Safe Lane Merging for Autonomous Cars on the Highway through Markov Decision Process

Adarsh Anil Kumar
Dept. of Mechanical Engineering
Stanford University
Stanford, USA
adarshak@stanford.edu

Yiyang Mu
Dept. of Mechanical Engineering
Stanford University
Stanford, USA
yiyangmu@stanford.edu

Vishal Singh
Dept. of Mechanical Engineering
Stanford University
Stanford, USA
vishal98@stanford.edu

***Abstract*—The safety of autonomous driving on highways is critical, and the ability of the ego vehicle to make efficient decisions under uncertainty is crucial. One of the most significant decisions is determining the right timing to merge into a particular lane, which is influenced by neighboring vehicles' states. This paper explores a simplified lane merging problem using Markov Decision Processes (MDPs), where an ego-car aims to merge into another lane without colliding with neighboring cars. Q-learning, SARSA, and double Q-learning algorithms were used to solve the MDP, and their performance was compared. The results show that Q-learning is the best algorithm for this problem when comparing the benchmarks values of performance, computational complexity, and convergence for all the strategies. This paper provides insight into the various decisions and uncertainties involved in autonomous driving.**

***Index Terms*—Markov Decision Process, Q-learning, Reinforcement Learning, route planning, autonomous vehicles, self-driving, SARSA, Double Q-learning**

I. Introduction

The safety of autonomous driving on the highway highly depends on the ability of an ego vehicle to make efficient decisions under uncertainty, which is a crucial aspect of Markov Decision Processes (MDPs). One of the most significant decisions autonomous cars face on the highway is determining the right timing to safely merge into a particular lane. This scenario is influenced by the states of neighboring vehicles on the road, making optimizing and solving a series of sequential decision-making problems challenging.

In this paper, we explore a simplified lane merging problem, modeled as an MDP, where an ego-car navigates a two-lane highway and aims to merge into the other lane without colliding with the cars on that lane. Our simulation environment assumes the ego-car has access to the relative distances between itself and the neighboring cars using either a LIDAR or a camera-based sensor system. In order to navigate a successful lane change, the ego-car can take four possible actions: merge in, accelerate, decelerate, and remain at a constant velocity.

Since the objective is to successfully merge to the other lane, all the other actions are complimentary in order to avoid a collision with neighboring vehicles. In order to remain safe, the ego-car must also follow the road rules such as the highway speed limit. As the neighboring cars' positions are actively changing which creates a source of uncertainty, an optimal action plan must be generated for the ego-car to achieve a perfect collision-free lane merging. The ego-car can merge into a gap without uncertainty if it is at a safe following distance from the neighboring vehicles, measured by bumper-to-bumper distances. Otherwise, the ego can still choose to merge in, but the probability of successful merging will depend on the relative distances from the cars on the other lane. It can instead adjust the distance by accelerating or decelerating. It can also maintain a constant velocity to wait for another opportunity to merge.

This paper explores and compares the performance of various reinforcement learning algorithms such as Q-learning, State–Action–Reward–State–Action (SARSA), and double Q-learning for this lane-changing MDP problem for self-driving cars on the highway. To validate the performance of these strategies we have developed a simulator that generates simulation rollout data (i.e., state, action, reward, and next state) for multiple episodes.

II. Related Work

Lane changing is a crucial task for autonomous vehicles to ensure efficient and safe travel. Various studies have been conducted to develop different algorithms for autonomous lane changing. [1] uses Machine Learning models to navigate lane changing and merging maneuvers by training it with the data of traffic density, road geometry, and non-cooperative and cooperative driving behaviors but it fails to take into account uncertainty factors of cars on the road. In [2], the authors proposed a centralized control framework for autonomous lane changing but its success heavily depends on knowing more about the states of the neighboring cars at all times since it uses MPC. [3] presented a similar approach of a model predictive control-based for autonomous lane changing where a short-term trajectory plan is created for the vehicle that satisfies a set of constraints, such as collision avoidance and comfort constraints. In [4], a novel neural network-based approach was used for vision-based perception and control of a self-driving car which shows great success but is computationally expensive for live driving scenarios.

