2018 21st International Conference on Intelligent Transportation Systems (ITSC)
Maui, Hawaii, USA, November 4-7, 2018

POMDP and Hierarchical Options MDP with Continuous Actions for Autonomous Driving at Intersections

Zhiqian Qiao[1], Katharina Muelling[2], John Dolan[1,2], Praveen Palanisamy[3] and Priyantha Mudalige[3]

***Abstract*— When applying autonomous driving technology to real-world scenarios, environmental uncertainties make the development of decision-making algorithms difficult. Modeling the problem as a Partially Observable Markov Decision Process (POMDP) [1] allows the algorithm to consider these uncertainties in the decision process, which makes it more robust to real sensor characteristics. However, solving the POMDP with reinforcement learning (RL) [2] often requires storing a large number of observations. Furthermore, for continuous action spaces, the system is computationally inefficient. This paper addresses these problems by proposing to model the problem as an MDP and learn a policy with RL using hierarchical options (HOMDP). The suggested algorithm can store the state-action pairs and only uses current observations to solve a POMDP problem. We compare the results of to the time-to-collision method [3] and the proposed POMDP-with-LSTM method. Our results show that the HOMDP approach is able to improve the performance of the agent for a four-way intersection task with two-way stop signs. The HOMDP method can generate both higher-level discrete options and lower-level continuous actions with only the observations of the current step.**

I. INTRODUCTION

Traversing a four-way intersection with two-way stop signs can be difficult for autonomous vehicles. Upon arrival, the agent has to time its actions properly to make a turn onto the right-of-way road safely. If the car enters the intersection too soon, it can result in a collision or cause the approaching right-of-way vehicles to brake hard. On the other hand, if the driver keeps waiting for too long to make sure the situation is safe, valuable time is lost.

On a first glance, rule-based methods [4][5] seem to be a plausible approach to describe the human decision process for intersection traversal. The driver estimates the time an approaching vehicle needs to reach and traverse the intersection. If the traversal time is less than the approaching time, then the driver may make the decision to begin the traversal. Otherwise, the driver may choose to wait. For an autonomous vehicle, an algorithm called time-to-collision (TTC) [3] takes a similar approach. However, estimating the time accurately and dealing with unexpected situations by adjusting the ego vehicle's decision according to changes in the environment is hard for autonomous vehicles. As a result, such an approach is not reliable enough for the autonomous vehicle. An alternative to using rule-based methods is to model the problem as a Markov Decision Process (MDP) and to learn an optimal policy using reinforcement learning.

[1] Department of Electrical and Computer Engineering, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, USA. zhiqianq@andrew.cmu.edu
[2] The Robotics Institute, Carnegie Mellon University, Pittsburgh, USA.
[3] Research & Development, General Motors, Warren, MI, USA.

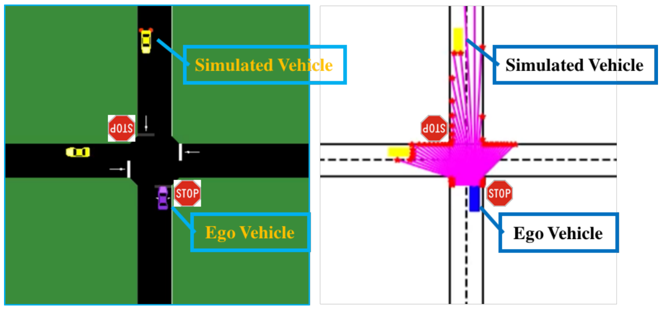

Fig. 1: The proposed intersection traversal scenario with simulated ray traces. The ego vehicle starts from the stop line with zero velocity and other simulated vehicles run in other lanes using the Krauss Traffic Model.

