Safe Handling of Intersection Events under other agents' Latent Driving styles (SHIELD)

Joonwon Kang[1], Tae Yang[2]

Abstract— **This work investigates how to tackle the problem of an ego vehicle navigating an uncontrolled four-way intersection where each road user possesses a latent driving style. Initially, the problem is formulated as a fully observable Markov Decision Process (MDP), assuming the driving styles of other road users are known. Two approaches are explored: solving a combined MDP involving the ego vehicle and a subset of nearby road users, and solving individual MDPs for each road user and the ego vehicle. The study is then extended to a more realistic setting where driving styles are unknown, reformulating the problem as a Partially Observable Markov Decision Process (POMDP). To address this, we compare two methods: directly solving the problem using a Deep Recurrent Q-Network (DRQN) and estimating latent driving styles using an LSTM model to use pre-trained Q-values of MDPs again. The results demonstrate how knowledge and inference of driving styles and interactions between vehicles influence decision-making and safety in autonomous navigation.**

I. Introduction

The increasing adoption of autonomous vehicles (AVs) presents significant challenges for ensuring safety and efficiency in complex traffic scenarios [1]. Among these, navigating uncontrolled four-way intersections poses a critical problem. Such intersections lack explicit signaling, requiring AVs to make real-time decisions while interacting with other road users. Complicating matters further, road users often exhibit diverse and latent driving styles, ranging from aggressive to cautious, which are typically unknown to the ego vehicle. Failure to account for these variations can lead to suboptimal decision-making, potentially resulting in delays or collisions. Consequently, some articles demonstrated the benefits of understanding these latent states [2] and outlined methods to estimate them [3], [4].

This paper explores the problem of autonomous decision-making in the context of uncontrolled four-way intersections, focusing on how latent driving styles of road users affect the ability of the ego vehicle to make safe and effective decisions. The study starts by formulating the problem as a fully observable Markov Decision Process (MDP), where the ego vehicle has full knowledge of the driving styles of other road users. Two methods are explored within the MDP framework: (1) solving a combined MDP for the ego vehicle and a set number of road users, and (2) solving individual MDPs for the ego vehicle and each road user. Both methods are evaluated to assess the impact of considering the interaction between other vehicles explicitly.

[1]J. Kang is with the Department of Mechanical Engineering, Stanford University, CA, USA urgemini@stanford.edu

[2]T. Yang is with the Department of Computer Science, Stanford University, CA, USA taeyang@stanford.edu

Next, we extend the problem to a more realistic setting by introducing a Partially Observable Markov Decision Process (POMDP), where the ego vehicle is unaware of the other road users' driving styles. Instead, the ego vehicle must infer these latent behaviors from observable states, which introduces a new layer of complexity. We propose two methods to tackle the POMDP: (1) a Deep Recurrent Q-Network (DRQN) that operates directly under partial observability without any prior knowledge of driving styles, and (2) an LSTM-based latent state estimation model that estimates hidden driving styles while using previously learned Q-values.

The primary contributions of this paper include a comparative study of two MDP-based decision-making approaches under full observability, the extension of the decision-making problem to a POMDP framework to handle hidden driving styles, and a comprehensive analysis comparing MDP and POMDP solutions in the context of intersection navigation with latent driving styles. These contributions aim to advance the development of decision-making frameworks that enhance the safety and efficiency of AVs in complex, partially observable traffic scenarios.

II. Problem Formulation

A. Simulation Environment Setup

The simulation environment used in this study models an uncontrolled four-way intersection in SUMO (Simulation of Urban MObility). The intersection consists of four incoming and four outgoing roads, each with a speed limit of $40\,\mathrm{km/h}$. The roads are configured with multiple lanes, and traffic conditions vary based on the behaviors of individual road users.

In this setup, road users are generated randomly with assigned driving styles determined by an "impatience" parameter [5]. This impatience parameter, which can be either 1 (*impatient*) or 0 (*patient*), governs their driving behavior:

- **Impatient Vehicles:** Accelerate more aggressively and make rapid decisions when navigating through the intersection.
- **Patient Vehicles:** Drive more conservatively, with slower acceleration and cautious decision-making.