Given the fact there are several sources of uncertainty in this lane-changing scenario, defining the model as a Markov

Decision Process can yield a lot of success when accompanied by training models and optimal policy algorithms. [5] proposed a model-based reinforcement learning approach to solve Markov Decision Processes where the authors propose an algorithm, called MBAR, that uses a linear programming formulation to estimate the average reward function and the optimal policy. The paper provides a theoretical analysis of the algorithm's convergence and sample complexity, as well as experimental results on a variety of benchmark problems. We combine leanings from this paper along with the book, [6], to simplify the lane-changing scenario model and implement other straightforward optimal policy and exploration strategies for the best course of action. [6] provides a detailed overview of working code and algorithms on the topic of decision-making, including reinforcement learning, for various applications.

For our approach, we take inspiration from [7] which introduces a simulation-based approach to decision-making in autonomous vehicles. The authors use a high-fidelity driving simulator to generate training data for a reinforcement learning algorithm, which learns to make driving decisions based on a variety of inputs, such as road geometry, traffic conditions, and vehicle speed. We develop a simulator to generate training data based on a pre-defined transition and reward model which is then used to implement the various optimal policy algorithms such as Q-learning, SARSA and double Q-learning.

III. Methodology and Approach

The environment for the model in this paper is a highway that consists of two lanes where the ego-car is driving on one lane and there are 0 to 2 neighboring cars on the other lane. The primary objective of the ego-car is to merge into the other lane. A simulator is developed to model the problem as a Markov Decision Process and generates rollout data for several episodes of lane-merging based on pre-defined state, action, transition, and reward spaces. Then, algorithms for determining the optimal policy through apt exploration techniques are developed and compared. The training of each of the strategies is done over a common horizon of 100 for the rollout data. This section describes the setup of the simulator and implementation of the Q-learning, SARSA, and Double Q-learning strategies. The latter algorithms aim to learn the optimal policy that maximizes the expected total reward received by an agent while interacting with an environment, which is ideal for solving MDPs. This paper explores the applicability of each of these algorithms as they differ in off-policy, on-policy, and modifications on certain aspects.

A. Simulator Setup

1) State Space Definition: The objective of this model is to enable the ego-car to safely merge into the neighboring lane, without colliding with the forward and backward neighbors. The state space of the ego-car, denoted by S, is hence defined as follows:

$$\mathcal{S} = (v, d_1, d_2)$$

where $\mathbf{v}$ is the one-dimensional velocity of the ego-car, and d_1 and d_2 denote the relative bumper-to-bumper distances of the ego-car to the forward and backward neighbors respectively. A visual representation of the scenario is shown in Figure 1.

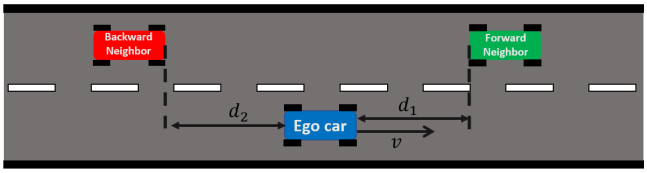


Fig. 1: State Space of Ego-Car in two-lane merging scenario

The values in the state space S are confined within the following intervals

$$\mathbf{v} \in [50, 51, 52, ..., 69, 70]$$

$$\mathbf{d_1}, \mathbf{d_2} \in [0, 1, 2, ..., 13, 14]$$

It is important to note that d_1 and d_2 are defined in terms of car lengths to account for change in velocity from both the neighboring cars when determing the relative distances. It can be seen that v can take up to 21 unique values, whereas each distance can take 15 unique values. In total, there are $21 \cdot 15^2 = 4725$ possible states, excluding the terminal states.

In order to allow for efficient computation and easy storage, we implement linear indexing of the state space as follows

$$s = (v - 50) \times 15^2 + d_1 \times 15 + d_2$$

The reason for linear indexing the state space in the given manner is to map the multi-dimensional state space into a one-dimensional index, which can be used as an index for arrays or tables to efficiently store and access the values associated with each state-action pair. As a result, we can represent each state with a unique index ranging from 0 to 4,724.