Some previous work [7][8] on the intersection traversal problem uses bird's-eye view to construct the state information for decision making, which is hard to obtain in a real vehicle. In this work, we only use the information coming from resources available to an autonomous vehicle as input, such as a simulated LIDAR and the ego car's state information. The challenge is that the LIDAR ray trace information does not include everything happening near the intersection, but only the information within the visibility range of the ego vehicle. In the paper, we assume that the intersection traversal problem is an MDP; however, the ego vehicle cannot directly observe everything which is happening out of the visibility range due to the geometry of the intersection. For any driving scenario, it is hard for the ego car to make proper decisions based only on the current states in the surrounding environment. Meanwhile, the agent cannot access the historical information of other vehicles in order to make a proper decision. Using POMDP to model the whole system can maintain a probability distribution for the agent's observation. As a result, our first contribution is to model the problem as a POMDP and use deep reinforcement learning (DRL) to optimize the policy for generating continuous actions. The second contribution is proposing Hierarchical Options for MDP (HOMDP) to solve the problem. Instead of modeling the problem as a POMDP, we still apply the MDP model and solve the problem by adding hierarchical options, which gives better results than POMDP alone.

II. RELATED WORK

This section introduces previous work related to this paper, which can be categorized as follows: 1) papers that address deep reinforcement learning (DRL) algorithms; 2) work focused on using DRL to solve the POMDP problem; 3) papers which use hierarchical structure and the concept of options in combination with DRL methods.

A. Deep Reinforcement Learning

Recently, various DRL algorithms have shown advantages in handling high-dimensional sensory inputs with discrete or continuous actions. Deep Q-Networks [9] (DQN) and its variants have been successfully applied to various fields. However, DQN is restricted to the discrete action domain, which makes it hard to apply to autonomous driving. DDPG [10], which was proposed by Lillicrap et al., adapted the ideas underlying the success of DQN to the continuous action domain. All these solutions have in common that they model the problem as an MDP, i.e., they assume full knowledge of the environment, which is often not true in the real world.

B. POMDP with Reinforcement Learning

[11] proposed to extend the DQN method for POMDP problems by learning the optimal policy of a model-based RL problem. They used a fully-connected network to approximate the mapping function from observations and belief vectors to Q-values for different actions in order to solve a POMDP problem through DQN. [12] proposed a method called Deep Recurrent Q-network (DRQN) which can be applied to a model-free representation by adding LSTM layers in the policy network. The trained policy network is capable of capturing all the historical information in the LSTM layer and the output actions are based on all the historical observations with the help of LSTM layers. [7] used DRQN with states from a bird's-eye view of the intersection to learn the policy for traversing the intersection. Based on DRQN, [13] proposed an algorithm called Deep Distributed Recurrent Q-networks (DDRQN) which not only used the previous observations but also previous actions as the inputs. The results show the first success in learning communication protocols by RL. [14] is a recent method called Action-specific Deep Recurrent Q-Network (ADRQN), which also used historical observations and actions as input to get an optimal policy. However, instead of inputting all historical information into the LSTM layer, they used paired last step action and current step observations as inputs into the LSTM layer for training, which showed improved results on some POMDP tasks.

C. Hierarchical Reinforcement Learning and Options

Based on the context of RL, [15] proposed the idea of options to generate actions at different levels. The options are used to define a policy governing when the action policy is initialized and terminated. [16] introduced the concept of hierarchical Q learning called MAXQ. They proved the convergence of MAXQ mathematically and showed faster computing power than the original Q learning experimentally. [17] proposed an improved MAXQ method by combining the R-MAX [18] algorithm with MAXQ. It has both the efficient model-based exploration of R-MAX and the opportunities for abstraction provided by the MAXQ framework.[19] presented hierarchical DQN (h-DQN), a method which is based on the structure of DQN. H-DQN adds another high-level Q-value function to learn a high-level policy while the DQN is responsible for the low-level policy. The algorithm is effective on the sparse and delayed feedback of ATARI games. Instead of using observations in the memory, [20] proposed a memoryless options method to solve the POMDP problem by RL. They compared their results with finite state controllers (FSC) [21], which is a state-of-the-art approach that is commonly used to solve the POMDP problem. The results showed that the method can be both as expressive as FSC and more efficient than a recurrent neural network when learning optimal policies in challenging POMDPs.

In this paper, we firstly extend the original POMDP with LSTM [7] structure to an application that can generate continuous actions instead of discrete actions. Second, we propose a structure which combines HMDP and DRL and apply the new structure to the autonomous vehicle traversing intersection problem.

