, we can see that our hierarchical approach achieves a lower collision rate of 4.2% compared to the learned PGM approach (9.9%) and the slot-based approach (12.9%). We note that no method achieves a collision rate of 0. Since data are replayed from NGSIM, the neighboring vehicles do not necessarily respond to the ego-vehicle, as a human would.

In particular, we observe that there is a significant disparity ($\sim 55\%$) in the performance of an RL agent trained with high-level actions and one trained with low-level actions. While a low-level action space has been shown to work effectively on simulated driver data [3], [4], we believe that the complexity of real human driving compared to simulated

models of human behavior makes it much more difficult to learn in a low-level action space. Using temporally-extended actions via a controller allows for better exploration of the environment and shortens the horizon of the problem, thus making it easier to solve with RL.

From Table IV, we observe that the performance of the hierarchical approach is largely unaffected by Gaussian noise in the observation space (values of the observations fell in the range $[\pm 50]$, and collision rate remained within 0.5% of the collision rate without noise). We observe that training the policy with noisy observations did not yield a significant difference in performance, even when testing with noisy observations. In Table V, we can observe that using both the dense and sparse rewards yields better performance than either reward alone.

As mentioned before, our approach bears some similarity to the Semi-Markov Decision Process defined by Bradtke et al. [17], and the options framework described by Sutton et al. [16]. Our approach can also be viewed as an extension of Dong et al.'s MML-PGM [2] in a reinforcement learning setting. Both our approaches and MML-PGM rely on a low-level controller to execute high-level decisions as determined by some decision-making mechanism. We can thus think of the policy network that we train using RL as serving an equivalent function to the rules and yield prediction used by MML-PGM.

VIII. CONCLUSION

We have presented a study on using reinforcement learning to execute merging maneuvers on real highway data. Our study shows that reinforcement learning can be used to execute competent merging behaviors at a lower collision rate than other methods. Our study also shows that learning a high-frequency, low-level action space using RL on real data is challenging, and present learning hierarchically with a low-level controller as a potential solution.

A potential benefit of our approach is that it is agnostic to the low-level control scheme, and can thus benefit from better controllers, such as the model-predictive control used in Urmson et al. [6]. Further work could include applying hierarchical techniques to other scenarios. We believe that learning the low-level options along with the high-level policy may be beneficial, depending on the scenario.

IX. APPENDIX

Due to the disparity in velocity between the vehicles in the host and merging lane (in general, the traffic in the merging lane was much faster than that of the host lane), we observed that the PGM method provided by Dong et al. [1], [2] yielded aggressive merging behaviors. We were able to improve the performance of the PGM method to the values given in the paper by clipping accelerations outputted by the low-level controller to $\{-4.25m/s^2, 3.25m/s^2\}$ instead of the original $\{-4.25m/s^2, 4.25m/s^2\}$. Our scenarios also did not make the guarantee of having a lead vehicle. As such, we modified the MML-PGM algorithm to default to PGM (i.e., distance-keep to the last vehicle predicted to not yield) in the absence of a leading vehicle, or if the leading vehicle was beyond a distance of $25m$.

X. ACKNOWLEDGEMENTS

The authors would like to thank Zhiqian Qiao and Christoph Killing for their valuable discussions on past work in ramp merging and on reinforcement learning.

REFERENCES

[1] C. Dong, J. M. Dolan, and B. Litkouhi, "Intention estimation for ramp merging control in autonomous driving," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 1584–1589.

[2] ——, "Interactive ramp merging planning in autonomous driving: Multi-merging leading pgm (mml-pgm)," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2017, pp. 1–6.

[3] Y. Hu, A. Nakhaei, M. Tomizuka, and K. Fujimura, "Interaction-aware decision making with adaptive strategies under merging scenarios," *arXiv preprint arXiv:1904.06025*, 2019.

[4] M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer, "Cooperation-aware reinforcement learning for merging in dense traffic," *arXiv preprint arXiv:1906.11021*, 2019.

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

[6] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[7] C. Dong, J. M. Dolan, and B. Litkouhi, "Smooth behavioral estimation for ramp merging control in autonomous driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1692–1697.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[9] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, "Tactical decision making for lane changing with deep reinforcement learning," 2017.

[10] S. Aradi, "Survey of deep reinforcement learning for motion planning of autonomous vehicles," *arXiv preprint arXiv:2001.11231*, 2020.

[11] P. Wang and C.-Y. Chan, "Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2017, pp. 1–6.

[12] P. Wolf, K. Kurzer, T. Wingert, F. Kuhnt, and J. M. Zollner, "Adaptive behavior generation for autonomous driving using deep reinforcement learning with compact semantic states," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 993–1000.

[13] "Next generation simulation (ngsim)." [Online]. Available: https://ops.fhwa.dot.gov/trafficanalysistools/ngsim.htm

[14] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[15] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[16] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[17] S. J. Bradtke and M. O. Duff, "Reinforcement learning methods for continuous-time markov decision problems," in *Advances in neural information processing systems*, 1995, pp. 393–400.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.