Even though the objective is to change lanes, the scenario can end up with various results other than a successful lane change such as a collision or over/under speeding. Hence, The terminal states for this model are expressed in the following manner defined under the linear index s:

- Successful Lane Merge: $s = 10000$
- Collision: $s = -10000$.
- Out of Bounds: $s = -1$. This occurs when the velocity is not within the highway speed limit ($v > 70$ or $v < 50$).

It is important to note that if $d_1 > 14$ and $d_2 > 14$, it is considered that there are no cars nearby and the transition probability to $s = 1000$ (i.e., successful lane merge) is 1.

2) Action Space: In order to attempt to change lanes, the vehicle has only a limited set of actions it can carry out to reach its goal position, taking the environment of other cars on the highway into account. As result, the action space of the ego-car is given by

$$\mathcal{A} = [a_1, a_2, a_3, a_4]$$

where the actions are defined as

- a_1 = merge in
- a_2 = accelerate in
- a_3 = decelerate in
- a_4 = remain in constant velocity

To keep the dynamics of the ego-car simple and the model constrained within one dimension, we assume that a_1 happens instantaneously. This enables us not to consider the transverse velocity and displacement of the ego-car during merging.

3) Transition Model: In order to establish the transition model for this scenario, we introduce the term d_s which is the 'Safe following distance' defined as

$$d_s = \frac{v}{5}$$

This describes the "two-second" rule of thumb used as a safe-driving measure on the highway which ensures a distance of 1 vehicle length for every 5MPH of the driver's speed [9]. We also assume the neighboring drivers are mostly rational in how they navigate the highway but may make inappropriate decisions randomly due to the lack of information, which introduces uncertainty into this model. For each of the actions, the probabilities are defined in Table I.

TABLE I: Transition Probabilities for different actions

s'	**a**	**T(s' | s, a)**
10000	a_1	$\begin{cases} 0 & \text{if } d_1 \text{ or } d_2 = 0 \\ 0.7^F & \text{otherwise} \end{cases}$
-10000	a_1	1 - T(s'= 10000 | s, a)
-1	a_1	0
$(v+1, d_1', d_2')$	a_2	$P_{a_2}(d_1') \times P_{a_2}(d_2')$
$(v-1, d_1', d_2')$	a_3	$P_{a_3}(d_1') \times P_{a_3}(d_2')$
(v, d_1', d_2')	a_4	$P_{a_4}(d_1') \times P_{a_4}(d_2')$

The values of the constants and dependant probability functions defined in Table I are further described in Appendix A. It can be seen that velocity increases by 1, decreases by 1, and stays the same for actions a_2, a_3, and a_4 respectively.

4) Transition Model: The reward for the transition model and action taken is as follows

If a = a_1:

$$R(s', a, s) = \begin{cases} 10 & \text{if } s' = 10000 \text{ (success)} \\ -1000 & \text{if } s' = -10000 \text{ (collision)} \end{cases}$$

If a = a_2, a_3, or a_4:

$$R(s', a, s) = \begin{cases} 0 & \text{if } s \in [0, 4724] \\ -10 & \text{if } s' = -1 \text{ (out of bounds)} \end{cases}$$

The values assigned have been fine-tuned after running several simulations to provide the best behavior from the optimal policy algorithms.

B. Q Learning

Q learning is a model-free reinforcement learning algorithm (see [6]) that involves learning a state-action value function, called the Q function: Q(s, a), which maps state-action pairs to their expected total discounted rewards. It is used to find the optimal policy for a given Markov Decision Process. For the purpose of our algorithm, the action values function is defined as a modification of the Bellman expectation equation given by

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s' \mid s, a) \max_{a'} Q(s', a')$$

The algorithm updates the Q values iteratively by taking actions in the environment and observing the resulting state and reward. This is defined in the following form

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

The updated Q values are then used to choose the optimal policy, which is the policy that maximizes the expected cumulative reward. In this implementation, we apply Q learning to a simulated autonomous driving scenario, where the goal is to learn a policy that can merge an autonomous vehicle into the other lane of a highway with other vehicles.