III. Preliminaries

This work is based mainly on the ideas of Markov Decision Processes (MDPs), Hierarchical MDP (HMDP) and Partially Observable MDP (POMDP). For a better understanding of the proposed methodology, we will give a short overview of all three frameworks before explaining our approach in Section IV.

A. Markov Decision Process

A MDP is defined as a tuple $\{S,A,R,T,\gamma\}$ in which S denotes a set of states, A defines the set of available actions, and $T(s_{t+1},a_t,s_t)$ is a transition function that maps a state-action pair (s_t,a_t) to a new state s_{t+1}. The reward function R defines the immediate rewards for each state-action pair and γ is a discount factor for long-term rewards. At every time step t, the agent receives the state s_t from the environment and then selects an action a_t according to the policy π, which results in the agent being transitioned to a new state s_{t+1} according to the transition function T. Meanwhile the agent will receive an immediate reward r_t after taking action a_t in state s_t.

B. Hierarchical MDP

Hierarchical MDP (HMDP) is a general framework to solve problems with large state and action spaces. The framework can restrict the space of policies by separating the action policy into different levels. One way of realizing an HMDP is via the options framework [22]. An HMDP includes a set of options O. An option $o \in O$ is a tuple of $\{\pi_o, I_o, \beta_o\}$. π_o is the policy for the option o. $I_o \in S$ is the set of states in which an option can be initiated. $\beta_o(s_t)$ defines the probability that the options will be terminated in state s_t.

C. Partially Observable MDP

When applying decision processes to real-world problems, we run into the problem that we often can only infer the state of the environment from potentially incomplete and noisy sensor data, i.e., we have a partially observable environment. POMDPs [1] provide a powerful tool to model such

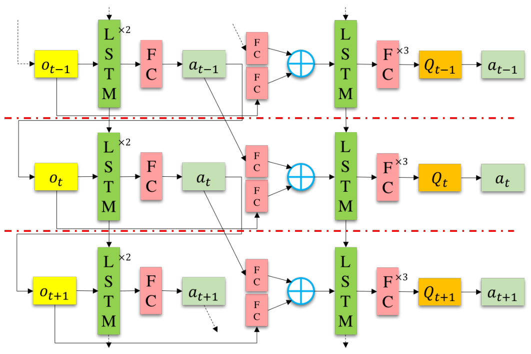

Fig. 2: POMDP with LSTM network for generating continuous actions. The input observation o_t is a 126-D vector which generates a 3-D continuous action vector a_t through two 512-D LSTM layers and one fully connected (FC) layer. Meanwhile, the observation vector followed by an FC layer is concatenated with the action vector coming from the previous step followed by an FC, and then followed by a 128-D LSTM layer and three FC layers to produce a 1-D Q-values Q_t corresponding to the continuous action vector.

partially observable processes. A POMDP is defined as a tuple $\{S, A, R, T, \Omega, V, \gamma\}$ which extends the MDP formulation through the definition of a set of environment observations Ω and a set of conditional observation probabilities $V(v_{t+1}|s_{t+1}, a_t)$.

IV. Methodology

In this paper, we model the intersection problem as a reinforcement learning problem. In particular, we consider the scenario of a four-way intersection with two-way stop signs (see Figure 1). We assume a start configuration in which the ego-vehicle (purple rectangle in Figure 1) has already stopped near the stop line and would like to traverse the intersection in order to reach a pre-defined destination beyond the intersection in order to complete Turn Right, Go Straight or Turn Left tasks. The yellow rectangles are simulated vehicles generated by the SUMO simulator [23] and are all moving according to the Krauss Car Following Traffic Model [24].

In this section, we propose two methods to solve the RL problem. The first method models the problem as a POMDP and uses the past observation and action pairs as inputs for the reinforcement learning algorithm. It generates continuous actions as outputs at each time step. The second method models the problem as an MDP with hierarchical options, which only takes the latest observations as input and then generates discrete high-level options as well as low-level actions through the option policy π_o simultaneously.

