2021 IEEE Intelligent Transportation Systems Conference (ITSC)
Indianapolis, USA. September 19-21, 2021

Trajectory Planning for Autonomous Vehicles Using Hierarchical Reinforcement Learning

Kaleb Ben Naveed[1], Zhiqian Qiao[2] and John M. Dolan[3]

***Abstract*— Planning safe trajectories under uncertain and dynamic conditions makes the autonomous driving problem significantly complex. Current heuristic-based algorithms such as the slot-based method rely heavily on hand-engineered parameters and are restricted to specific scenarios. Supervised learning methods such as Imitation Learning lack generalization and safety guarantees. To address these problems and to ensure a robust framework, we propose a Robust-Hierarchical Reinforcement Learning (HRL) framework for learning autonomous driving policies. We adapt a state-of-the-art algorithm, Hierarchical Double Deep Q-learning (h-DDQN), and make the framework robust by (1) constituting the decision of selecting driving maneuver as a high-level option; (2) for the lower-level controller, outputting waypoint trajectories to track with a Proportional-Integral-Derivative (PID) controller instead of direct acceleration/steering actions; and (3) using a Long-Short-Term-Memory (LSTM) layer in the network to alleviate the effects of observation noise and dynamic driving behaviors. Moreover, to improve the sample efficiency, we use Hybrid Reward Mechanism and Reward-Driven Exploration. Results from the high-fidelity CARLA simulator while simulating different interactive lane change scenarios indicate that the proposed framework reduces convergence time, generates smoother trajectories, and can better handle dynamic surroundings and noisy observations as compared to other traditional RL approaches.**

***Index Terms*— Trajectory Planning, Hierarchical Deep Reinforcement Learning, Double Deep Q-Learning, PID controller.**

I. INTRODUCTION

Planning safe trajectories for autonomous vehicles is a challenging problem. In reality, this problem is particularly difficult because of the maneuver planning complexities, stochastic surroundings, and noisy perception system of the car. While performing trajectory planning, the autonomous vehicle has to plan different maneuvers, which might include lane following, waiting, changing lanes, and traversing intersections.

Most existing trajectory planners rely on either traditional heuristic-based methods or machine learning methods. Some of the state-of-the-art heuristic-based methods include the slot-based method [1] and Time-To-Collision (TTC) [2]. The slot-based method selects a target slot through rules-enumeration from the set of feasible slots. On the other hand, TTC calculates the time to collision based on estimating the target cars' state information. Both methods require hand-engineered parameters and thus lack optimal solutions.

[1]Student of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong, China. kaleb-ben.naveed@connect.polyu.hk
[2]Electrical and Computer Engineering, Carnegie Mellon University
[3] The Robotics Institute, Carnegie Mellon University

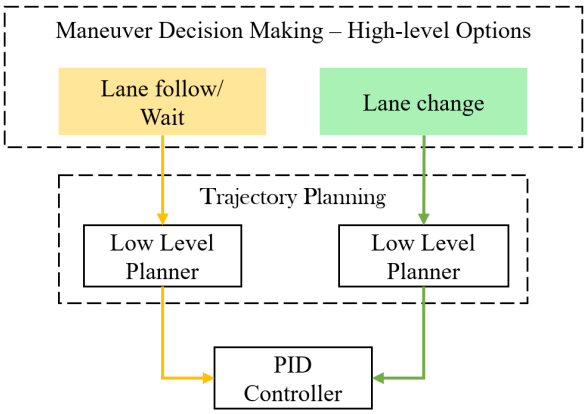

Fig. 1. Three-layer Hierarchical Structure has two high-level options in the first manuever decision making layer and a low-level trajectory planner for each high-level option, which generates waypoint trajectories. Proportional-Integral-Derivative (PID) Controller is used to follow the planned trajectory.

Moreover, these methods are not generalizable to more complex scenarios. The other way of planning trajectories is through machine learning methods. The supervised learning method called Imitation Learning [3] has shown some promising results. However, this method might not generalize well to complex conditions and does not guarantee stability and an optimal solution [4].

