Reinforcement Learning Survey for Intelligent Lane Maneuvering

Avni Sharma*,
*sharm104@msu.edu,

Abstract—**There are numerous reinforcement learning surveys, but none critique the algorithms used for lane maneuvers by automated/autonomous vehicles. We investigate different lane maneuvering tasks and the results of the respective reinforcement learning algorithms used. The lane maneuvering tasks are divided into lane changing, merging, overtaking, and lane centering/keeping. This survey provides context to an important application of reinforcement learning and outlines concepts and insights for future work.**

I. Introduction

An Automated Vehicle (AV) must perform all tasks that a human driver is capable of, in a manner that maximizes safety and efficiency. AV's have the potential to reduce collisions, improve mobility and access thereof, and to better traffic conditions [1]. To accomplish this task, AV's require many capabilities, including the ability to identify, track within, and switch between lanes. Such systems can be trained explicitly using conventional Artificial Intelligence or Machine Learning implementations, but the results may suffer from an inability to operate successfully in new environments and situations. On any given day, human drivers can be exposed to new situations that they have never encountered before, which makes the ability to generalize essential for AVs.

Another tool supporting the creation of automated vehicles is Reinforcement Learning (RL), a reward-policy based system for learning latent and overt rules governing system behavior. Like a child learning not to touch a hot stove based on a negative experience, systems can "learn" those behaviors that result in desirable and undesirable states or state transitions in order to optimize outcomes. By being trained this way, RL algorithms can generalize well to new scenarios, which means RL has the potential to be more robust than many traditional machine learning methods.

RL has been an important field of machine learning since 1979 [2]. Reinforcement learning is a type of learning that tries to mimic human learning of the real-world through interactions with the environment. In its simplest embodiment, an agent is placed in an environment with no previous training or knowledge. The agent is given a set of actions to choose from that change the state of the environment and the state of the agent. The agent is then given a reward, which may be positive or negative. Through its experiences in simulation, the system learns how to maximize its reward, and over time, converges on a solution that more reliably earns high rewards.

Reinforcement learning has numerous applications, including robotics [3], bug detection in software [4], and more. A popular domain of reinforcement learning is in the advancement of intelligent vehicle technology. Applications for intelligent and automated vehicle RL include safety [5], object recognition [6], path planning [7], and lane maneuvering [8].

This survey focuses on the use of RL for lane maneuvering in automated vehicles, including approaches for lane merging, lane centering, vehicle overtaking, and changing lanes. These topics were selected because RL lends itself naturally to control problems, and lateral control is an essential and challenging component to autonomous vehicles. If an autonomous vehicle cannot smoothly change lanes or keep itself centered in its lane, the passengers can quickly become uncomfortable and even doubt the future of autonomous mobility if problems continue. If autonomous vehicles drive like a 15 year old who just got their permit, public adoption and acceptance of autonomous mobility as the future will suffer. However, if we are able to understand how RL can be used for autonomous vehicles, even only for lateral control, researchers can be more directed in their efforts to advance the future of autonomous vehicles.

A. Contribution

This document summarizes and compares reinforcement learning applications within the context of intelligent vehicle lane maneuvering for the purpose of understanding what has already been done in this space, their levels of success, and what future directions can be pursued. We aim to provide sufficient context and review of prior art in the discipline to help the reader understand RL's utility in enabling safe vehicle maneuvering and its potential application to other domains related to AV's. Our goal is to make this survey accessible and understandable for people who are new to RL while still providing enough details to make the survey useful to experienced RL researchers. Surveys exist on reinforcement learning within robotics [3], wireless networks [9], and intelligent vehicles and assistance systems [10], but none regarding lane or lateral vehicle control. This is an area of high importance and therefore a contribution to the state of knowledge has the potential to positively benefit the discipline.

B. Overview

The paper is organized as follows. A thorough introduction to RL is given in section II. Section III explains how RL can be used in various lane maneuvering tasks by looking at relevant and recent works in this area. Section IV depicts future research opportunities, and section V gives closing remarks.

II. Background

A. Fundamentals

In order to understand RL and related learning methods, a number of concepts need to be defined. An RL algorithm can be viewed as an agent making decisions based upon what will maximize its reward. Therefore, the agent must choose an action $\boldsymbol{a}$ from its possible set of actions $\boldsymbol{\mathcal{A}}$. The proper action to choose depends on the current state of the system, where the current state $\boldsymbol{s}$ is in the set of all states $\boldsymbol{\mathcal{S}}$. The state can take into account factors of what the agent is controlling and how the agent currently exists in its environment. Transition dynamics $\Gamma(s_{t+1}|a_t, s_t)$ can also be defined to determine how an action at time t determines the next state of the system at time $t+1$.

Consider the scenario of a robotic arm learning to play ping pong. The RL agent is the control software for the robot, and its set of actions, $\boldsymbol{\mathcal{A}}$, would be the ways it can move its joints. The state, $\boldsymbol{\mathcal{S}}$, would take into account the position and velocity of the ping pong ball and the positions and rotational velocities of the joints in the robotic arm. The state should contain everything the agent needs to know in order to choose an action.

Viewing RL as an agent interacting with its environment by choosing actions that determine its next state and reward can be viewed as a Markov Decision Process (MDP), which is visualized in figure 1. In this figure, the assumption has been made that the state of the agent in its environment follows the Markov property, which means that the next state only depends on the current state and the chosen action, or equivalently, S_t and A_t. Without the Markov property, the next state would depend on the current and all previous states and actions $(S_t, A_t, S_{t-1}, A_{t-1}, ..., S_0, A_0)$.

This use of the Markov property significantly simplifies the calculations necessary to advance to the next state. In fact, Sutton and Barto state that any process which does not follow the Markov property should be considered as an approximation of a Markov process in the context of RL [2].

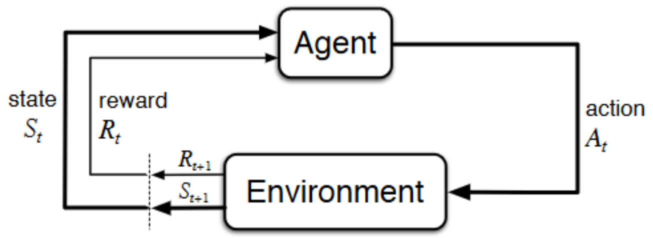


Fig. 1. RL agent and environment as a MDP. Figure from Sutton and Barto [2]

A crucial part of any RL implementation is how the agent is rewarded. By giving the agent positive rewards for desirable actions and negative rewards for undesirable actions, the potential exists for the agent to learn how to maximize its reward. The reward for any given action depends on the current state of the system and the next state that the action brought the system to and can be expressed as $\boldsymbol{\mathcal{R}}(\boldsymbol{a}_t, \boldsymbol{s}_t, \boldsymbol{s}_{t+1})$ where $\boldsymbol{a}_t$ represents the action chosen at time t which moved the system from state $\boldsymbol{s}_t$ to $\boldsymbol{s}_{t+1}$. By selecting actions and interacting with the environment, the agent can form an optimal policy π that maps states to actions.