A. POMDP with LSTM

In our work, we propose a network based on the ideas of DDPG [10] and ADRQN [14] to generate continuous actions based on the observations coming from previous steps. In the network, instead of using all the previous observations, only a fixed number of previous steps $n_{steps} = 20$ are used as the inputs to the LSTM layer. Figure 2 shows the structure

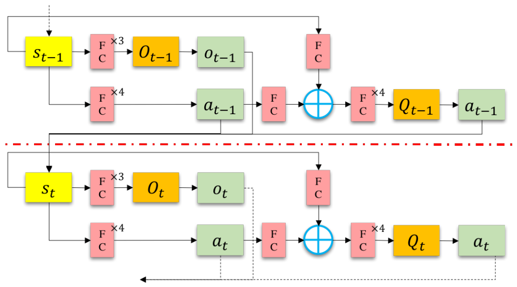

Fig. 3: MDP with Hierarchical Options network for generating continuous actions. The 126-D input state vector s_t is followed by three FC layers in order to generate a 2-D Q-values O_t corresponding to two hierarchical option candidates. Meanwhile, the input state vector produces a 2-D continuous action vector a_t through four FC layers. Then, the state vector followed by an FC layer is concatenated with the action vector followed by one FC layer. The output produces a 1-D Q-values Q_t which corresponds to the action vector through four FC layers.

of the proposed network. The input of the network is the observation at each time step, and through the first LSTM-layer and a FC-layer, the network gets the information coming from previous steps to get the action output for each step. After that, we pair the observation coming from the current step with the action taken for the last step (o_t, a_{t-1}) to form the input for generating the Q-value Q_t.

B. MDP with Hierarchical Options

For MDP with hierarchical options, we use three networks to produce hierarchical options, low-level actions, and Q-values. The structure of the network is shown in Figure 3. Unlike the POMDP network, the system does not take observations from previous time-steps into account for the decision process. However, it uses hierarchical options to make a decision on whether the ego vehicle can trust the environment or not. Then, based on the observations and high-level decision, the agent makes the decision on the low-level policies. The lower-level decision here means the acceleration or deceleration the agent takes at the current step. The overall flow of the algorithm is given in Algorithm 1.

C. State Space

In the representation of the state space, we assume that the map information is known by the agent. We define the state space as containing the ego vehicle's velocity v and road geometry information, including the distance between the ego vehicle and the lower intersection boundary d_{lb}, the mid-point d_{mp} and the destination d_{goal} of the intersection. Figure 4 shows the geometric information related to the intersection and the generated/simulated rays. The **video**[1] shows the simulated ray trace representations during Go Straight, Turn Right and Turn Left at the intersection in SUMO [23].

At each time step, a square of size $100m \times 100m$ is constructed whose center is the middle front of the ego

[1]https://youtu.be/5HMB8SWanW4

Algorithm 1 MDP with Hierarchical Options via RL

```
1: procedure HOMDP
2:     Construct two empty replay buffers $\mathbf{B}_a$ and $\mathbf{B}_o$.
3:     Randomly initialize actor network $NN^a$, critic network $NN^Q$ and option network $NN^O$ with weights $\theta^\mu$, $\theta^Q$ and $\theta^O$ and the corresponding target actor network $NN^{\mu'}$, critic network $NN^{Q'}$ and option network $NN^{O'}$ with weights $\theta^{\mu'}$, $\theta^{Q'}$ and $\theta^{O'}$.
4:     for $e \leftarrow 1$ to $E$ epochs do
5:         Get initial state $s_0$. Initial option is $o_0 = SlowForward$. $r_o = 0$.
6:         for $t \leftarrow 1$ to $T$ time steps do
7:             $o_t$, $a_t$ = GetAction($s_t, o_{t-1}$)
8:             $s_{t+1}, r_t, done =$ StepForward($s_t, o_t, a_t$)
9:             if $o_t$ is $Forward$ then
10:                $r_o += r_t$ and add $(s_t, o_t, r_t, s_{t+1}, done)$ to $\mathbf{B}_o$.
11:                if $done$ then
12:                    Add $(s_t, o_t, r_o, s_{t+1}, done)$ to $\mathbf{B}_o$.
13:            else
14:                Add $(s_t, o_t, r_t, s_{t+1}, done)$ to $\mathbf{B}_o$.
15:            Sample random mini-batch of $M$ transitions $(s_i, o_i, r_i, s_{i+1})$ from $\mathbf{B}_o$ and $(s_j, a_j, r_j, s_{j+1})$ from $\mathbf{B}_a$.
16:            $o_{i+1} = \arg\max_o O'(s_{i+1}|\theta^{O'})$. $y_i^o = r_i + \gamma O'(s_{i+1}|\theta^{O'})$.
17:            Minimize $L^o = \frac{1}{M}\sum_i y_i^o - O(s_i|\theta^O)$ to update $NN^O$.
18:            $y_j^\mu = r_j + \gamma Q'(s_{j+1}, a_{j+1}|\theta^{Q'})$
19:            Minimize $L^\mu = \frac{1}{M}\sum_j y_j^\mu - Q(s_i|\theta^Q)$ to update $NN^Q$.
20:            $\frac{1}{M}\sum_j \nabla_{\mu(s_j)} Q(s_j, \mu(s_j)|\theta^Q) \nabla_{\theta^\mu} \mu(s_j|\theta^\mu)$ is the policy gradient and is used to update actor network.
21:            Update the target networks: $\theta^{z'} \leftarrow \tau\theta^z + (1-\tau)\theta^{z'}$ for $z$ in $\{\mu, Q, O\}$.
```