An alternative approach to the rule-based planners and supervised learning methods is Reinforcement Learning (RL) [5]. The RL framework works on the principle of maximizing reward for a particular action at a given state. Existing RL works have shown promising results for an ego-car to learn policies for multiple scenarios. However, traditional RL methods for autonomous driving are less sample-efficient and less stable, especially for tasks with multiple sub-goals. In comparison to traditional RL, Hierarchical Reinforcement Learning (HRL) [6] allows a model to learn the policies for multiple sub-goals, which allows the policies learned to be reused for any other scenario. Furthermore, HRL has shown a faster convergence rate, which decreases training time for the model to learn an optimal policy. In this paper, we propose Robust-HRL as an improvement to the existing HRL framework for learning an autonomous driving policy for the interactive lane change scenario. This paper's contributions towards making the HRL framework for trajectory planning more robust are:

- High-level decision making: The higher level of the HRL framework is responsible for selecting a maneuver

option, which could be either lane follow/wait or lane change;
- Planning smooth waypoint trajectories: Based on the high-level option, the low-level planner generates variable-length waypoint trajectories which are tracked with a PID controller;
- History of state observations: We use an LSTM layer with history of state observations to compensate for observation noise and to improve learning with interactive driving conditions;
- Improving sample efficiency: We use Hybrid Reward Mechanism [14] and Reward-Driven Exploration in order to improve sample efficiency and convergence time.

II. RELATED WORK

A. Non-RL-based Trajectory and Behavior Planning

Previous non–RL methods include classical planners, heuristic-based methods, supervised learning methods, and statistical methods.

Rapidly Exploring Random Trees (RRTs) is a state-of-the-art sampling-based classical planner [7]. RRTs generate trajectories by constructing a tree-like structure through the space. RRTs are proven good for environments with obstacles, but might not converge to the optimal solution. [8] addressed this problem by introducing RRT*, which showed optimal convergence and shorter routes. The methods, which work on rule-based enumeration, include Time-To-Collision (TTC) [2] and the slot-based method [1]. TTC estimates arrival time for the vehicle and thus the time to collision. On the other hand, the slot-based method checks whether it is safe to merge into the target-lane or to traverse the intersection based on the slots present in the target-lane and state information of the target cars. These heuristic rule-based methods are time-consuming to develop and also lack generalization to every possible scenario.

Alternative approaches include decision making through supervised learning and statistical methods. Imitation learning requires a considerable amount of data from expert drivers in order to generate the autonomous driving polices. [9] used a Deep Imitation Learning framework to learn a driving policy for urban scenarios through offline learning. They also added a safety controller module which increased the safety while testing. Another method, UrbanFlow [10], uses supervised learning for trajectory prediction. This method consists of a complete pipeline from collecting raw data to the final processing of trajectories. They used the UrbanFlow pipeline for LSTM-based trajectory prediction based on human drivers' driving behavior, which allowed the ego-car to make better decisions. For statistical methods, [11] proposed a behavior prediction and multipolicy decision-making framework for autonomous vehicles using a changepoint-based method. They sample policies of the target cars from the distribution, which is estimated using Bayesian changepoint detection on state history, and then create a likelihood of actions for the target cars. Based on the prediction and by modelling interactions between ego-car and target cars, they select a policy for the ego-car which yields highest reward.

All these approaches are either computationally expensive, not generalizable to all scenarios, or do not promise an optimal solution. On the other hand, RL has shown promising results handling these problems.

B. RL-Based Trajectory and Behavior Planning

By extending the framework of RL, [6] proposed the idea of hierarchical Deep Q-Network (DQN), in which the action-value functions were integrated to operate at two levels of abstraction, and learned a policy over metagoals and low-level actions. Later improvements to Hierarchical RL were proposed by [12] and [13]. In the autonomous vehicles domain, [14] proposed an HRL-based method for behavior planning with high-level options and low-level actions. Furthermore, in order to improve sample efficiency and yield better results, the state-attention model, hybrid reward mechanism, and hierarchical prioritized experience replay were introduced. Another work which made use of hierarchical abstraction and used the scenario of lane-change was proposed by [15]. They used RL for high-level decision making and optimization or rule-based methods for a low-level controller. Moreover, they considered prior knowledge, low-level controller parameters, and constraints for training which improve convergence time.