These behavioral differences introduce dynamic uncertainty into the environment, requiring the ego vehicle to adapt and react appropriately to the diverse driving styles of surrounding road users.

The simulation is configured to progress in discrete time steps of $0.1\,\mathrm{s}$. During each step, the ego vehicle's acceleration is controlled programmatically through an external interface that interacts with the SUMO simulation.

TABLE I

COMPONENTS OF THE MDP FORMULATION

Component	Description
S: State Space	*Ego Variables*: Simulation step, position (x,y), speed, acceleration. *Road User Variables*: For each road user, position (x,y), speed, acceleration, impatience parameter.
A: Action Space	Discrete acceleration levels available to the ego vehicle: $[-3,-2,-1,0,1,2,3]$.
T: Transition Model	SUMO simulator runs the environment. Our model-free approach learns from state transitions and rewards.
R: Reward Function	Rewards safe and efficient navigation; penalizes collisions, speed-limit violations, and excessive delays.
γ: Discount Factor	Balances near-term and future rewards; $\gamma = 0.95$.

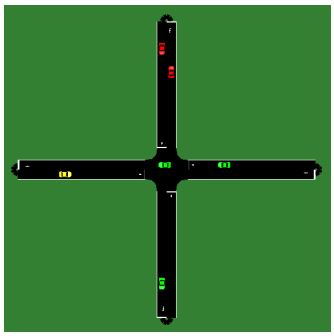

Fig. 1. Illustration of the SUMO simulation environment. The ego vehicle navigates through the intersection, interacting with road users of varying impatience levels.

Figure 1 provides a visual representation of the simulation environment. The ego vehicle, highlighted in yellow, starts from the west and aims to safely navigate through the intersection to its destination. Surrounding road users are color-coded based on their impatience levels (Green: Patient vehicles, Red:Impatient vehicles).

The simulation framework, with its stochastic generation of road users and real-time interaction, provides a robust platform for testing and refining reinforcement learning policies.

B. MDP

The ego vehicle's navigation through an intersection is modeled as a Markov Decision Process (MDP), defined by the tuple (S,A,P,R,γ), where the components are outlined in Table I.

The reward function is defined to encourage safe and efficient navigation through the intersection. Table II outlines the key components of the reward function.

TABLE II

REWARD FUNCTION DETAILS

Event	Reward/Penalty
Collision with another vehicle	-100
Inducing strong decel of another vehicle	-10
Successfully passing the intersection	$+10$
Speed exceeds the limit (40 km/h)	Proportional penalty $(-0.1 \cdot (v - v_{\max}))$
Stopping far from the stop line	-1
Time consumed to pass the intersection	$-0.5 \cdot$ simulation_step

The SUMO simulator employs a complex, deterministic traffic simulation model that incorporates stochasticity in road user behavior. The next actions of each vehicle are heavily influenced by interactions with other vehicles, introducing significant uncertainty into state transitions.

A straightforward solution for this issue might be a neural network that can estimate the transition probability for the next state. However, the state space is still too large to expect a neural network to give the right transition probabilities. Also, regarding our limited time window, it would take too long to do both transition neural network training and policy iterations.

Therefore, this MDP formulation is solved using a **model-free approach**, especially Deep Q-Learning (DQN), which utilizes the neural network to learn Q-value function directly from state transitions and rewards without requiring knowledge of $T(s'|s,a)$ [6].

This general setup is utilized across two distinct approaches:

- **Combined MDP:** A single MDP includes the ego vehicle and a fixed number (N) of nearby road users, creating a high-dimensional state space.
- **Independent MDPs:** Separate MDPs are solved for each road user, where the state space only includes the ego vehicle and one road user at a time. This reduces state dimensionality at the cost of independent policy solutions.

C. POMDP

Building on the previous MDP formulation, the navigation problem is extended to a Partially Observable Markov Decision Process (POMDP) to address uncertainty in road user behavior. Specifically, the impatience level of road users, which influences their driving behavior, is treated as a hidden state that cannot be directly observed.