Algorithm 2 Get Action

```
1: procedure GETACTION($s$, $o$)
2:     if $o$ is SlowForward then
3:         $o \leftarrow \arg\max_o O'(s|\theta^{O'})$ according to $\varepsilon$ greedy.
4:         $a = 0$.
5:     if $o$ is Forward then
6:         $a = \mu(s|\theta^\mu) + \mathcal{N}$ where $\mathcal{N}$ is a random process.
```

Algorithm 3 Move one Step to Forward

```
1: procedure STEPFORWARD($s$, $o$, $a$)
2:     if $o$ is SlowForward then
3:         Slowly move forward with $d$ meters and stop.
4:     else if Any $\frac{l_i}{w_i} \leq a$ then
5:         Decrease speed of ego car with deceleration $c_d$.
6:     else
7:         Increase velocity of ego car with acceleration $c_a$.
```

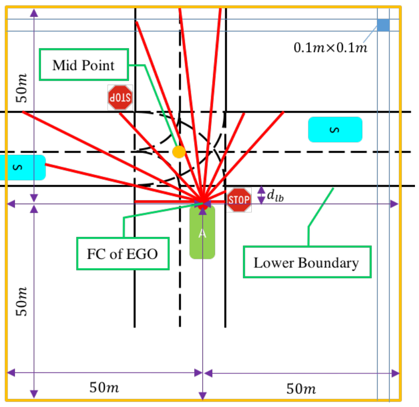

Fig. 4: Explanation of state space and method of constructing ray trace system

vehicle. The square is divided into a million small squares with size $0.1m \times 0.1m$. Each small square is occupied if there is any obstacle or moving object in this area. There are 61 ray traces spanning π radians produced from the middle front of the ego vehicle that cover the front view of the vehicle. Each ray has a resolution of 0.5 meter and has a maximum reach of 50 meters. Each ray is emitted from the front center of the ego vehicle and if it reaches any obstacle like the road boundary or a moving vehicle, the corresponding distance is sensed. Meanwhile, we assume that the velocity of the end-point of each ray can be determined as well. To sum up, the agent obtains the distance and the velocity at the end point of each ray trace, which results in a 126-D vector:

$$s_t = \begin{bmatrix} v, & d_{lb}, & d_{mp}, & d_{goal}, & l_i, & c_i \end{bmatrix} \tag{1}$$

in which $i \in [0, 60]$ and l_i and c_i are respectively the length and the end point's velocity for each ray trace i. v is the velocity of the ego vehicle at the current time step t.