Algorithm 1 Q-learning

```
Initialization: Q(s, a) = 0 ∀s ∈ S, a ∈ A
for each episodes do
    s = initial state
    while s not terminal do
        a ← argmax(Q(s', A))
        Take action a, observe next state s' and reward r
        Q(s, a) ← (1−α)Q(s, a) + α(r + γ max_a' Q(s', a'))
        s ← s'
    end while
end for
return policy π from Q
```

The Q-learning implementation takes in the rollout data which contains information regarding state, action, reward, and next state for several episodes, and determines the optimal policy using Algorithm 1. A greedy action is taken at each step as an exploration strategy. The training method trains the Q learning algorithm for a given number of iterations, updates the Q table for each transition, and outputs the learned policy. The policy is generated in a general manner as the action with the highest Q-value in each state.

A random policy for the given data is also generated by picking random actions to compare the performance of Q-learning. The comparison evaluation calculates the mean discounted return for a set of initial states, as well as the rates of success merge and collision for each policy. The policy score is thus the mean discounted return for a policy relative to a random policy. The policy score, rate of collision, and rate of a successful merge of the evaluation are noted, along with the time elapsed for the entire process.

C. SARSA

The SARSA algorithm is also implemented to learn an optimal policy for the lane merging problem. It derives its

name from the fact that it uses (s, a, r, s', a') to update the Q function at each step. It uses the actual next action a' to update Q instead of maximizing over all possible actions:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma Q(s',a') - Q(s,a)\right)$$

SARSA is classified as an on-policy reinforcement learning approach as it aims to estimate the value of the exploration policy while following it. On the other hand, Q-learning is considered an off-policy method because it seeks to determine the value of the optimal policy while following the exploration strategy. The implementation of this algorithm can be seen in Algorithm 3. The algorithm initializes its hyper-parameters such as the learning rate, discount factor, exploration rate, and decay factor. The algorithm then reads data from the simulation rollouts, which consist of state-action pairs, the resulting reward, and the next state. The Q function is updated for each state-action pair, and it updates the Q using the updating method described above. The exploration is done through an implementation of Epsilon-greedy strategy to select the next action based on the current state which is shown in Algorithm 2 i.e with the probability 1-ϵ, it selects the action with the highest Q-value for the current state otherwise it selects a random action. The algorithm runs the exploration and updating steps for a fixed number of episodes defined by the hyper-parameter horizon. After the Q function is trained, the policy and its score are generated. The Q function, along with other outputs from the evaluation, is then compared to the random policy similar to how it was done in Q-learning.

Algorithm 2 Epsilon-Greedy Algorithm

r = random()
if $r > \epsilon$ **then**
  $a \leftarrow argmax_{a'} Q(s, a')$
else
  $\epsilon \leftarrow \beta\epsilon$
  $a \leftarrow random(\mathcal{A})$
end if
return a

D. Double Q Learning

Q-learning can perform poorly due to it overestimating action values. The issue arises in the manner the Q function is updated as the maximum action value is used as an approximation for the maximum expected action value. To avoid overestimation in Q-learning, the double Q-learning algorithm was proposed. Here a double estimator is implemented from independent sets of experiences, with one estimator determining the maximizing action and the other providing the estimate of its value. The only caveat is that double Q-learning sometimes underestimates the action values. We use a simplification of the implementation suggested in [8]. The implementation of this strategy can be seen in Algorithm 4.

Algorithm 3 SARSA Algorithm

Initialization: $Q(s,a) = 0\ \forall s \in \mathcal{S},\ a \in \mathcal{A}(s)$
for each episode **do**
  s = initial state
  $a \leftarrow$ Epsilon-Greedy(s)
  while s not terminal **do**
    Take action a' and observe reward r and next state s'
    $a' \leftarrow$ Epsilon-Greedy(s')
    $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot (r + \gamma Q(s',a') - Q(s,a))$
    $s \leftarrow s'$
    $a \leftarrow a'$
  end while
end for
return policy π from Q

The algorithm randomly chooses one of the two Q-matrices (Q_A or Q_B) to update the Q-value of a state-action pair at each iteration, which is done in the following manner

$$Q_{s,a,A} \leftarrow Q_{s,a,A} + \alpha(r + \gamma Q_{s',a',B} - Q_{s,a,A})$$

$$Q_{s,a,B} \leftarrow Q_{s,a,B} + \alpha(r + \gamma Q_{s',a',A} - Q_{s,a,B})$$

An Epsilon-greedy exploration strategy is used to choose the next action. With a probability of ϵ, a random action is chosen. Otherwise, the action with the highest Q-value in the selected matrix is chosen.