In this framework, the ego vehicle seeks to infer hidden states from observable variables, such as position, speed, and acceleration, while making decisions to optimize navigation performance. Like the unknown state transitions in the MDP formulation, it is challenging to obtain the observation model for this process. Therefore, we utilized model-free methods.

Again, with the same condition that the impatience levels of other vehicles are hidden, two different approaches were used.

- **DRQN:** A single observation space includes the ego vehicle and a fixed number (N) of nearby road users.
- **DQN with impatience-estimating LSTM:** Aims to convert POMDPs back into MDPs by estimating the impatience of each vehicle using Long Short-Term Memory (LSTM) networks. The reconstructed state with the estimated impatience is used to obtain Q-values from a DQN that is employed to solve independent MDPs.

III. METHODS

A. MDP

As explained in Section II, two different approaches were tested to solve the navigation problem. Both approaches share the same action space and reward model, but differ in the state space configuration.

The following part will explain the two different approaches to solving the previously defined MDP in more detail.

1) Combined MDP: In this approach, the ego vehicle solves a single, high-dimensional MDP that accounts for the ego vehicle and up to 20 vehicles in the simulation. The state space includes the physical states of the ego vehicle and the first 20 vehicles that appear in the simulation, ensuring consistency by always considering the road users in the order they are generated. This method assumes that considering 20 vehicles is sufficient for the given task, where the ego vehicle navigates an intersection.

The state space for each MDP is defined as follows:

$$s = \{t, \quad q_{ego}, \quad q_1, \quad q_2, \quad \dots \quad q_{20}\} \in S \tag{1}$$

where $q_i = \{x_i, y_i, v_i, a_i, imp_i\}$ represents the physical state of each vehicle, $imp_i \in [0,1]$ indicates the impatience of the vehicle, and i ranges from 1 to 20. Each element in q_i except imp_i is rounded to the first decimal place. For ego, we do not care about its impatience level. Therefore, $q_{ego} = \{x_{ego}, y_{ego}, v_{ego}, a_{ego}\}$ only represents its physical state. The state space thus has a dimensionality of $5 + 5 \times 20 = 105$, with each vehicle's state including position, speed, acceleration, and impatience. If fewer than 20 vehicles are present, the missing vehicles' states are padded with zeros to maintain consistency.

The decision-making algorithm using this approach is shown in Algorithm 1. The algorithm starts by collecting the current state. The ego vehicle then selects an action based on the estimated Q-values from the DQN network. The state and action are updated as the simulation progresses.

Algorithm 1 Decision-making algorithm to solve Combined MDP

```
ego_state, v_states ← reset()
while ¬terminate(ego_state, v_states) do
    state ← ego_state + Σ_{v∈v_states} v_state
    a, Q ← DQN(state)
    ego_state, v_states ← next_step(action)
end while
```

The DQN architecture includes three hidden layers. The first hidden layer has 64 neurons and uses the ReLU activation function. The second hidden layer has 128 neurons, also using ReLU activation. The third hidden layer has 64 neurons and uses ReLU activation as well. The output layer consists of a number of neurons equal to the number of possible actions. Each neuron corresponds to the Q-value for a specific action, and the output layer uses a linear activation function to produce these Q-values.

The DQN is trained to approximate the Q-values for each state-action pair. The target Q-values are updated according to the Bellman equation, and the loss function used is the mean squared error (MSE) between the predicted Q-values and the target Q-values. The optimizer employed is Adam with a learning rate of 0.001.

Training occurs over 5000 episodes. During training, an experience replay buffer stores past experiences, allowing the agent to learn from a diverse set of transitions. To ensure stability during training, the Q-network periodically updates its target network. This structure allows the agent to process high-dimensional state inputs, including the ego vehicle and surrounding vehicles, and learn the best action to take when navigating the intersection despite the complexity and uncertainty of vehicle interactions.

2) Solving Separate MDPs: In this approach, the ego vehicle solves MDPs for each surrounding vehicle. For instance, with four other vehicles present, four MDPs need to be addressed. For each MDP, the state space is defined as follows:

$$s = \{t, \quad q_{ego}, \quad q_{veh}\} \in S \tag{2}$$

This equation also follows the notation introduced in Section III-A-1. To be clear, the state space has a dimension of 10.