Terminology regarding reinforcement learning simulations must be defined, as almost all approaches utilize simulation to train the agent. An episode refers to the time in which an agent is first placed into an environment, up until it does some undesirable action or successfully completes the task at hand. For example, Kaushik et al. [11] train an agent vehicle to perform overtaking using 20 episodes. They define the start of an episode as the beginning where all cars start at 0 km/hr, and the end of an episode is when the agent either collides with another vehicle or environment boundary such as walls, or when the car the successfully completes the overtaking task. Unless otherwise noted, we refer to this methodology when it comes to simulation of episodes (though the number of episodes may vary).

B. Exploration vs. Exploitation

The goal of an RL algorithm is to maximize its reward. If the best policy is known, the agent can simply follow this policy and it will achieve the highest possible reward. However, if a better policy exists, the agent will never discover the better actions by only following the policy that it currently believes to be the best. Therefore, the agent must explore its state-action space in order to learn a better policy.

This is the motivation for an RL algorithm to balance how much it explores or exploits a policy. The benefit of exploiting a good policy is that high rewards are generated, but the cost of never deviating from a policy is that any better policy will not be discovered and no new learning will occur. This is precisely the benefit of exploring, which is where a RL algorithm randomly tries new actions in order to find a better one. The cost of trying new actions is that the agent could get a lower reward.

Consider the following example that illustrates the explore/exploit dilemma for humans. A person would like to go out to dinner. It almost goes without saying that the person would like to enjoy their dinner. The person faces a choice - they have been to restaurant A and had a good meal. However, they have never been to restaurant B, and it is possible that the food from restaurant B is better than the food from restaurant A. In this scenario, the quality of the meal is what determines the reward. Going to restaurant A and having a good meal would be choosing to exploit the current policy. Going to restaurant B would be choosing to explore a new action.

C. Q-Learning

In order to be able to compare policies and determine which performs better, there needs to be a way to quantify their performance, where performance is thought of as how well a policy selects actions to perform. This performance value needs to be comparable between policies, so it can be determined which policy performs best.

Since a policy is what maps states to actions, a good way to quantify the performance of a policy is to start at a specified state, follow the policy for a set number of steps, and calculate the total reward value that should be expected. This total reward value is called a quality value, but the name is usually shortened to Q value. A function to calculate these Q values, called a Q function, can be defined as

$$\boldsymbol{Q}^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}(\boldsymbol{\mathcal{R}} | \boldsymbol{s}, \boldsymbol{a}, \pi) \tag{1}$$

which gives the expected reward by following a policy π. However, in order to learn the optimal π, future rewards are continually discounted by a discount rate γ, where $0 < \gamma < 1$. Therefore, equation 1 can be rewritten as

$$\boldsymbol{Q}^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{t+1}[\boldsymbol{r}_{t+1} + \gamma \boldsymbol{Q}^{\pi}(\boldsymbol{s}_{t+1}, \pi(\boldsymbol{s}_{t+1}))] \tag{2}$$

The effect of the discount rate is that policies will value immediate rewards more than rewards that could come very far in the future. The smaller the discount rate, the more immediate rewards will be valued over future rewards.

Watkins showed that Q-learning, which is the process of learning these Q values, will converge to the optimal policy as long as all actions are repeatedly tested in all states and the actions themselves are discrete [12]. Q-learning essentially works by trying actions and updating the expected reward for that given state-action pair. Once this has been done many times for every state-action pair, the best action can be chosen by iterating over all the possible actions for a given state and choosing the one that has returned the highest Q value in its simulations.

An algorithm for Q-learning from Sutton and Barto is shown in Algorithm 1 [2]. In this implementation, α represents the learning rate, which is a value between 0 and 1. A higher learning rate allows for faster learning. A learning rate of 0 means no learning will occur, since the Q-values will never be updated. As mentioned earlier in this section, γ represents the discount factor. max_A represents the maximum reward attainable in the next state, S'.

Algorithm 1: Q-Learning Algorithm [2]

```
1  Initialize Q($s, a$) arbitrarily
2  for every episode do
3      Initialize $S$;
4      for each step do
5          Choose $A$ from $S$ using policy
             derived from $Q$;
6          Take action $A$, observe $R$, $S'$;
7          $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma max_A$
             $Q(S',A') - Q(S,A)]$;
8          $S \leftarrow S'$;
9      end
10     until $S$ is terminal;
11 end
```

D. Double Q-Learning

While Q-learning learns a single Q function, Double Q-learning, which was introduced by Hasselt, learns two Q functions [13]. Since there are two Q functions, the learning process must train them together - training them separately would be the same as normal Q-learning. Consider the two Q functions as Q_1 and Q_2. If Q_1 is used to choose an action, Q_2 is used to calculate the expected future reward in the update step, and vice versa if Q_2 was used to choose an action.

The benefits of double Q-learning may not be obvious. Normal Q-learning can struggle to converge on an optimal policy if the rewards from state-action pairs are stochastic. An example of this is if a state-action pair usually has a negative reward but occasionally will return a positive reward. Double Q-learning will be able to account for this stochastic reward and understand that the expected reward is negative even though it will occasionally be positive, while normal Q-learning will not. The drawback of double Q-learning is that it could underestimate the maximum expected reward, whereas normal Q-learning can overestimate it.

An algorithm for double Q-learning as described by Sutton and Barto is defined in algorithm 2, where the parameters used to update the Q-values in lines 7 and 8 are defined similarly to in the Q-learning implementation in algorithm 1 [2].

E. SARSA

The SARSA algorithm, short for State-Action-Reward-State-Action, in Reinforcement Learning is different from the standard Q-Learning algorithm in that SARSA is an online approach. This means SARSA allows the agent to approach the optimal policy at each time step without waiting for the algorithm to converge.

SARSA algorithm pseudocode as described by Sutton and Barto [2] is explained in algorithm 3. The parameters used in the update process (line 8) is the same as the parameters used in the Q-learning algorithm.