Algorithm 4 Double Q-learning Algorithm

Initialization: $Q_A(s,a), Q_B(s,a) = 0\ \forall s \in \mathcal{S},\ a \in \mathcal{A}(s)$
for each episode **do**
  s = initial state
  $a \leftarrow$ Epsilon-Greedy(s)
  while s not terminal **do**
    Randomly choose Q_A or Q_B with probability 0.5
    $a \leftarrow$ Epsilon-Greedy(s')
    Take action a' and observe reward r and next state s'
    if s' is terminal **then**
      $Q_{s,a} \leftarrow r$
    else
      if Q_A was chosen **then**
        $Q_{s,a,A} \leftarrow Q_{s,a,A} + \alpha(r + \gamma Q_{s',a',B} - Q_{s,a,A})$
      else
        $Q_{s,a,B} \leftarrow Q_{s,a,B} + \alpha(r + \gamma Q_{s',a',A} - Q_{s,a,B})$
      end if
    end if
    $s \leftarrow s'$
    $a \leftarrow a'$
  end while
end for
$Q_{s,a} \leftarrow (Q_{s,a,A} + Q_{s,a,B})/2$
return policy π from Q

TABLE II: Performance of Optimal Policy Algorithms

Strategy	Rate of Successful Lane Merge (%)	Rate of Collision (%)	Computational Time (s)
Q-learning	70.01	0.59	208.35
SARSA	69.4	0.87	224.36
Double Q-learning	73.37	0.46	313.16

Training the Double Q-Learning algorithm is executed for a fixed number of iterations over all the state-action pairs in the training data. It then calculates the optimal policy using the average of the two Q-matrices. Finally, the policy score is generated for the optimal policy calculated by the algorithm. The policy scores along with other outputs such as the rate of collision, rate of successful lane merge, and computational time are saved and compared to a random policy.

IV. Analysis and Results

A. Overall Performance for Safe and Optimal Lane-Changing

The simulator produces rollout data which represent the various number of episodes of lane changing for the ego-car in the model described above. All three algorithms are trained using the same exact data and produce the optimal policy. The key aspects of comparison between the different strategies implemented are the rate of successful lane merge, rate of collisions, and computation time for each optimal policy algorithm. The final results can be seen in TABLE II. The relative score of the policy generated by each of the algorithms is also crucial to understand how they perform with respect to a random policy generator. These results can be viewed in Figure 2. Here the relative policy score is the mean discounted return for a policy relative to a random policy.