Another approach that has produced significant results in decision making through trajectory prediction is Inverse Reinforcement Learning (IRL). IRL works on the principle of extracting the reward structure by observing an optimal trajectory of an expert agent. By building on the IRL approach, [16] proposed a framework for predicting off-road vehicle trajectories by integrating kinematics and environment to recover the reward structure.

Another RL approach for decision making is multi-agent RL. [17] proposed a safe, multi-agent RL framework to improve the functional safety of the ego-car in multi-agent settings by using policy gradient iteration without the Markovian assumption and Hierarchical Temporal abstraction through desires (learned) and trajectory planning with hard constraints (not learned).

In this paper, we propose Robust-HRL, which is an improvement on the existing HRL framework for learning autonomous driving policy. We learn policy for both a high-level decision making network and a low-level trajectory planner network. Moreover, we facilitate learning in stochastic conditions by using the history of state observations and LSTM layers in the network for training. Lastly, in order to improve sample efficiency, we use Reward-Driven Exploration and Hybrid Reward Mechanism [14].

III. PRELIMINARIES

A. Double Deep Q-Learning

Double Q-Learning is an extension of the state-of-the-art Deep Q-Learning algorithm. The Q-Learning algorithm in RL is used to find an optimal action-selection policy π using

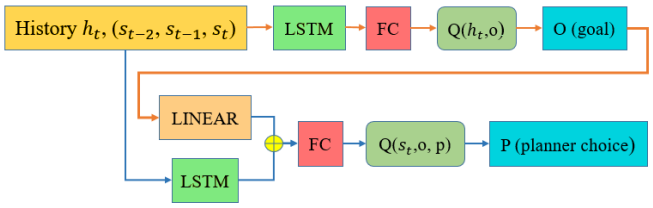

Fig. 2. The Hierarchical RL network consists of a high-level Options Network and low-level Planner Network. In both networks, a Long Short Term Memory (LSTM) layer is used with a fixed number of previous steps $n_{steps} = 3$ for the history vector input. The $Tanh$ activation function is used in the LSTM layer. The output of the Options network, which is the goal, is concatenated with the history vector for the input to the planner network. A fully connected (FC) layer is used as a last layer for each network. Within the FC, a $ReLU$ activation function is used except for the last layer, which uses a $Linear$ activation function in order to generate both of the networks' final values.

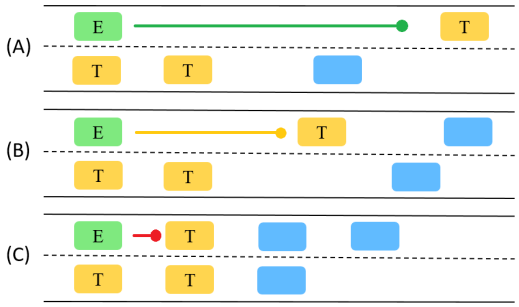

Fig. 3. Variable-length sub-trajectories for lane follow are generated based on different traffic density and conditions. Green objects with letter E are the ego-car and objects with letter T are target vehicles whose features are included in the state space. Blue objects represent other vehicles whose features are not included in the state space.

a Q function, which is used to maximize the action-value function $Q^*(s, a)$.

Deep Q-Learning uses neural networks to update network parameter θ through minimizing the loss function between predicted action-value Q and the target action-value Y^Q. Deep Q-Learning uses the same values to select and evaluate an action, which results in overestimation.

Double Deep Q-Learning [18] solves the problem of overestimation by revising the target action updates from another target Q' network with different weights, which is shown in Equation 1.

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q'(S_{t+1}, a|\theta_t)|\theta_t^{'}) \quad (1)$$

B. Hierarchical Reinforcement Learning

HRL [6] learns a policy at multiple levels as meta-controller Q^1 generates the sub-goal g for the following steps and a controller Q^2 outputs the action a based on the sub-goal selected until the next sub-goal is generated by the meta-controller.

$$Y_t^{Q^1} = \sum_{t'=t+1}^{t+1+N} R_{t'} + \gamma \max_g (S_{t+1+N}, g|\theta_t^1) \quad (2)$$

$$Y_t^{Q^2} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a|\theta_t^2, g) \quad (3)$$

IV. METHODOLOGY

A. Hierarchical structure and decision making

In the proposed Robust-HRL framework we use a three-layer structure for decision making and trajectory planning with two fully connected networks: one for high-level decision making and the other for low-level trajectory planning. The topmost level is responsible for selecting the high-level maneuver option from lane follow/wait or lane change options. Once the high-level selection is made, the information is passed down to the low-level planner, which generates waypoint trajectories based on learned policy. After that a PID controller is used to track trajectories. The division through HRL into two high-level options (lane follow/wait and lane change) helps the ego-car to learn a policy for both high-level options and for the low-level trajectory planners. The details of the hierarchical network structure can be seen in Figure 2.