D. Reward Function

The reward function is used to evaluate the goodness of the action taken at every time step based on the current states or observations from the environment. For the proposed problem, the reward function is designed to be the sum of the following terms:

$$r_t = r_{ego} + r_{ray} + \sigma_5 + \sigma_6 + \sigma_7 \tag{2}$$

- For the ego vehicle, a positive reward is assigned that reflects the progress towards the destination position. A negative constant is added at each time step to encourage the ego-vehicle to make fast progress. The initial position and destination of the ego vehicle are fixed for each training iteration. Equation 3 represents the reward term for the ego-vehicle in which p_{des}, p_{ego} and p_{ini} are the destination position, current position and initial position, respectively:

$$r_{ego} = \sigma_1 \frac{\|p_{des} - p_{ego}\|^2}{\|p_{des} - p_{init}\|^2} - \sigma_2 \tag{3}$$

  where σ_1 and σ_2 are constants.
- To penalize the agent getting too close to other traffic participants and obstacles, a negative reward is imposed if the ego vehicle interacts with other simulated vehicles within the visibility. An interaction occurs when the ego vehicle can see any approaching vehicle which has the right of way through the simulated ray trace, and the ego vehicle is blocking the future path of any approaching vehicles (has already passed the lower boundary of the intersection). The penalty is:

$$r_{ray} = -\sum_i \frac{\sigma_3}{l_i} \mathbf{1}\left\{\frac{l_i}{v_i} \leq \sigma_4\right\} \tag{4}$$

  in which $\mathbf{1}(x) = 1$ if $x \geq 0$ and $\mathbf{1}(x) = 0$ if $x < 0$.
- A constant penalty σ_5 is imposed in the case of a crash between the ego vehicle and any simulated vehicles:

$$\sigma_5 = -10000 \text{ if } \|p_{sim}^i - p_{ego}\|^2 = 0, \tag{5}$$

  where p_{sim}^i is the position of the i^{th} simulated vehicle within the visibility of the ego vehicle.

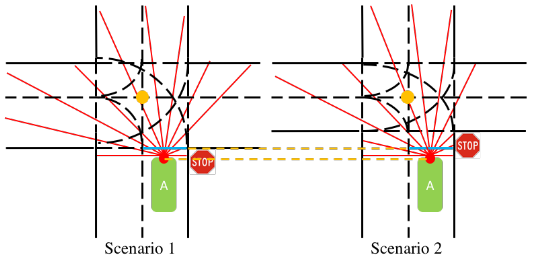

Fig. 5: Two initial scenarios for experiments

- A constant penalty σ_6 is added if the task cannot be finished within 1000 steps, which is 100 seconds in the real world.
- Finally, a positive reward σ_7 is added if the ego vehicle reaches the goal:

$$\sigma_7 = 1000 \text{ if } \|p_{des} - p_{ego}\|^2 = 0. \tag{6}$$

V. Experiments

We tested our algorithm in the SUMO simulator [23] with two scenarios which are shown in Figure 5. In the two scenarios, the initial positions of the ego vehicle, the position of the stop-line, the position of the mid-point of the intersection and the position of the destination are all the same. However, we change the position of the lower-bound of the intersection, which may result in less information from the ego vehicle's ray traces, especially at the beginning of each training epoch. This kind of situation is closer to real-world scenarios, in which the average distance between the lower bound of the intersection and the stop-line is four to six meters. As a result, most vehicles don't roll all the way up to the edge of the intersection. The farther the stop-line is away from the lower bound of the intersection, the more difficult it is to make a proper decision on when to traverse, because the geometry of the intersection blocks visibility.

During the traverse process, the vehicle has to either Go Straight, Turn Right or Turn Left successfully while limiting the required interactions between the ego vehicle and other simulated vehicles. To evaluate the effectiveness of the proposed algorithms, we tested the following four methods: 1) time-to-collision (TTC) [3], 2) DDPG [10], 3) our proposed POMDP with LSTM algorithm and 4) Our proposed HOMDP with vanilla DRL for continuous actions. For the evaluation we applied the resuliting policies for 1000 iterations on the Go Straight, Turn Right and Turn Left tasks separately. We compared the different algorithms using five metrics:

- **Percentage of Success**: the percentage of epochs in which the ego vehicle successfully reaches the destination.
- **Percentage of Collision**: the percentage of epochs in which the ego vehicle collides with any other simulated vehicle before reaching the destination.
- **Percentage of Unfinished cases**: the percentage of epochs in which the ego vehicle does not achieve the goal within 1000 simulation steps, which equals 100 seconds in the real world.