As explained in Section II-B, DQN was chosen to learn the Q-value function directly from the state input. The difference with the previously explained approach is that there can be multiple state inputs per scene and the ego has to choose the action between the resultant actions from each state. Since the safest decision is to do its best for the most dangerous state, we made ego choose the best action of the worst state.

The decision-making algorithm using this method is shown in Algorithm 2. For functions shown in the 2, $reset()$ runs the simulation until ego becomes apparent for the first time and returns the state of ego and other vehicles, $terminate(ego\_state, v\_states)$ return bool value about whether the current scene satisfies the terminate condition, and $next\_step(action)$ returns the next state of ego and vehicles by running the simulation for one step with given ego acceleration value.

The structure and training methods used for the DQN in this section are the same as those described in Section III-A-1, , allowing for an easier comparison of the performance between the two methods. Additionally, the maximum number of episodes for training the network was set to 5,000, which remains consistent across both methods. However, this approach accumulates more data because it is capable of collecting information from multiple states at each time step.

B. POMDP

As discussed in Section II, two approaches were tested to solve the navigation problem under a Partially Observable Markov Decision Process (POMDP), where impatience is unobserved. The following sections will explain these approaches in more detail.

Algorithm 2 Decision-making algorithm to solve MDPs per each vehicle

```
ego_state, v_states ← reset()
while ¬terminate(ego_state, veh_states) do
    states ← [ego_state + v_state | v_state ∈ v_states]
    worstQ ← −∞
    action ← 0
    for state ∈ states do
        a, Q ← DQN(state)
        if Q < worstQ then
            worstQ ← Q
            action ← a
        end if
    end for
    ego_state, v_states ← next_step(action)
end while
```

1) DRQN: In a Partially Observable Markov Decision Process (POMDP), the agent operates with incomplete knowledge of the environment. Specifically, the agent's observation does not fully describe the state of the system, which means the agent must make decisions based on a partial view of the environment. To handle this challenge, a Deep Recurrent Q-Network (DRQN) is employed, allowing the agent to process sequential observations and infer unobserved components of the environment [7], such as the driving styles of other vehicles.

The DRQN architecture consists of a recurrent neural network (RNN) layer, typically implemented using LSTM units, which enables the agent to maintain a memory of past observations. This is critical in environments where the current state does not fully capture the history of the system.

a) Architecture of DRQN: The DRQN architecture is composed of the following layers:

- **Input Layer:** At each time step, the agent receives an observation vector. This observation includes information about the ego vehicle's state and surrounding vehicles, such as their positions, velocities, and accelerations. Due to partial observability, the agent cannot directly observe the driving styles or future intentions of other vehicles. To handle varying numbers of surrounding vehicles, the observation is padded to a fixed length (up to a maximum of 20 vehicles).
- **Recurrent Layer (LSTM):** The core feature of the DRQN is the recurrent layer, which uses an LSTM network to process the sequence of observations. The LSTM unit maintains a hidden state, h_t, that stores the memory of past observations. This allows the agent to capture temporal dependencies and build a better understanding of the environment's dynamics, such as predicting the movements of other vehicles, even without knowing their future actions.
- **Fully Connected Layer:** After processing the sequence of observations through the LSTM, the final hidden state is passed through a fully connected layer to predict the Q-values for each possible action. The agent chooses actions based on the Q-values produced by the network.
- **Action Selection:** The agent selects an action by following an epsilon-greedy policy similar to the previous DQN.

b) Sequential Processing of Observations: The LSTM layer is crucial for capturing the temporal dependencies in the sequence of observations. As the agent receives a new observation at each time step, the LSTM updates its hidden state, which allows it to retain information about previous observations. This enables the agent to learn patterns over time, such as how other vehicles typically behave in certain situations, without explicitly observing their actions.

c) Training and Reward Mechanism: During training, the DRQN uses experience replay to store and sample past experiences in the form of observation-action-reward-next observation tuples. The agent learns by minimizing the Temporal Difference (TD) error. The DRQN updates its network weights to minimize the TD error, improving its ability to predict the best actions over time.