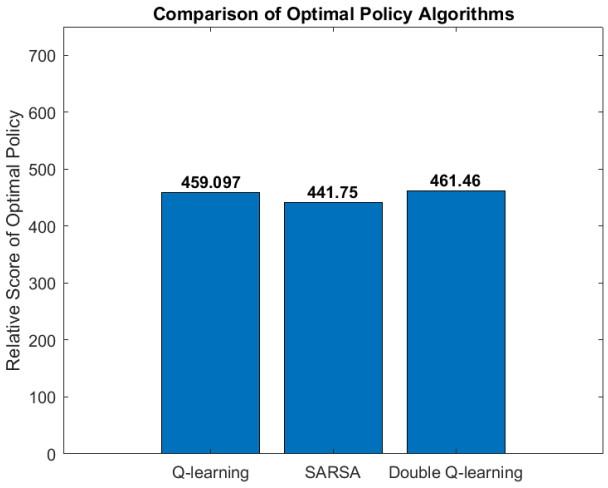


Fig. 2: Comparison of Relative policy scores

From these results, it can be easily deciphered that both Q-learning and double Q-learning perform better than SARSA. SARSA is an on-policy algorithm, while Q-learning is an off-policy algorithm. This means that SARSA learns the value of the policy it is currently following, while Q-learning learns the value of the optimal policy. Hence this lower success rate and score may have arisen due to the fact that the optimal policy at any state may differ significantly from the current policy being followed by SARSA, which can lead to sub-optimal results. This is also an environment with a large variance in the transition dynamics. In such environments, the policy being followed by SARSA may lead to a large variance in the next state, which can also be a reason for its sub-optimal results. The difference between Q-learning and double Q-learning is minimal in our scenario which can be explained by the limited overestimation of the action value by the Q-learning strategy. Though double-Q learning performs slightly better (as it is designed to), it is the slowest solver out of all three of the algorithms. This can be credited to working with two estimators for the action value function. As a result, the implementation of Q-learning seems like the best strategy from these aspects for our scenario of a simplified lane-changing event on the highway. It is to be noted that though the simulator parameters and training horizon size are uniform across all the strategies, other hyper-parameters such as learning rate, discount factor, exploration rate, and decay factor have been laboriously fine-tuned to produce the best results (though it is not guaranteed to be the **best**). An impressive aspect of the quality of performance of these algorithms lies in the fact that they all achieve a **rate of collision less than 1%** and a **rate of successful lane merge greater than 69%**.

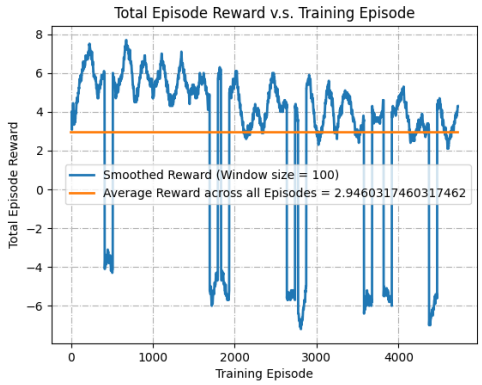
B. Behavior of Rewards across all Episodes

The total reward of each optimal policy over all the training episodes was also calculated. Due to the high variance in data, we implement a moving average with a sliding window of length 100. This data along with the average total reward over all episodes for all three algorithms is plotted in Figure 3.

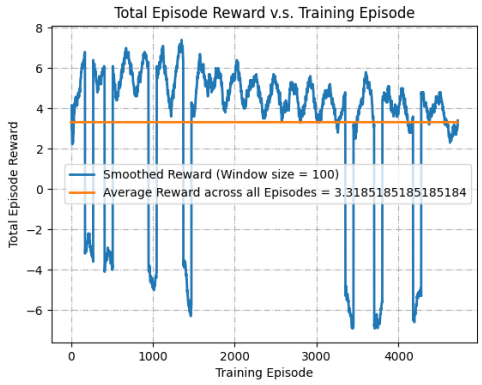
From these results, we can see that SARSA shows a better convergence of its total rewards as the number of episodes increases for the training. Ignoring outliers, the graphs also present that **Q-learning and double Q-learning are quicker to converge** to a certain range of reward across episodes. The **average reward data is highest for Q-learning**. This shows that it is better at generating a policy plan that can produce safer and more successful actions.

V. Conclusion

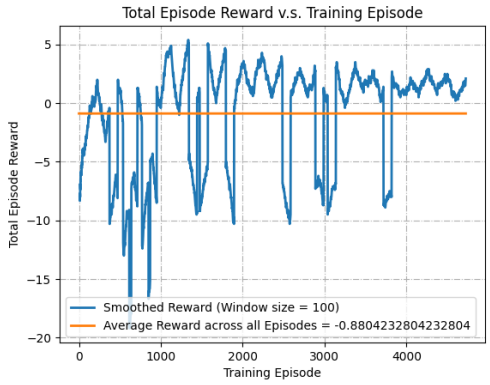
In this paper, a simplified approach for decision-making under uncertainties in the realm of safe lane-changing for autonomous driving was discussed and analyzed. Here uncertainties are due to the random behaviors of the other drivers and the limitation of the vehicle's ability to perceive the surrounding environment. Specifically, the focus was on addressing uncertainties arising from the random behaviors of neighboring vehicles. The proposed method utilized a Markov Decision Proces model to plan the vehicle's motion based on



(a) Q-learning



(b) SARSA



(c) Double Q-learning

Fig. 3: Plot of Total Episode Reward vs Training Episode

the states of neighboring cars, followed by implementing Q-Learning, SARSA, and Double Q-Learning algorithms to solve the MDP. The score for each policy was calculated as the mean discounted return for all possible states relative to a random policy. Each of the policies was optimized by adjusting the hyper-parameters. The three algorithms were compared using their relative policy score, rates of collision and successful lane merge, computational time, and convergence of total rewards. From the results, we can conclude that Q-learning performs the best across all the factors considered. This result is mainly justified by the fact that Q-learning with simply greedy exploration is most optimal for the simplified MDP model as there isn't much room for overestimation, it is computationally less complex and has a higher rate of convergence along with higher average reward across training episodes.