Algorithm 3: SARSA Algorithm [2]

```
1  Initialize $Q(s, a)$ arbitrarily;
2  for every episode do
3      Initialize $S$;
4      Choose $A$ from $S$ using policy derived
         from $Q$;
5      for each step do
6          Take action $A$, observe $R, S'$;
7          Choose $A'$ from $S'$ using policy
             derived from $Q$;
8          $Q(S,A) \leftarrow Q(S,A) + \alpha[R +$
             $\gamma Q(S',A') - Q(S,A)]$;
9          $S \leftarrow S'$;
10         $A \leftarrow A'$;
11     end
12     until $S$ is terminal;
13 end
```

F. Actor-Critic Method

Actor-Critic RL algorithms combine value functions and policy algorithms in order to stabilize the learning process. A value function maps an action/state to a single value, and one of the most common examples is the Q function (see section II-C). On the other hand, a policy algorithm tries to learn a mapping from states to the next action to perform. For Actor-Critic algorithms, the actor is a policy

Algorithm 2: Double Q-Learning Algorithm [2]

```
1  Initialize $Q_1(s,a)$ and $Q_2(s,a)$ arbitrarily;
2  for every episode do
3      Initialize $S$;
4      for each step do
5          Choose $A$ from $S$ using policy derived from $Q_1$ and $Q_2$;
6          Take action $A$, observe $R$, $S'$;
7          With 0.5 probability:
             $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha(R + \gamma Q_2(S', argmax_a Q_1(S',a)) - Q_1(S,A))$;
8          else $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha(R + \gamma Q_1(S', argmax_A Q_2(S',A')) - Q_2(S,A))$;
9          $S \leftarrow S'$;
10     end
11     until $S$ is terminal;
12 end
```

function that determines the next action. The critic function is then a value function that learns Q values. The critic function receives the state and the reward from the previous action as inputs, and its output is then sent as an input to the actor. The actor uses the state and the value from the critic as inputs. Since the critic function is learning the Q values, it is essentially telling the actor how good its decisions are. Both the actor and critic can be written as neural networks and they both use the temporal difference (TD) error calculated by the critic for updates. Figure 2 illustrates the actor-critic method.

A variation of the Actor-Critic method is the Advantage Actor-Critic (A2C) method. The basic difference between the Actor-Critic method and A2C is that the critic in Actor-Critic is learning Q values, while the critic in A2C is learning advantage values. Q values can be thought of as the total reward for taking a specific action in a given state. Advantage values can be thought of as how taking a specific action in a given state compares to the average value for that state.

Another variation of Actor-Critic methods is the Asynchronous Advantage Actor-Critic (A3C) method, which was introduced by Mnih et. al [14]. This method builds upon A2C, as the name suggests. With this approach, simulations are run in parallel and update a centralized network, which accounts for the addition of asynchronous in its title. The ability to run multiple simulations in parallel decreased the needed training time, but did not improve the results when compared to A2C methods.

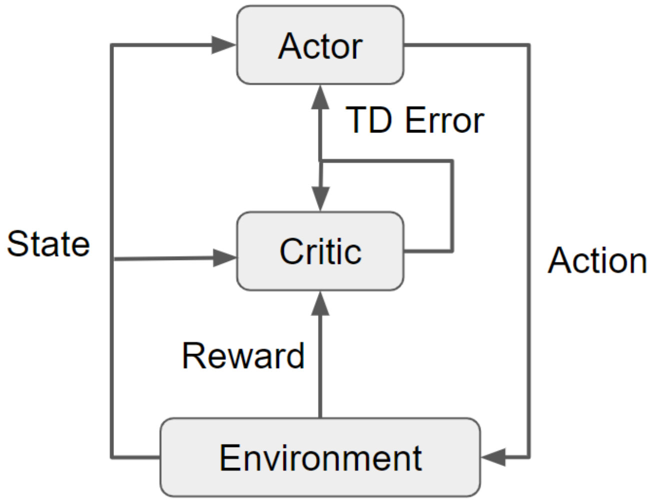


Fig. 2. A visualization of the actor-critic method indicating how the actor and critic interact with each other and the environment. Figure based from Sutton and Barto [2].

G. Deep Q Network

Q-learning, as described in section II-C, has a few major drawbacks. First, it is limited to a discrete action space. It might be possible to partition a continuous action space into a discrete action space, but this might not be acceptable if precise control is required. The second issue is Q-learning's requirement to explore the complete state-action space. Any RL algorithm that has many elements to its state will have a very large state space, and the same holds for action spaces. The state-action space can quickly become so large that it is impossible to explore the state-action space in a reasonable amount of time. The state-action space could also be so large that it exhausts a computer's memory.

The solution to this problem lies in Q-function approximation, where a complete mapping of states to actions is no longer computed explicitly, but approximated. Deep Neural Networks (DNNs) are universal function approximators, and therefore are a possible way to approximate Q-functions.

Deep Q Networks (DQN) were first implemented to beat expert humans in Atari 2600 games [15]. This new approach generalizes the approximation of the Q-value function rather than remembering the solutions. A deep neural network along with Q-Learning forms deep Q-Networks which overcome unstable learning by experience networks, target networks, clipping rewards and skipping frames [16]. The equation for DQN's can be written as

$$\begin{aligned} Q(s_t, a) &\leftarrow Q(s_t, a) \\ &+ \alpha[(r_{t+1} + \gamma Q(s_{t+1}, p) - Q(s_t, a))] \end{aligned} \tag{3}$$

H. Deep Deterministic Policy Gradients (DDPG's)

Q-learning when combined with deep learning (DQN) has given quite successful results however the action space represented by those cases has always been discrete. Various physical control tasks require the implementation of RL algorithmns on continuous action space and thus discretizing the action space too finely results in a huge action space and convergence becomes extremely difficult. Deep Deterministic Policy Gradients (DDPG), first introduced by Lillicrap et al. [17], is based on the actor critic architecture. The actor is used to tune the parameter $\boldsymbol{\theta}$ for the policy function in order to decide the best action for a particular state. The policy function is given by -

$$\pi_\theta(s, a) = P[a|s, \theta] \tag{4}$$

The critic is used for evaluating the policy function estimated by the actor according to the temporal difference error. The deterministic policy gradient is the expected gradient of the action-value function and thus the deterministic policy gradient can be estimated more efficiently than the usual stochastic policy gradient. To ensure adequate exploration, Silver et al. in [18] introduced an off-policy actor-critic algorithm that develops a deterministic target policy from an exploratory behavior policy and demonstrates that that deterministic policy gradient algorithms can outperform their stochastic counterparts in high-dimensional action spaces. DDPG can only be used for environments with continuous action spaces, it can be thought of as being Deep Q-Learning but with continuous action spaces. The pseudocode is displayed in Algorithm 4.

I. Inverse Reinforcement Learning

Inverse Reinforcement Learning [19] re-imagines the learning process by first observing the task being completed. From these observations, the agent learns the reward. In typical reinforcement learning, the reward function is known and fixed. However, in human behavior, a reward function may not always be known. Thus, the goal of inverse reinforcement learning is to find the reward function that describes the observed behaviors.