Algorithm 3 DRQN Algorithm

```
Initialize DRQN agent and replay buffer
while simulation running do
    action, hidden_state ← DRQN(observation, hidden_state)
    Execute action, get reward
    Store transition in buffer
    agent.train()
    if update condition then
        Update target network
    end if
end while
```

2) DQN with Impatience-estimating LSTM: This approach uses a impatience-estimating LSTM to infer the hidden state of impatience. When impatience can be inferred and if we can believe that result, it can be changed to MDPs again. This allows us to reuse the DQN trained in Section III-B-2.

We assume that all other state variables are fully observable, except for the impatience level of each vehicle. Therefore, estimating the state for these variables could be a waste of resources. Drawing from the concept of Mixed Observable Markov Decision Processes (MOMDPs) [8], we focus solely on the hidden state, which is the impatience level of each vehicle. However, we cannot use model-based MOMDP solvers since both the state transition model and the observation model remain unknown. Consequently, we reformulate the problem as a Markov Decision Process (MDP) after estimating the impatience levels, allowing us to utilize Deep Q-Networks (DQN) once again.

Solving this problem is akin to addressing a POMDP focused on belief updates related to impatience, even though it may not yield an exact solution. The state space consists solely of one variable: the impatience level of each vehicle. The observable state encompasses the historical physical states of all vehicles, including the ego vehicle. There are

no actions to take, and the transition model can be described as $T(s'|s) = 1(s = s')$ assuming that the impatience of the vehicle remains constant throughout the episode. With this formulation, LSTM was chosen to estimate the impatience level since it is well known for its performance on time-dependent inputs and used in the article using a similar approach [3]. The LSTM model designed for estimating impatience is structured as follows:

- **Input Layer**: Accepts the concatenated features of the target vehicle and surrounding vehicles of that vehicle over 15-time steps.
- **Masking Layer**: Handles variable sequence lengths by masking irrelevant inputs, when only a limited history of the vehicle is available.
- **LSTM Layer**: Captures temporal dependencies and predicts the impatience state. The layer was set to have 128 hidden units.
- **Output Layer**: Outputs the belief state representing the estimated impatience probabilities.

The input for the LSTM model is carefully structured to capture the relative positions and states of vehicles surrounding the target vehicle. At each time step, the input vector starts with the state of the target vehicle itself. The next slot is allocated to the vehicle directly in front of the target vehicle. The subsequent two slots are reserved for vehicles approaching from the left, while the final two slots are designated for vehicles approaching from the right. This input vector formulation can be expressed as follows:

$$\{p_{veh}, \quad p_{front}, \quad p_{left_1}, \quad p_{left_2}, \quad p_{right_1}, \quad p_{right_2}\}$$

All components in this expression originate from the same time step, and $p_i = \{x_i, y_i, v_i, a_i\}$ is used instead of q_i since the impatience level of each vehicle is unobservable. If there are no vehicles in any of these positions, the corresponding slots are filled with zeros. This structured approach ensures that the LSTM model receives a consistent and comprehensive representation of the traffic environment surrounding the target vehicle. The LSTM model was trained with the data from 1000 episode runs for 200 epochs.

The entire decision-making process used in this method is expressed in Algorithm 4. Since the impatience levels of each vehicle are unobservable, ego can only get information about the physical state of each vehicle, expressed as vp_states.

To utilize history of each vehicle to estimate the impatience level of each vehicle, these physical states are tracked using $v\_history$, and updated per timestep using $update\_v\_history(v\_history, ego\_state, vp\_states)$. Since the relative position matters in building the input vector for the LSTM, histories of all vehicles in the current scene are divided depending on its current direction, using $build\_direction\_dict(ego\_state, vp\_states)$. With this direction dictionary, vehicles on front, left and right can be easily defined for each vehicle.