TABLE I: Results for Scenarios One and Two with TTC and DDPG methods

Task	Metrics	Scenario One		Scenario Two	
		TTC	DDPG	TTC	DDPG
Go Straight	% Success	95.6	97.8	95.3	96.2
	% Collision	2.6	2.2	4.7	3.1
	% Unfinished	1.8	0.0	0.0	0.7
	Total Steps	385	136	70	195
	% Interaction	44.82	24.49	18.87	30.02
	Reward	673	735	552	587
Turn Right	% Success	98.6	100.0	98.2	97.6
	% Collision	1.4	0.0	1.8	2.4
	% Unfinished	0.0	0.0	0.0	0.0
	Total Steps	117	106	66	63
	% Interaction	17.05	10.14	13.97	11.4
	Reward	743	792	665	721
Turn Left	% Success	86.6	97.6	91.42	92. 4
	% Collision	1.0	2.0	8.58	5.4
	% Unfinished	12.38	0.4	0.0	2.2
	Total Steps	514	196	108	289
	% Interaction	35.47	33.26	39.57	37.31
	Reward	-821	217	-437	-223

- **Total steps** $step_{total}$: the average number of total steps the ego vehicle takes to finish each test iteration including all the successful and fail cases. Fewer steps means less time is taken to finish the whole process.
- **Percentage of Interaction** $\frac{step_{inter}}{step_{total}}$: the average percentage of steps in each test iteration in which the ego vehicle has an interaction with one of the other traffic participants. An interaction is defined as an unscheduled behavior change of a vehicle in response to the ego vehicle's behavior. Other traffic participants can *react* to the ego vehicle if the ego vehicle has passed the lower bound of the intersection and is within the visibility range of the ego vehicle.
- **Total Reward** $\sum_{t=0}^{T} r_t$: the reward the agent gets for each test iteration. More interactions means that the ego vehicle relies more on other vehicles' decelerations to avoid collision.

VI. Results

First, we compare the results for our two baselines (i.e., TTC and the vanilla DDPG) for both scenarios (Figure 5). The results are shown in Table I . In Scenario One, DDPG outperforms TTC in terms of the success rate, total steps to finish the traversing, and the interaction rate, which is in alignment with other reported results [7] that use bird's-eye view data instead of ray trace data. However, for Scenario Two, where there are more unobservable states initially and during the training process, DDPG and TTC vary as to which has better performance. Although DDPG has a higher success rate for Go Straight and Left Turn scenarios, it also has a higher interaction rate, which means the lower collision rate is due at least in part to the other vehicles' adjustments instead of the ego vehicle's skill.

Next, we compared POMDP with LSTM and HOMDP to get the results for Scenario Two alone, which are presented in Table II. The performance of the final policies for POMDP and HOMDP can be seen in the **video**[2]. In general, TTC

[2]https://youtu.be/Rj5bYtKshh8

TABLE II: Results for Scenario Two with All Four Methods

Task	Metrics	Scenario Two			
		TTC	DDPG	POMDP	HOMDP
Go Straight	% Success	95.3	96.2	97.1	**98.3**
	% Collision	4.7	3.1	**1.7**	**1.7**
	% Unfinished	**0.0**	0.7	1.2	**0.0**
	Total Steps	**70**	195	261	102
	% Interaction	18.87	30.02	27.86	**26.41**
	Reward	552	587	621	**873**
Turn Right	% Success	98.2	97.6	99.5	**99.8**
	% Collision	1.8	2.4	0.5	**0.2**
	% Unfinished	0.0	0.0	0.0	0.0
	Total Steps	66	63	**57**	62
	% Interaction	13.97	11.4	**6.26**	9.28
	Reward	665	721	892	**903**
Turn Left	% Success	91.4	92.4	95.6	**97.3**
	% Collision	8.6	5.4	**2.4**	2.6
	% Unfinished	**0.0**	2.2	2.0	0.1
	Total Steps	**108**	289	276	132
	% Interaction	39.57	37.31	33.21	**28.90**
	Reward	-437	-223	213	**632**

can cause a higher collision rate and may result in more interactions, which means the method relies on the other simulated vehicles to yield to the ego vehicle in order to avoid collision, which is not similar to the situation in the real world. The result from POMDP is more conservative, which causes more unfinished fail cases, meaning the epochs in which the ego vehicle cannot finish the traversing task within 1000 simulation steps. The HOMDP method, which is our main contribution, achieves the highest total reward and success rate for all the Go Straight, Turn Right and Turn Left tasks. For the Go Straight and Turn Left tasks, the interaction rate is lowest with HOMDP which means the results rely least on the other vehicles' yielding. HOMDP take more total steps than other methods (TTC for Go Straight and Turn Left task, POMDP for Turn Right Task), which means HOMDP is a more conservative strategy which values the success rate most.