J. Experience Replay

Many machine learning algorithms require having large data sets to train an algorithm. One of the benefits of RL is that preexisting data sets are not needed, which is especially beneficial for work being done where these large data sets do not exist. Instead, a RL algorithm is given a way to interact with its environment and and is presented with a reward signal to guide its learning.

Lin proposed a way to accelerate learning titled experience replay [20]. With experience replay, experiences are saved and can be replayed to the RL algorithm after policy updates have been made. An experience consists of the current state S_t, a chosen action A_t, the next state S_{t+1}, and the reward given R_{t+1}, which can be saved in a tuple as $(S_t, A_t, S_{t+1}, R_{t+1})$. This experience can

be re-presented to the RL algorithm multiple times as it learns since the experience is not dependant on the policy - the resulting state and reward for a given state-action input does not change. However, if the *environment* changes over time, then it is possible that the saved experience is no longer useful.

These experiences can be sampled and replayed randomly for an off-policy algorithm such as DQL. Random batching of DQL updates can improve the variance of updates because any one batch is not overfitting to a specific sequential set of events. This results with a more stable learning process [2].

Algorithm 4: DDPG Algorithm [17]

```
1  Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^M)$ with weights $\theta^Q$ and $\theta^\mu$.
2  Initialize target network $Q_0$ and $M_0$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$.
3  Initialize replay buffer $R$
4  for episode=1,M do
5      Initialize a random process $N$ for action exploration.
6      Receive initial observation state $s_1$
7      for t=1,T do
8          Select action $a_t = \mu(s_t|\theta^\mu) + N_t$ according to the current policy and exploration noise.
9          Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$.
10         Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
11         Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
12         Set $y_i = r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
13         Update critic by minimizing the loss: $L = 1/N \sum_i (y_i - Q(S_i, a_i|\theta^Q))^2$
14         Update the actor policy using the sampled policy gradient.
15         Update the target networks.
16     end
17 end
```

III. RL Lane Maneuvering

Lane maneuvering is an important and sometimes dangerous task necessary for the safe and efficient operation of vehicles and vehicle networks. It combines all aspects of driving, as well as lateral and longitudinal operations and considerations. Agents must be trained to perform the most optimal policy for themselves in the context of surrounding vehicles, or may alternately optimize its policy at a network scale to consider the impact of its decisions on nearby vehicles and traffic networks. There may exist a malicious driver or adversary who does not allow another vehicle to pass, exit, merge, or change lanes. There are timing considerations, in which the speed and distance of the neighboring vehicles in the target lane must be calculated so that a crash does not ensue when maneuvering lanes. There may be road obstacles such as potholes or construction. The more difficult the lane operation, the more likely that a human error will occur. This section summarizes existing work towards using reinforcement learning as a means of automated of lane maneuvering. We divide the section into categories of different lane maneuvers, including merging, centering, overtaking, and changing lanes, and evaluate the application of RL to addressing such maneuvers.

A. Merging

In most Advanced Driving Systems (ADS), automated driving is typically limited to restricted-access freeways, that is, the transition from manual to automated modes takes place only after the ramp merging process is completed manually by a human driver. One major challenge to extend the automation to ramp merging is that the automated vehicle needs to optimize long-term objectives (e.g. successful lane merging) with near-term objectives (e.g. minimizing jerks). Moreover, the merging process involves interactions with other vehicles whose behaviors are difficult or impossible to predict, but whose behaviors also influence the merging vehicle's optimal actions. To tackle this complicated control problem, Wang and Chan proposed to apply Deep Reinforcement Learning techniques for finding an optimal driving policy by maximizing the long-term reward in an interactive environment [21]. They use Long Short-Term Memory (LSTM) architecture to model the environment and to learn the internal relations between the ego vehicle and other surrounding vehicles. An internal state representation from LSTM at each time step is then fed into a Deep Q-network for action selection. To avoid local optima and divergence problems, Q-network is updated by an experience replay and second target Q-network. In this way, an interactive merging policy is learned.

Deep Q-learning uses neural networks, parameterized by θ, to approximate the Q-function. Q-values, denoted as $Q(s, a; \theta)$, can be used to get the best action for a given state. To guarantee an optimal action and fast convergence, the Q-function approximator was designed as a quadratic function, and its parameters were selected using neural networks.

To sum up Q-learning process was divided into: (1) The Q-value approximation for action selection. (2) Q-network update,when Q-network parameters θ and θ^- are updated based on the error between predicted and target Q-Values.

For their implementation, the scenario was simulated on data collected from a section of US Interstate Highway I-80. This paper considers the typical scenario with three vehicles (the merging vehicle, the gap lag vehicle and the gap front vehicle). For state representation, 9 variables were considered. For the merging vehicle, the authors used 5 variables to describe its driving state namely speed, position, heading angle, and distances to the right and left lane makings of the current lane. For the other two vehicles, we assume we can only observe their speeds and positions thus making the total considered variable count as 9. The action is the acceleration and the steering angle taken by the merge vehicle. The data for training is obtained from real world driving scenarios using a camera.This paper successfully proposes a Deep Reinforcement Learning architecture for learning an on-ramp merge driving policy.

D. Isele developed a stochastic rule-based merge behavior in [22]. The paper starts by developing the interactive decision making process for an AV as a stochastic game. As the number of players and the actions increase the game trees grow exponentially ,the paper thus discusses how to limit the branching factor of both the agent being controlled (the ego vehicle) and the other traffic participants as their number increases.To reduce the number of ego-agent actions, the actions are decomposed into a sub goal selection which in turn is done using a probabilistic tree search.The decision making was broken down as follows: 1) Select an intention class based 2) Identify all the traffic participants which will be interacting with te ego vehicle 3) Predict their agents' intentions 4) Sample and evaluate the ego intentions and 5) take action, observe, and update probability models.

The model was experimented with cars having a specified length and a width. The lane width was $3.7m$. Traffic was generated with a mean gap width between cars corresponding to $\{2.4, 4.8, 9.6\}m$ with added noise drawn uniformly from the range $\pm\{0.4, 0.8, 1.6\}$ respectively. It was tested on 20 simulations varying the tree depth from one to three layers deep. Each experiment was initialized with the same random seed and the agents encounter identical situations. The entire experiment was then repeated multiple times with different random seeds to generate confidence intervals related to the observed change.

While this approach demonstrates a successful method for interactive merging in simulation, there is more investigation required to run safely on an actual vehicle.