When the probability of being impatience is given, the impatience level of each vehicle is set to 1 if the probability is higher than 0.4. This metric was used to make safer decisions if there is high uncertainty on the impatience level because assuming an impatient vehicle to be patient can be dangerous. With the estimated impatience level, the entire state can be constructed and the rest of the decision-making process is the same as in Section III-B-2.

Algorithm 4 Decision-making algorithm to solve POMDPs per each vehicle with estimated belief on impatience

```
ego_state, vp_states ← reset()
v_history ← {}
while ¬terminate(ego_state, vp_states) do
    update_v_history(v_history, ego_state, vp_states)
    N, S, W, E ← build_direction_dict(ego_state, vp_states)
    imp_belief ← {}
    for veh, hist ∈ v_history do
        if veh ≠ ego then
            F, L, R ← surrounding_vehs(veh, N, S, W, E)
            inputs ← []
            for t ∈ len(hist) do
                input ← hist[t] + F[t] + L[t] + R[t]
                inputs.append(input)
            end for
            imp_belief[veh] ← LSTM(inputs)
        end if
    end for
    worstQ ← −∞
    action ← 0
    for veh, p_state ∈ vp_states do
        a, Q ← DQN(ego_state, p_state, imp_belief[veh])
        if Q < worstQ then
            worstQ ← Q
            action ← a
        end if
    end for
    ego_state, vp_states ← next_step(action)
end while
```

IV. Results

The evaluation results for the different approaches are summarized in Table III. This table presents key metrics such as the number of successful passes, average time on successful passes, collisions, speedings, stoppings, and emergency stoppings from road users. Each evaluation is run on 500 episodes.

In the following subsections, we analyze and compare the results for each approach, focusing on the performance of methods under MDP and POMDP frameworks, as well as the contrast between these two types of decision processes.

A. Between MDP Methods

Table III indicates slightly better navigation performance for the Independent DQN than that of Combined DQN, in terms of both safety and efficiency. However, the Combined DQN maintains better speed compliance.

The performance differences stem from the distinct approaches of the two methods. The Combined DQN integrates

Metric	Combined DQN	Indepedent DQN	Combined DRQN	Est. Impatience DQN
Successful Passes	442	454	439	449
Average Time on Successful Passes (s)	6.81	6.0	9.14	6.0
Number of Collisions	58	46	61	51
Average Timelength Exceeding the Speed Limit (s)	1.39	2.25	1.66	2.25
Average Timelength of Stopping at the Wrong Place (s)	0.13	0.1	0.47	0.1
Average Number of Emergency Stoppings from Road Users	0.79	0.77	0.97	0.64

TABLE III

COMPARISON OF EVALUATION RESULTS FOR COMBINED DQN, SEPARATE DQN, COMBINED DRQN, AND DQN WITH ESTIMATED IMPATIENCE.

inputs from up to 20 vehicles in a single Markov Decision Process (MDP), which can lead to suboptimal decisions due to its high-dimensional state space. This complexity makes effective learning difficult, especially with limited training data, and hinders the model's ability to generalize. Although it handles speed compliance well, which is clearly represented in the state, it struggles to manage interactions between vehicles effectively.

In contrast, the Independent DQN focuses on individual MDPs for each vehicle, prioritizing safe maneuvers. By managing smaller state spaces, it learns from available data more efficiently, giving it a performance advantage.

B. Between POMDP Methods

For the approaches to solve the POMDP problem, DQN with estimated impatience showed slightly better performance on safety and much higher efficiency, while the Combined DRQN was better at obeying the speed limit.

The varied performance can be attributed to the underlying methods of each approach. Despite leveraging historical data to improve decisions, the Combined DRQN can still find it difficult to manage the intricate dynamics of the environment since the input space is too large, given limited amount of data. Especially, LSTM is well known for its requirement for huge amount of data [9].

Conversely, DQN with estimated impatience utilizes an LSTM to only estimate the impatience levels of individual vehicles. Therefore, it can focus more precisely on critical variables. After the impatience estimation, this method leverages the values from the previously trained Independent DQN, trained in much smaller state spaces. This approach leads to enhanced performance, offering better efficiency and safety.