B. Trajectory Planning and waypoint generation

Trajectory planning is implemented at the second level of the hierarchical framework. Once the high-level option is selected, the low-level trajectory planner selects the final waypoint from the discrete waypoint choices. The selection is based on the ego-car's state information through the epsilon-greedy strategy. All final waypoints are generated at the center of the lane, which prevents lane invasions and ensures stability. Once the final waypoint is selected, the ego-car's target speed is calculated using the maximum acceleration/deceleration constraints in order to ensure a smooth sub-trajectory. Then the target speed and final waypoint values are given to the PID controller, which in turn generates longitudinal and lateral control. These sub-trajectories altogether form a complete trajectory constituting lane follow, wait and lane change maneuvers. The detailed working of the Robust-HRL, including training details, is shown in Algorithm 1. The details of the decision-making strategy for low-level planner choices can be found in the sections below:

1) Lane Follow/Wait: Once the ego-car selects the lane follow/wait option, the low-level trajectory planner is used to plan the path. The low-level planner generates variable-length trajectories based on different driving scenarios. In figure 3, three scenarios are considered. In scenario A, the ego-car selects the more distant final waypoint, which lets it plan a sub-trajectory for a longer time horizon. In scenario B, an intermediate-length sub-trajectory is generated given denser traffic conditions as compared to scenario A. In scenario C, the ego-car decelerates to a slower speed profile trajectory or chooses to wait in the ego-lane in order for traffic to clear. For all trajectories, the final waypoint and target speed information is given to a PID controller for smooth tracking.

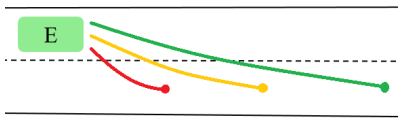

Fig. 4. Different speed profile sub-trajectories are generated for the ego-car (E) for a lane change maneuver based on different traffic densities and ego-car speed profiles.

2) Lane Change: Once the decision to perform a lane change is made, the waypoint in the target-lane is selected using the ego-car's state information through the epsilon-greedy strategy. Each different final waypoint shown in Figure 4 represents a different velocity profile the ego-car may choose to make a lane change. This ensures smoothness and stability while planning for the lane change maneuver. The green point is selected if the ego-car is required to do a faster lane change because of the higher speed in the ego-lane. The yellow point is for the normal velocity profile trajectory. The red point selection helps the ego-car to take sharper turns, which might be needed when the velocity of the ego-car is lower or it was in the wait option. The RL selects one point between the three points through the low-level planner network under the lane change high-level option.

C. History of state observations

We use the history of state observations as an input for both LSTM layers in our model to compensate for observation noise in the state space and to facilitate learning in interactive and stochastic driving conditions. In order to sample experiences from replay memory, we use the approach of bootstrapped random updates proposed by [19]. This strategy randomly samples an n-step sequence from the batch of episodes drawn from experience replay and then trains the neural network. The hierarchical network structure details can be found in Figure 2.

D. Reward-Driven Exploration

One of the most common strategies for training in reinforcement learning is the epsilon-greedy strategy. This strategy works on the principle of the explore-exploit dilemma, in which the agent's tendency to explore for the optimal policy decays with time. In our training routine, instead of periodically decaying the ϵ we use average total reward to adjust it. When the average reward is higher for a period of episodes, we decrease ϵ to favor exploitation and vice versa. This reduces convergence time and helps the ego-car explore more while training in difficult scenarios.

V. EXPERIMENTS

In this section, we apply the proposed Robust-HRL algorithm to the scenario shown in Figure 5 in the CARLA simulator [20]. In order to validate the effectiveness of our proposed approach, we compare Robust-HRL with previous state-of-the-art RL approaches and a heuristic-based method [1]. We also stress-tested the state space by introducing Gaussian noise in the observation vector.