Lin et al. [23] propose using DDPG for the on-ramp merging task in an automated vehicle. They use Simulation of Urban Mobility (SUMO) to create the merging environment. The agent vehicle uses sensors (lidar, radar, etc.) to sense surrounding vehicles within a radius of 100 meters. They train the merging vehicle for 1 million simulation time steps. During testing, there was only one collision, out of the 16,975 episodes that were observed. Out of the 16,975 testing episodes, the agent vehicle successfully merged in 16,024 of them.

B. Lane Centering

Lane centering may be thought of as a simple task; where the driver of the car stays in the center of the lane. This task may also be called lane keeping interchangeably. This task is very important for vehicles, for purposes including collision avoidance.

A simple implementation and approach to autonomous lane keeping is shown by Lee et al. [24], where they use a slight variation of the Q-learning algorithm, an approximate Q-learning algorithm proposed by Sutton and Barto [2]. The algorithm is shown in 5. The approximate Q-learning algorithm is useful when there are too many states and actions to learn the Q-function. It can be used for faster training. They use their own function approximator and stochastic gradient descent to update the Q values. In other words, they generalize unvisited states from already visited states.

Zheng et al. [25] use model-free reinforcement learning to develop the agent vehicle's lateral control strategy and to stay within the center of a lane.

Algorithm 5: Approximate Q-learning Algorithm [2], used in lane keeping task by Lee et al. [24].

```
1  Initialize $Q(s,a)$ arbitrarily;
2  for every episode do
3      Initialize $S$;
4      Choose $A$ from $S$ using policy derived from $Q$;
5      for each step do
6          Take action $A$, observe $R, S'$;
7          Choose $A'$ from $S'$ using policy derived from $Q$;
8          $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$;
9          $S \leftarrow S'$;
10         $A \leftarrow A'$;
11     end
12     until $S$ is terminal;
13 end
```

Their proposed algorithm, shown in algorithm 6, implements an optimal feedback controller for the agent. The RL algorithm is tested in simulation to demonstrate lateral control for lane keeping. With system parameters assigned as $m = 1150kg$, $I_z = 2000kg/m^2$, $r_s t = 1.78$, $m_x = 80km/h$, $l_f = 1.27m$, $l_r = 1.37m$, $C_{af} = C_{ar} \approx 8000N/rad$, $Q = diag(2,0,0,0.2)$, $r = 1$, X^2= [0 1 0 0], X^3=[0 0 1 0], and X^4=[0 0 0 1]. Under the road curve shown in Figure 3, Figure 4 shows the simulation results for each system state trajectory using Algorithm 6. We can see that they converge to zero quickly.

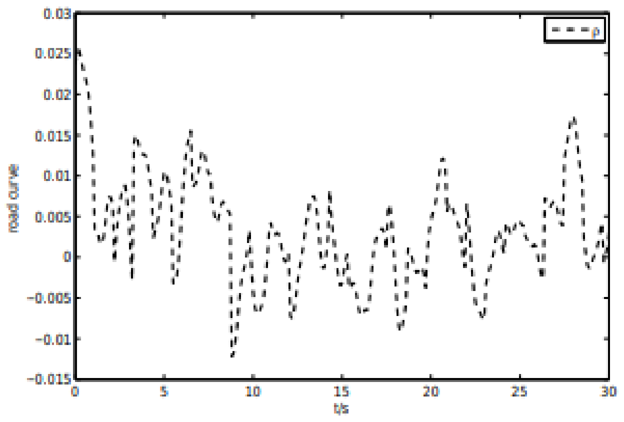


Fig. 3. Road curve in simulated environment in [25].

Algorithm 6: Model-Free RL Algorithm for optimal lane-keeping [25]

```
1  Start with an initial stabilizing feedback gain $K_0$. Choose parameters $Q, r$, time interval $T$, and convergence bound $n$.
2  Let $X^1$=0. Compute $X^l$ for $l$=2,3,4 with $C_1 X^l$=0. Collect generated data $\Theta_i$ and $Xi_l$ under the initial controller for each $X^l$.
3  Solve $P(j)$, $K^{j+1}$, and $(C_1 + A_1 X^l)P^j$ with index $j = 0, 1, \ldots$.
4  Let $j = j + 1$. Update $P(j)$, $K^{j+1}$, and $(C_1 + A_1 X^l)P^j$ until $||K^j - K^{j-1}|| \leq n$ with $j* = j$.
5  Determine $A_1 X^l$ for $l$=1,2,3,4, and compute parameter $\alpha_l$, $U*$, and $X*$.
6  Obtain the optimal controller $\delta*$= $-K^{j*}x + (U * + K^{j*} X*)p$.
```

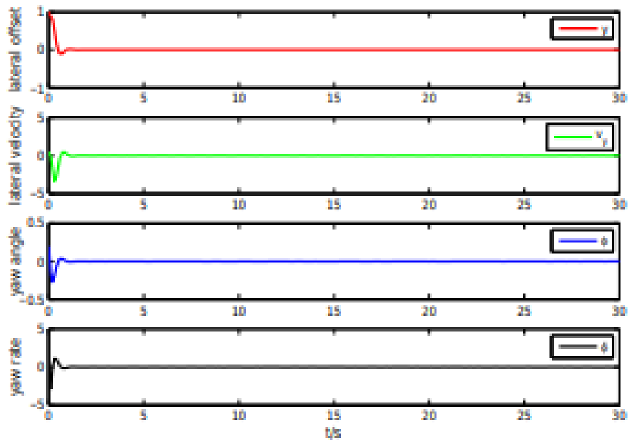


Fig. 4. System state trajectories converge in simulated environment given road curve shown in Figure 3, proposed by Zheng et al. [25]

Sallab et al. [26] provide their own deep reinforcement learning framework to solve lane maneuvering tasks in complex environments, such as varying road curves with other vehicles present. Their framework is illustrated in Figure 5. It provides training of a deep neural network for AVs. The framework is tested only in simulation, specifically in The Open-source Racing Car Simulator (TORCS). The results show that their proposed framework is successful in lane centering.