C. Between MDP and POMDP Methods

When comparing MDP methods (Combined DQN and Independent DQN) to POMDP methods (Combined DRQN and Estimated Impatience DQN), the MDP methods generally demonstrate stronger performance in terms of navigation and collision avoidance, as expected.

The performance difference is mainly due to the complexity and data needs of POMDP methods, which use LSTM and require substantial training data. For instance, training the LSTM model in DQN with estimated impatience took about 10 minutes per epoch, and the model had a 29% error rate in estimating impatience during evaluations. This indicates that the LSTM models for both POMDP methods were insufficiently trained, resulting in weaker performance compared to MDP methods.

Notably, the policy from the DQN with estimated impatience, which is a POMDP method, outperformed the policy from the Combined DQN. This difference in performance could be attributed to the disparities between the Independent DQN and Combined DQN. We can anticipate that the performance gap resulting from misunderstandings of the interactions between vehicles was larger than the gap caused by misestimating the impatience level of each vehicle.

V. DISCUSSION

The results highlight the distinct advantages and trade-offs of the various methods evaluated in both MDP and POMDP frameworks. In terms of performance, the Independent DQN offers superior efficiency and safety in simpler environments, while the Combined DQN excels in speed regulation despite its higher complexity. The DRQN and the DQN with estimated impatience show that handling partial observability through memory-based and belief-estimation approaches can continue to perform decision-making, although challenges remain in fully capturing interactions and inferring hidden states like impatience. Our findings indicate that understanding the latent state of agents impacts both navigation safety and efficiency. Interestingly, we observed that DQN using inaccurate impatience information outperformed a combined DQN that had complete visibility of impatience. This suggests that it may be more advantageous to effectively manage the observable state before concentrating on latent states, as handling the observable state is significantly easier.

However, there are several key areas for improvement. Future work can focus on refining reward engineering to better balance safety and efficiency, enhancing environment simulations to capture more complex traffic scenarios, and leveraging more computing resources to optimize training and model performance. Additionally, exploring advanced architectures for belief updates and improving memory handling could lead to more effective solutions for partially observable environments.

In conclusion, while the current methods provide reasonable policies for decision-making in traffic navigation, further advancements in model sophistication and computational resources will be crucial for improving performance and applicability in real-world, complex environments.

VI. Appendix

- Joonwon Kang mainly worked on Independent DQN and DQN with estimated impatience.
- Tae Yang mainly worked on Combined DQN and DRQN.

References

[1] J. Wang, L. Zhang, Y. Huang, and J. Zhao, "Safety of autonomous vehicles," *Journal of Advanced Transportation*, vol. 2020, p. 13 pages, 2020. [Online]. Available: https://doi.org/10.1155/2020/8867757

[2] Z. N. Sunberg, C. J. Ho, and M. J. Kochenderfer, "The value of inferring the internal state of traffic participants for autonomous freeway driving," in *2017 American control conference (ACC)*. IEEE, 2017, pp. 3004–3010.

[3] D. J. Phillips, T. A. Wheeler, and M. J. Kochenderfer, "Generalizable intention prediction of human drivers at intersections," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1665–1670.

[4] S. Liu, P. Chang, H. Chen, N. Chakraborty, and K. Driggs-Campbell, "Learning to navigate intersections with unsupervised driver trait inference," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 3576–3582.

[5] German Aerospace Center (DLR), "Definition of vehicles, vehicle types, and routes," 2024, (Accessed: 2024-12-08). [Online]. Available: https://sumo.dlr.de/docs/Definition_of_Vehicles%2C_Vehicle_Types%2C_and_Routes.htmlimpatience

[6] 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, pp. 529–533, 2015.

[7] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," 2017. [Online]. Available: https://arxiv.org/abs/1507.06527

[8] I. Chades, J. Carwardine, T. Martin, S. Nicol, R. Sabbadin, and O. Buffet, "Momdps: a solution for modelling adaptive management problems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, no. 1, 2012, pp. 267–273.

[9] G. Van Houdt, C. Mosquera, and G. Nápoles, "A review on the long short-term memory model," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5929–5955, 2020.