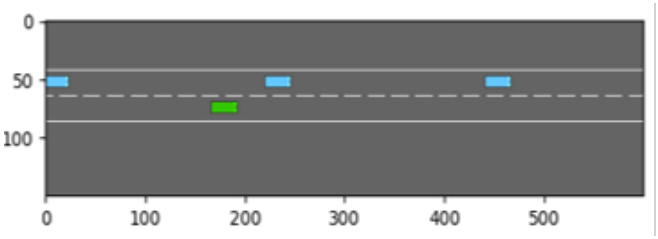


Fig. 4: Ego-car (green) and neighboring cars (blue) modeled in dynamic Highway environment using "highway-env"

The room for improvement mainly is with respect to handling continuous state spaces and moving away from tabular methods which take exponentially longer to converge especially for real-life data. Taking inspiration from [10], the model can be developed to use actual highway data instead of a simulator which produces a random data set. For future work, we will consider a continuous Partially Observable Markov Decision Process to better model the highway situation. We attempted to incorporate continuous and discrete actions to do the low-level controls of vehicle kinematics, namely speed, steering, and throttle (see Figure 4 and Appendix A) using a simulator called "highway-env" (see [11]). Due to the time limitation, we were not able to completely incorporate optimal policy solvers into this highly dynamic highway environment. But it is certainly the next step of progress as it will provide a better understanding of the various decisions and uncertainties involved in autonomous driving. Tabular methods used in the study can become impractical when dealing with a large state-action space. To address this limitation, our study suggests switching to strategies like Deep Q-Networks, which can handle higher dimensional data space and perform better for autonomous lane changing. Overall, this study serves as a foundation for us to further research the decision-making under uncertainties for autonomous driving.

VI. Contribution

All the team members worked together in developing a simplified model for lane-changing of self-driving cars and formulating it as an MDP process. A. Anil Kumar worked on the complete write-up of the paper and analyzed the results using different parameters and plots. Y. Mu developed the simulator and implemented all three reinforcement learning algorithms in Python. V. Singh contributed to exploring dynamic and realistic highway simulators as well as fine-tuning all hyper-parameters for the best results. All the members equally contributed to the project from ideation to completion through different avenues of expertise. The GitHub repository for this project can be accessed here: AA228 W23 Final Project- Simple MDP Lane Merge on Freeway.

REFERENCES

[1] Elbanhawi, M., Hussein, M., A Survey of Lane Changing and Merging Manoeuvres, IEEE Transactions on Intelligent Transportation Systems, 18(11), 3015-3030, 2017

[2] Deo, A., Ioannou, T. R. (1998), Autonomous lane change maneuvers for intelligent highway systems, IEEE Transactions on Intelligent Transportation Systems, 1(2), 102-116, 1998

[3] Al-Mosawi, S., Happee, R., Pauwelussen, J. P. T. (2016), An MPC-based approach for autonomous lane change maneuvers, IEEE Transactions on Intelligent Transportation Systems, 17(7), 1978-1988, 2016

[4] Pomerleau, D. A. (1989), ALVINN: An autonomous land vehicle in a neural network, Advances in neural information processing systems, 1, 305-313, 1989

[5] Tadepalli, P., Ok, D., Model-based average reward reinforcement learning. Journal of Machine Learning Research, 9, 1575-1603, 2008

[6] Kochenderfer, M. J., Wheeler, T. A., Wray, K. H., Algorithms for decision making, MIT Press, 2015

[7] Zheng, R., Liu, C., Guo, Q., A decision-making method for autonomous vehicles based on simulation and reinforcement learning, IEEE Transactions on Intelligent Transportation Systems, 18(11), 3004-3014, 2017

[8] Zhang, H., Pan, L., Mykel, J., Weighted double Q-learning, 2018

[9] Bi, H., Mao, T., Wang, Z., Deng, Z. (2017), A data-driven model for lane-changing in traffic simulation, Transportation Research Part C: Emerging Technologies, 79, 212-229, 2017

[10] NYS DMV - Driver's Manual - Chapter 8: Defensive Driving, New York State Department of Motor Vehicles, September 2011

[11] Highway-env documentation. Read the Docs. Available: https://highway-env.readthedocs.io/en/latest/.