C. Overtaking

Ngai and Yung [27] envision overtaking as a multiple-goal reinforcement learning (MGRL) task

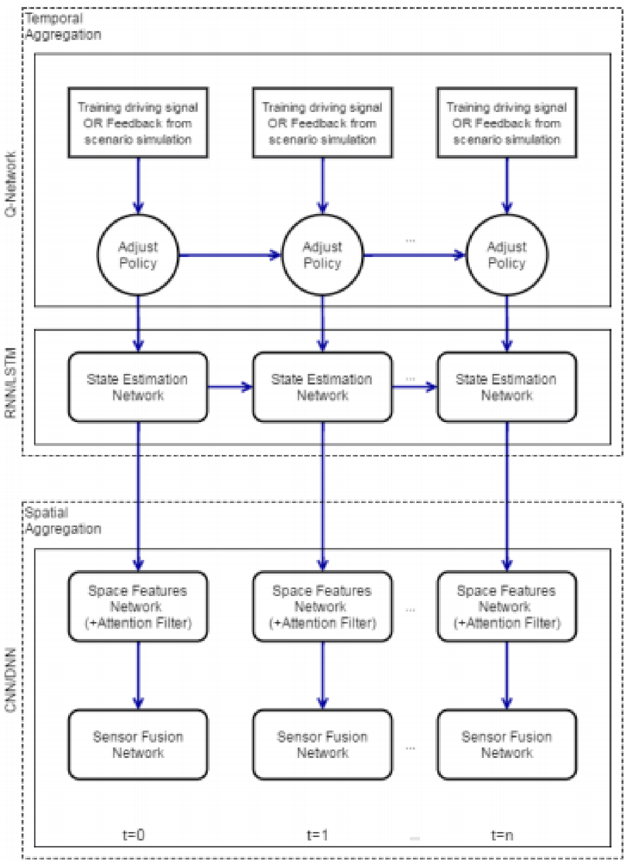


Fig. 5. Deep Reinforcement Learning framework proposed by [26]

with seven goals: collision avoidance, target seeking, lane following, slow lane, fast lane, steady speed, and steady steering angle. The agent vehicle uses Double Q-learning for collision avoidance and Q-learning for target seeking and learning appropriate reactions to maneuvers. They test their approach in simulation, after training it in simulation for 1000 episodes. Training showed that after the first 400 episodes, the vehicle had no further collisions in the remaining 600 episodes. They showed that the agent vehicle was able to complete the overtaking task successfully in four cases: (1) straight road with other vehicles moving at a constant speed, (2) straight road with other vehicles moving at variable speeds, (3) curved road with other vehicles moving at variable speeds, and (4) a complex maneuver, where the road curves twice, and vehicles move at variable speeds and spawn at different sections of the road.

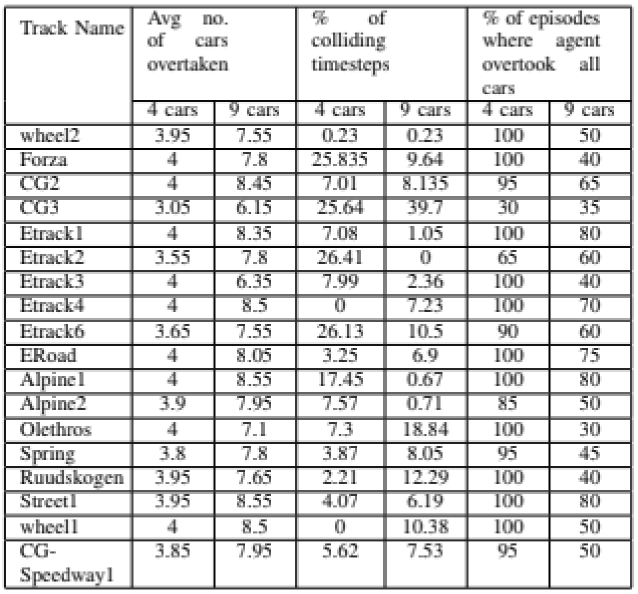
Kaushik et al. [11] use Deep Deterministic Policy Gradients to learn how to overtake other vehicles in a highway simulation. They show that the agent successfully learns smooth lane maneuvers where the agent is able to overtake all other vehicles, no matter the path of the highway or number of vehicles in the environment. DDPGs are used in this approach for the agent to learn actions in curriculum learning manner [28], where the agent first learns a simple task and then moves on to learn a more difficult task. The authors train the agent vehicle in lane keeping, and then move on to learn the task of overtaking. Their results show that without curriculum learning, the agent vehicle collides with neighboring vehicles instead of safely overtaking them, and does not stay in the center of the lane. They test their combined DDPG with curriculum learning approach in simulation of 18 different highway tracks, where there are either 4 or 9 cars on the road (not including the agent vehicle). Their approach is effective, regardless of the track. The agent is able to overtake all 4 vehicles on average after 20 episodes. On simulations with 9 neighboring vehicles, the agent overtakes about 7 vehicles on average. Their results are shown in Figure 6. The second column of their results show a low percentage of collisions between the agent vehicle and neighboring vehicles. Kaushik et al. display the effectiveness of combining a reinforcement learning algorithm with the idea of curriculum learning. Rather than training for the complex task from scratch, training for a simple task first will yield desirable results in a shorter amount of time. Their successful overtaking task is shown in Figure 7.

Zheng et al. [29] use a highway simulation to test their solution to the overtaking problem, implementing the Least Square Policy Iteration (LSPI) algorithm. They also use linear approximation to approach the Q-function. Their approach is only tested in simulation, and they show that the agent vehicle is able to successfully overtake vehicles in traffic. However, not many details are given about simulation training or testing, such as how many episodes until the policy converged or how many collisions occurred.

You et al. [30] models the overtaking problem in a highway traffic simulation, and uses the Q-learning algorithm. After about 6000 training episodes, the policy converges.

D. Changing Lanes

Changing lanes is another complicated steering problem for AVs that can be handled with RL.



Track Name	Avg no. of cars overtaken		% of colliding timesteps		% of episodes where agent overtook all cars	
	4 cars	9 cars	4 cars	9 cars	4 cars	9 cars
wheel2	3.95	7.55	0.23	0.23	100	50
Forza	4	7.8	25.835	9.64	100	40
CG2	4	8.45	7.01	8.135	95	65
CG3	3.05	6.15	25.64	39.7	30	35
Etrack1	4	8.35	7.08	1.05	100	80
Etrack2	3.55	7.8	26.41	0	65	60
Etrack3	4	6.35	7.99	2.36	100	40
Etrack4	4	8.5	0	7.23	100	70
Etrack6	3.65	7.55	26.13	10.5	90	60
ERoad	4	8.05	3.25	6.9	100	75
Alpine1	4	8.55	17.45	0.67	100	80
Alpine2	3.9	7.95	7.57	0.71	85	50
Olethros	4	7.1	7.3	18.84	100	30
Spring	3.8	7.8	3.87	8.05	95	45
Ruudskogen	3.95	7.65	2.21	12.29	100	40
Street1	3.95	8.55	4.07	6.19	100	80
wheel1	4	8.5	0	10.38	100	50
CG-Speedway1	3.85	7.95	5.62	7.53	95	50

Fig. 6. Kaushik et al.'s results using DDPG and curriculum learning in overtaking task. [11]

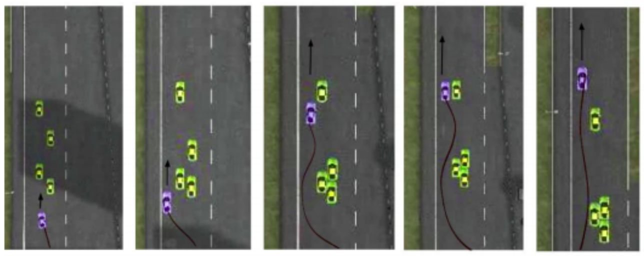


Fig. 7. Results from Kaushik et al. using DDPG and curriculum learning in overtaking task. [11] The purple car is the agent, yellow cars are the neighboring vehicles, and the red line shows the agent's successful trajectory in overtaking all four yellow cars.