Algorithm 1 Robust-HRL

Initialize options, target options, planner, and target planner network Q^o, $Q^{o'}$, Q^p, and $Q^{p'}$ with weights θ^o, $\theta^{o'}$, θ^p, and $\theta^{p'}$ respectively.
Construct empty replay buffer B with max memory length l_b.
for 1 to E training episodes **do**
  Initialize episode reward r^e and reward Deque r^e_t with length $2t$, where $t \in Z^+$
  Get Initial State s.
  Initialize history vector h_t containing 3 time steps.
  while s is not terminal state **do**
    o_t = $\arg\max_o$ $Q(h_t)$ based on the $\epsilon - greedy$, where o_t is the option.
    p_t = $\arg\max_p$ $Q(h_t, o_t)$ based on the $\epsilon - greedy$, where p_t is planner waypoint choice.
    Get the target waypoint w_t based on goal o_t and planner p_t selection.
    Calculate the target velocity value v based on w_t.
    Get throttle and steer values from PID controller based on v and w_t.
    Perform the step for sub-trajectory in simulation and get s_{t+1}, r^o_{t+1}, r^p_{t+1}, where r^o is the option reward and r^p is the planner reward.
    Deque s_{t+1} to h_t to get h_{t+1}.
    Store transition T into B: $T = \{s_t, h_t, o_t, p_t, r^o_{t+1}, r^p_{t+1}, s_{t+1}, h_{t+1}\}$.
    $r^e \leftarrow r^o_{t+1} + r^p_{t+1}$
  end while
  Append r^e to r^e_t
  if $\sum_{\frac{t}{2}}^{0} r^e_t < \sum_{t}^{\frac{t}{2}} r^e_t$ **then**
    $\epsilon = \gamma \times \epsilon$, $\gamma \in [0, 1]$
  else
    $\epsilon = \epsilon$ / γ, $\gamma \in [0, 1]$
  end if
  Train with Buffer $RelayBuffer(e)$.
end for

A. Scenario and Experiment Setup

The overview of the scenario can be seen in Figure 5. We considered the interactive lane change scenario for conducting experiments and testing our algorithm. We simulated three scenarios: High, Moderate, and Low traffic flow. In order to add complexity, we randomly blocked the ego-lane in some training episodes.

In addition, in order to simulate real-world driving behaviors, we used CARLA's Traffic Manager module to randomly generate target cars' (red cars in Figure 5) behavior. Using Traffic Manager, all target cars were assigned to Autopilot status, which uses a rule-based method and PID controller to control vehicles. Under Autopilot, all target vehicles react to each other and the ego-car, which makes the environment interactive. Moreover, using Traffic Manager, we randomly assigned the status of dangerous to some target cars by giving

Fig. 5. A screenshot of one of the CARLA simulator scenarios is shown, where the ego-car is the blue car in the white rectangle. In this scenario, the ego-lane is randomly blocked and the ego-car is required to perform a lane change.

them higher velocity and less distance to the front vehicle.

B. State Space

For all the scenarios, we selected three nearest target-cars to be included in the state observation vector. The information about target cars is included in the tuple $t \in \{t_1, t_2, t_3\}$, where t_1, t_2, and t_3 refer to target 1, target 2, and target 3 car, respectively. The state space consists of 14 parameters and is given by tuple s:

$$s = [v_e, lane_{ide}, v_t, d_t, d_{tr}, lane_{idt}]$$

- v_e = Ego-car velocity
- $lane_{ide}$ = Lane-ID for the ego-car
- v_t = Velocities of the target vehicles
- d_t = Ego-car distance to the target cars
- d_{tr} = Ratio of distance to safety threshold
- $lane_{idt}$ = Lane-ID of the obstacle car and target vehicles

C. Reward Structure

We use Hybrid Reward Mechanism [14] for our reward structure, which divides total reward into task reward and sub-goal reward. We did not use an unsmoothness penalty, as smooth trajectories are ensured by the PID controller. The overall reward can be categorized into two parts:

1) For each step:

- Time penalty: $-\sigma_1$
- Regular time step reward for progressing towards final destination: σ_2
- Unsafe penalty: $\exp -(d_{tr})$, where $t \in \{t_1, t_2, t_3\}$