APPENDIX A
ACTION DEPENDANT PROBABILITIES FOR TRANSITION MODELS

For a = a_1:

$$F = \max(d_s - d_1, 0) + \max(d_s - d_2, 0)$$

For a = a_2, a_3, a_4:

TABLE III: Probability Functions for different actions

d'	a	**P**(d'
$d'_1 = d_1 - 1$	a_2	$\begin{cases} 0.9 & \text{if } d_1 \geq d_s \\ 0.6 & \text{otherwise} \end{cases}$
$d'_1 = d_1$	a_2	$\begin{cases} 0.05 & \text{if } d_1 \geq d_s \\ 0.2 & \text{otherwise} \end{cases}$
$d'_1 = d_1 + 1$	a_2	$\begin{cases} 0.05 & \text{if } d_1 \geq d_s \\ 0.2 & \text{otherwise} \end{cases}$
$d'_1 = d_1 - 1$	a_3	$\begin{cases} 0.05 & \text{if } d_1 \geq d_s \\ 0.2 & \text{otherwise} \end{cases}$
$d'_1 = d_1$	a_3	$\begin{cases} 0.05 & \text{if } d_1 \geq d_s \\ 0.2 & \text{otherwise} \end{cases}$
$d'_1 = d_1 + 1$	a_3	$\begin{cases} 0.9 & \text{if } d_1 \geq d_s \\ 0.06 & \text{otherwise} \end{cases}$
$d'_1 = d_1 - 1$	a_4	$\begin{cases} 0.05 & \text{if } d_1 \geq d_s \\ 0.1 \cdot 0.9^{d_s - d_1} & \text{otherwise} \end{cases}$
$d'_1 = d_1$	a_4	$\begin{cases} 0.9 & \text{if } d_1 \geq d_s \\ 0.9^{d_s - d_2 + 1} & \text{otherwise} \end{cases}$
$d'_1 = d_1 + 1$	a_4	$\begin{cases} 0.05 & \text{if } d_1 \geq d_s \\ 1 - 0.9^{d_s - d_1} & \text{otherwise} \end{cases}$
$d'_2 = d_2 - 1$	a_2	$\begin{cases} 0.9 & \text{if } d_2 \geq d_s \\ 0.6 & \text{otherwise} \end{cases}$
$d'_1 = d_2$	a_2	$\begin{cases} 0.05 & \text{if } d_2 \geq d_s \\ 0.2 & \text{otherwise} \end{cases}$
$d'_1 = d_2 + 1$	a_2	$\begin{cases} 0.05 & \text{if } d_2 \geq d_s \\ 0.2 & \text{otherwise} \end{cases}$
$d'_2 = d_2 - 1$	a_3	$\begin{cases} 0.9 & \text{if } d_2 \geq d_s \\ 0.6 & \text{otherwise} \end{cases}$
$d'_2 = d_2$	a_3	$\begin{cases} 0.05 & \text{if } d_2 \geq d_s \\ 0.2 & \text{otherwise} \end{cases}$
$d'_2 = d_2 + 1$	a_3	$\begin{cases} 0.9 & \text{if } d_2 \geq d_s \\ 0.06 & \text{otherwise} \end{cases}$
$d'_2 = d_2 - 1$	a_4	$\begin{cases} 0.05 & \text{if } d_2 \geq d_s \\ 0.1 \cdot 0.9^{d_s - d_2} & \text{otherwise} \end{cases}$
$d'_2 = d_2$	a_4	$\begin{cases} 0.9 & \text{if } d_2 \geq d_s \\ 0.9^{d_s - d_2 + 1} & \text{otherwise} \end{cases}$
$d'_2 = d_2 + 1$	a_4	$\begin{cases} 0.05 & \text{if } d_2 \geq d_s \\ 1 - 0.9^{d_s - d_2} & \text{otherwise} \end{cases}$