Changing lanes is a difficult control problem for many reasons, which we will separate into two sub-problems: deciding when to change lanes and executing the lane change.

1) Lane Change Decisions: To decide when to change lanes, the AV first needs to determine if it is an acceptable time to change lanes. In its most basic form, the AV must check the adjacent lane to see if there is enough room to safely fit between other cars. This is difficult, as other vehicles can be moving at different speeds and can change speeds at any time. In a more advanced form, the vehicle could look further backwards in the desired lane to check if it should let a faster moving vehicle pass before changing lanes, which would both optimize traffic flow and avoid inconveniencing another vehicle.

Mirchevska et al. trained a DQN to make safe lane change decisions for highway driving [31]. Their argument for safety was that they had a safety verification algorithm that calculated the necessary safe distance to keep from the next car to allow for stopping distances in case of emergency braking. The reward function for their RL algorithm was maximized when the vehicle drove at the desired velocity, and since other vehicles in the simulation drove slower than the AV's desired velocity, the RL algorithm learned to change lanes in order to maximize the time it could drive at the desired speed. Another one of the authors' goals was to have a minimal state representation to accelerate learning, which they achieved by defining their state as the longitudinal distance between the agent and the next leading and trailing vehicle in every lane, as well as all of the vehicles velocities (including the agent). The state space is therefore continuous, but the action space was discrete and only consisted of deciding to stay in the current lane, change lanes to the left, or change lanes to the right. This highlights the difference between lane change *decision* algorithms vs lane change *execution* algorithms. Their results showed their RL algorithm could achieve higher average velocities than a rule-based agent.

In a work that has similarities to the work by Mirchevska et al., Hoel et al. used a DQN to make lane change decision for highway driving [32]. However, they had multiple agent variants. A first agent only made lane change decisions, while a second agent was able to make lane change and speed decisions. The second agent's action space consisted of telling the vehicle to change lanes to the left, change lanes to the right, accelerate ($+2m/s^2$), partial brake ($-2m/s^2$), full brake ($-9m/s^2$), or maintain the current speed. For the first agent, speed was controlled by the IDM algorithm [33].

Both agents performed well, but the second agent was able to achieve higher performance ratings since it was also able to maximize speed within its specified limits. The authors also tested how the agents performed and trained differently when fully connected neural networks (FCNN) were used as compared to convolutional neural networks (CNN). The convolutions were used to process the state values that represented the other vehicles in the simulation, whose values consisted of their speed, position, and lane with respect to the ego vehicle. The CNN agents trained significantly faster than the

FCNN agents, which is not surprising considering a FCNN has significantly more weights to train than a CNN. However, even after training the FCNN agent that controlled both lane and speed changes for magnitudes longer than the CNN agent, the FCNN agent was not able to perform as well as the CNN agent.

It should be recognized that the simulations by Hoel et al. were quite rigorous. The ego vehicle would spawn with 8 other vehicles randomly around it, and the other vehicles were specifically designed to randomly and abruptly change their speeds during simulations. Furthermore, the vehicles in front of the ego vehicle would have a slower speed than the ego vehicle, and the vehicles behind the ego vehicle would have a faster speed than the ego vehicle. This way, the ego vehicle was required to understand the speeds of the vehicles behind it so it does not cause an accident by cutting off another vehicle. Their simulations would have been even more rigorous if they had the other vehicles randomly change lanes on top of changing their speeds.

After training and evaluating their agents on highway simulations, Hoel et al. retrained the same agents on two-lane, bidirectional roads. In these simulations, the ego vehicle was placed $50m$ behind a slower vehicle, and two oncoming vehicles were randomly spawned between 300 and $1100m$ ahead. In these experiments, they also trained an agent that only controlled lane changes as well as an agent that controlled lane changes and speed. The CNN variants of these algorithms were able to achieve better performance than IDM, but note that their IDM baseline did not change lanes.

2) Lane Change Execution: Executing a lane change maneuver is a delicate balance of a vehicle's dynamics, the road layout, and the other vehicles on the road. The primary goal is to safely move the vehicle to the next lane, and secondary goals could be to minimize the duration of the maneuver while also minimizing the accelerations subjected to the passengers. Note that the last two goals conflict with each other. A vehicle could change lanes quicker if it was allowed to make aggressive movements, and a vehicle could minimize accelerations if it was allowed to perform a lane change in 10 seconds as compared to 2 seconds.

In the work done by Wang et al., their goal was to implement a lane changing RL algorithm [34]. Their algorithm had a continuous action space that used DDPG in a model-free dynamic driving environment. A mid-level lateral controller was used to transform more generalized steering commands to values that can be applied by steering actuators, which could be a common setup in an AV with modular software. Their reward function had penalties for yaw rate and lane change duration (time), and a much larger penalty for driving past the lane the AV was trying to move into. They trained their RL algorithm for $5,000$ iterations and were able to get good results, some of which can be seen in Figure 8.

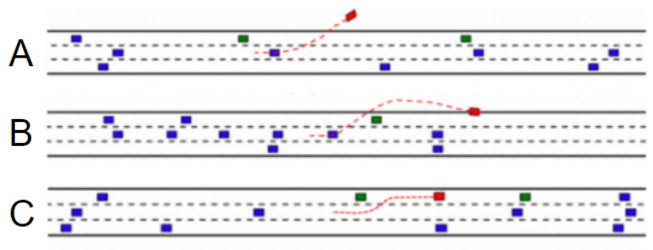


Fig. 8. Visualization of lane changing performance at different iterations of training. A has been trained the least, B has had some training, and C has had the most training. It is clear that the algorithm made progress from checkpoint A to B, since the vehicle attempts to correct back into its lane. Checkpoint C demonstrates a successful lane change. Figure taken and edited from [34].

A natural way that the work done by Wang et al. could be expanded would be to train a similar algorithm in a similar environment, but have the other vehicles in the environment randomly change lanes. In their work, the other vehicles remained in the same lane, but allowing them to randomly move would be a more challenging problem, and one that is closer to the real world.