2) For the termination conditions:

- Collision penalty: $-\sigma_4$
- Success reward σ_5
- Time out penalty, which results from excessive waiting: $-\sigma_6$

D. Results and Discussion

In order to evaluate the effectiveness of our proposed algorithm, Robust-HRL, we compared it with heuristic-based methods and existing state-of-the-art RL approaches. We evaluated the performance of the following methods: 1) Slot-based method, 2) Vanilla DQN with 6 discrete actions, 3) Hierarchical Double Deep Q-Learning (hDDQN), and 4) Robust-HRL. Performance of the slot-based method and Robust-HRL was also recorded with and without Gaussian noise. We compared the performance of different approaches using these metrics:

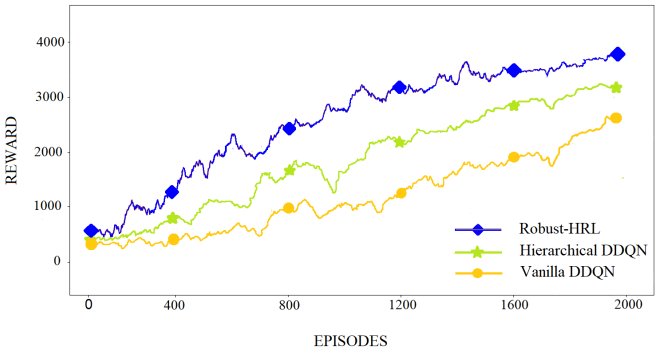

Fig. 6. Training results of different approaches

- **Total Average Reward**: The summation of high-level option reward and the low-level planner choice reward divided by the total number of test episodes;
- **Lane Invasion rate**: The average rate of lane invasion recorded in the test episodes. Lane invasion occurs when the ego-car crosses the ego-lane's boundaries while in the follow lane state;
- **Collision rate**: The percentage of test episodes in which collision occurs;
- **Success rate**: The percentage of test episodes in which the ego-car is able to complete its trajectory from starting point to the end point without collision.

The detailed results are shown in Table I. The training results for RL-based approaches are shown in the Figure 6. For the proposed methodology, we were able to get the optimal policy after 2000 episodes and the policy evaluation after training consisted of 200 test episodes. The Robust-HRL recorded a 97.5% success rate without noisy observations and a 95.5% success rate with a noisy state space.

Direct comparisons with non-hierarchical approaches suggest that the presence of sub-goals through the use of HRL decreased the overall convergence time and also recorded the best success rates. The heuristic-based rules enumeration approach recorded the lowest success rate, as rules, which are specific to particular scenarios and conditions, could not generalize well to interactive and stochastic driving conditions. One of the reasons the slot-based method had low reward was the unnecessary waiting of the ego-car in the ego-lane for lane change. On the other hand, for RL approaches, Table I shows that Vanilla DDQN only achieved a success rate of 83.5% in comparison to the hierarchical

TABLE I

COMPARISON OF RESULTS OBTAINED FROM DIFFERENT PLANNING ALGORITHMS

Method	Gaussian Noise	Total Average Reward	Lane Invasion Rate %	Collision Rate %	Success Rate %
Slot-based with PID	No	2679.40	0	17.5	82.5
Slot-based with PID	Yes	2150.25	0	27.0	73.0
Vanilla DDQN	No	2750.41	19.4	16.5	83.5
hDDQN	No	3117.82	15.9	12.0	88.0
Robust-HRL w/o LSTM	Yes	3531.24	0	9.0	91.0
Robust-HRL	**Yes**	**3728.00**	**0**	**4.5**	**95.5**
Robust-HRL	**No**	**3761.44**	**0**	**2.5**	**97.5**

approaches, which were able to achieve a minimum 88.0% success rate with no noise added to the state space and with direct low-level control. Lastly, the improved version of hDDQn, Robust-HRL, not only significantly improved the convergence time, as shown in Figure 6, but also recorded the lowest collision rate among all approaches.

The results and evaluation of final policies show that using waypoint-based trajectories with a PID Controller instead of direct throttle, steering, and brake ensured smooth tracking and increased safety. As the waypoints were generated in the middle of the road by the CARLA waypoint API, the ego-car recorded almost zero lane invasions while tracking the selected waypoints using the PID controller.