VII. Conclusion and Future Work

This paper proposes two methods to deal with the problem of an autonomous vehicle traversing an urban partially non-controlled stop-sign intersection. The first approach is based on the ideas of DDPG and ADRQN, which use previous steps' observations to learn an optimal policy with respect to the current observations by using an LSTM layer in the DRL method. The second method is based on the idea of hierarchical MDPs, which learn a discrete option in the high-level learning process and continuous low-level actions simultaneously. The results shows that HOMDP can provide better performance for both success rate and interaction rate.

For future work, we want to apply our model to real traffic data instead of the simulated traffic model in SUMO. As a starting point, we will apply the model on the Peachtree Dataset in NGSIM [25].

References

[1] G. E. Monahan, "State of the arta survey of partially observable markov decision processes: theory, models, and algorithms," *Management Science*, vol. 28, no. 1, pp. 1–16, 1982.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[3] 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.

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

[5] C. Dong, Y. Zhang, and J. M. Dolan, "Lane-change social behavior generator for autonomous driving car by non-parametric regression in reproducing kernel hilbert space," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 4489–4494.

[6] D. Isele and A. Cosgun, "Transferring autonomous driving knowledge on simulated and real intersections," *arXiv preprint arXiv:1712.01106*, 2017.

[7] D. Isele, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating intersections with autonomous vehicles using deep reinforcement learning," *arXiv preprint arXiv:1705.01196*, 2017.

[8] W. Song, G. Xiong, and H. Chen, "Intention-aware autonomous driving decision-making in an uncontrolled intersection," *Mathematical Problems in Engineering*, vol. 2016, 2016.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[11] M. Egorov, "Deep reinforcement learning with pomdps," 2015.

[12] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *CoRR, abs/1507.06527*, 2015.

[13] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate to solve riddles with deep distributed recurrent q-networks," *arXiv preprint arXiv:1602.02672*, 2016.

[14] P. Zhu, X. Li, P. Poupart, and G. Miao, "On improving deep reinforcement learning for pomdps," *arXiv preprint arXiv:1804.06309*, 2018.

[15] S. P. Singh, "Transfer of learning by composing solutions of elemental sequential tasks," *Machine Learning*, vol. 8, no. 3-4, pp. 323–339, 1992.

[16] T. G. Dietterich, "The maxq method for hierarchical reinforcement learning." in *ICML*, vol. 98. Citeseer, 1998, pp. 118–126.

[17] 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, 2008, pp. 432–439.

[18] R. I. Brafman and M. Tennenholtz, "R-max-a general polynomial time algorithm for near-optimal reinforcement learning," *Journal of Machine Learning Research*, vol. 3, no. Oct, pp. 213–231, 2002.

[19] 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*, 2016, pp. 3675–3683.

[20] D. Steckelmacher, D. M. Roijers, A. Harutyunyan, P. Vrancx, and A. Nowé, "Reinforcement learning in pomdps with memory-less options and option-observation initiation sets," *arXiv preprint arXiv:1708.06551*, 2017.

[21] P. Poupart and C. Boutilier, "Bounded finite state controllers," in *Advances in neural information processing systems*, 2004, pp. 823–830.

[22] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier, "Hierarchical solution of markov decision processes using macro-actions," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1998, pp. 220–229.

[23] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of sumo-simulation of urban mobility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, 2012.

[24] J. Song, Y. Wu, Z. Xu, and X. Lin, "Research on car-following model based on sumo," in *Advanced Infocomm Technology (ICAIT), 2014 IEEE 7th International Conference on*. IEEE, 2014, pp. 47–55.

[25] "NGSIM homepage. FHWA." 2005-2006. [Online]. Available: http://ngsim.fhwa.dot.gov.