In order to evaluate the performance with incomplete observations, we added Gaussian noise to the state space. We trained both slot-based and Robust-HRL with and without noise. This helped us to make direct comparisons between the two approaches. As noise was added to the slot-based method, performance significantly decreased. For RL-based approaches, the results in Table I show that the collision rate was brought down to 4.5% from 9.0% with the inclusion of the history of state observations and LSTM layers in the network. Although improved, the collision rate was not able to drop significantly after 2000 episodes of training, as LSTMs require more time to train because they are computationally expensive. But overall, adding an LSTM layer to the network yielded best results with Gaussian noise.

VI. CONCLUSION AND FUTURE WORK

In this paper, we improved the existing HRL framework for autonomous driving by proposing Robust-HRL. The proposed structure involves high-level maneuver decision making and low-level waypoint trajectory generation. In order to improve sample efficiency, we used Hybrid Reward Mechanism and Reward-Driven Exploration. Results show that Robust-HRL improves the existing HRL framework for autonomous driving. Future work includes combining policies learned from different scenarios and maneuvers such as intersections and ramp merging through HRL as sub-goals.

ACKNOWLEDGMENT

The authors would like to thank the Robotics Institute Summer Scholars Program (RISS), Carnegie Mellon University, and The Industrial Center, The Hong Kong Polytechnic University, Hong Kong for providing resources for research. I, Kaleb, credit this work to my late father, Naveed Joseph, for his unmatched support throughout RISS.

REFERENCES

[1] C. R. Baker and J. M. Dolan, "Traffic interaction in the urban challenge: Putting boss on its best behavior," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 1752–1758.

[2] D. N. Lee, "A theory of visual control of braking based on information about time-to-collision," *Perception*, vol. 5, no. 4, pp. 437–459, 1976.

[3] P. Cai, Y. Sun, Y. Chen, and M. Liu, "Vision-based trajectory planning via imitation learning for autonomous vehicles," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 2736–2742.

[4] T. Osa, J. Pajarinen, G. Neumann, J. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Foundations and Trends in Robotics*, vol. 7, no. 1-2, pp. 1–179, Mar. 2018.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: MIT Press, 1998.

[6] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Advances in Neural Information Processing Systems 29*, 2016, pp. 3675–3683.

[7] S. LaValle, J. Kuffner, and B. Donaldetal, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and computational robotics: new directions*, 2001, pp. 293–308.

[8] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[9] J. Chen, B. Yuan, and M. Tomizuka, "Deep imitation learning for autonomous driving in generic urban scenarios with enhanced safety," 2019.

[10] Z. Qiao, J. Zhao, Z. Tyree, P. Mudalige, J. Schneider, and J. M. Dolan, "Human driver behavior prediction based on urbanflow," 2019.

[11] E. Galceran, A. G. Cunningham, R. M. Eustice, and E. Olson, "Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction: Theory and experiment," *Auton. Robots*, vol. 41, no. 6, p. 1367–1382, Aug. 2017.

[12] T. G. Dietterich, "The maxq method for hierarchical reinforcement learning." *ICML*, vol. 98, pp. 1–179, 1998.

[13] N. K. Jong and P. Stone, "Hierarchical model-based reinforcement learning: R-max+ maxq," *in Proceedings of the 25th international conference on Machine learning. ACM*, pp. 1–179, 2008.

[14] Z. Qiao, Z. Tyree, P. Mudalige, J. Schneider, and J. Dolan, "Hierarchical reinforcement learning method for autonomous vehicle behavior planning," Nov. 2019.

[15] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, "Tactical decision making for lane changing with deep reinforcement learning," 12 2017.

[16] Y. Zhang, W. Wang, R. Bonatti, D. Maturana, and S. Scherer, "Integrating kinematics and environment context into deep inverse reinforcement learning for predicting off-road vehicle trajectories," *CoRR*, vol. abs/1810.07225, 2018.

[17] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," 2016.

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

[19] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *CoRR*, vol. abs/1507.06527, 2015. [Online]. Available: http://arxiv.org/abs/1507.06527

[20] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "CARLA: an open urban driving simulator," *CoRR*, vol. abs/1711.03938, 2017. [Online]. Available: http://arxiv.org/abs/1711.03938