Chandramohan et al. [35] uses a DQN to control an AV in a multi-lane highway simulation. They use SUMO as their simulation environment, implementing their DQN using SUMO's traffic control interface (TraCI). They choose DQN for this scenario due to the difficulty in maintaining a Q table. The agent is trained in a highway environment with either 3 or 2 lanes (randomly chosen per episode) over $7,000$ episodes. All vehicles follow traffic rules and maintain a minimum distance from the vehicle in front of them. They did not provide many details about the testing phase, other than noting that it was tested in random scenarios successfully over 500 episodes.

Folkers et al. [36] use proximal policy opitimization (PPO), a version of the actor-critic method,

Algorithm 7: Proximal Policy Optimization, used in lane changing execution task by Folkers et al. [36].

```
1 Initialize $\pi_\theta$ and $V^\theta$;
2 while not converged do
3     Set $\pi_{\theta_0} \leftarrow \pi_\theta$;
4     Generate a rollout set $M$ following $\pi_{\theta_0}$
        and compute $A_t$ for each data point;
5     for $K$ steps do
6         Draw a random batch of size $m$ from
            $M$;
7         Update $\pi_\theta$ and $V^\theta$ using a stochastic
            gradient descent algorithm with
            backpropagation and cost functions
            $C_\pi$ and $C_V$;
8     end
9 end
```

to efficiently learn and execute the lane changing maneuver. The pseudocode for their PPO algorithm is shown in Algorithm 7, where V is the state-value function. The approach is trained and tested in $10,000$ episodes each, with moderate success. The agent vehicle has a 90.7% success rate, with 6.1% episodes ending in collision, and 3.2% ending with the driver not completing the lane change in the given time frame.

IV. Future Research

Despite the successes of RL, many problems need to be addressed before these techniques can be applied to a wide range of complex autonomous vehicle problems. A deeper integration of RL techniques for AVs with other traditional AI approaches (tree search, etc.) promises benefits such as better sample complexity, generalisation and interpretability [37] and hence can be expanded for implementation on intelligent lane maneuvering. RL when combined with multiple sensor data fusion can result in highly efficient lane maneuvering.[38] Proposes a safe overtaking maneuver where the vehicle equipped with the system determines in real time the current driving mode and performs the least restrictive safe control actions. This system when combined with RL algorithmns where the data from multiple state of the art sensors is fed into the network can prove to be more efficient and safe that the system or RL working alone.

We find that most reinforcement learning approaches in this area only test within simulation. Even if tested at scale, it would be greatly beneficial for RL techniques to be tested in a real world environment. Additionally, multiple reinforcement learning algorithms could be used to control different aspects of an automated vehicle.

A. Proposed Ideas

1) Dynamic Obstacles: More work should be done with dynamic obstacles, especially concerning edge cases that an AV will be expected to handle. For example, if an AV is attempting to change lanes to the left, have a vehicle from two lanes to the left try to merge into the same lane as the AV at the same time. Another example is exposing a lane centering RL algorithm to situations where a secondary vehicle begins to merge into the same lane as the AV. In this case, drifting out of a lane should have less of a penalty than getting in an accident, but it should not do so if drifting out of its lane could cause a more severe accident. Determining which accident is less severe is a different problem altogether, but immediate progress could still be made by making accident severity assumptions and modeling these assumptions as part of a reward function.

V. Conclusion

Table I summarizes the reinforcement learning approaches critiqued throughout this survey. The table details the reference, the algorithm used, the environments the approach tested on, and the number of episodes experienced before complete training. It is worth describing the approaches in this manner so that researchers may have a clear idea of where to start or note which algorithm(s) performs well or is used frequently. With such a table, we can note approximately how much computation would be needed to complete training if one wanted to reevaluate an approach.

Improvements to this survey could be made by continuing to gather references that fall into the lane maneuvering categories. Additionally, the simulation of the same task repeated using the different algorithms would help to determine which algorithm should be used to learn a specific task.

TABLE I
REINFORCEMENT LEARNING APPROACHES WITHIN MANEUVERING CATEGORIES

Reference	Algorithm	Tested Environment	Num. of Episodes for Complete Training
MERGING			
Isele et al. [37]	DQN	simulation, real vehicle	7000
Wang and Chan [21]	DDPG	simulation	N/A, Preliminary work
Lin et al. [23]	DDPG	simulation	Not reported
LANE CENTERING			
Y. Zheng et al. [25]	Double DQN	simulation	N/A, preliminary work
Lee et al. [24]	Q-learning	simulation	140
Sallab et al. [26]	DQN	simulation	N/A, preliminary work
OVERTAKING			
Ngai and Yung [27]	double Q-learning	simulation	400
Kaushik et al. [11]	DDPG	simulation	20
You et al. [30]	Q-learning	simulation	6000
R. Zheng et al. [29]	LSPI	simulation	Not reported
CHANGING LANES			
Mirchevska et al. [31]	DQN	simulation	NR
Hoel et al. [32]	DQN	simulation	600K
Wang et al. [34]	DDPG	simulation	5000
Chandramohan et al. [35]	DQN	simulation	7000
Rehder et al. [39]	inverse RL	simulation	60
Saxena et al. [40]	PPO	simulation	Not reported
Makantasis et al. [41]	DDQN	simulation	Not reported
Folkers et al. [36]	PPO	simulation, real vehicle	10K

REFERENCES

[1] S. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. Eng, D. Rus, and M. Ang, “Perception, planning, control, and coordination for autonomous vehicles,” *Machines*, vol. 5, no. 1, p. 6, 2017.

[2] R. S. Sutton, A. G. Barto, *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998.

[3] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[4] A. J. Snoswell and C. L. Snoswell, “On the use of reinforcement learning for testing game mechanics,” *Computers in Entertainment*, 2018.

[5] S. Shalev-Shwartz, S. Shammah, and A. Shashua, “Safe, multi-agent, reinforcement learning for autonomous driving,” *arXiv preprint arXiv:1610.03295*, 2016.

[6] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “Deepdriving: Learning affordance for direct perception in autonomous driving,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2722–2730, 2015.

[7] B. Zhang, Z. Mao, W. Liu, and J. Liu, “Geometric reinforcement learning for path planning of uavs,” *Journal of Intelligent & Robotic Systems*, vol. 77, no. 2, pp. 391–409, 2015.

[8] P. Wang, C.-Y. Chan, and A. de La Fortelle, “A reinforcement learning based approach for automated lane change maneuvers,” in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1379–1384, IEEE, 2018.

[9] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, “Applications of deep reinforcement learning in communications and networking: A survey,” *IEEE Communications Surveys & Tutorials*, 2019.

[10] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, “A survey of autonomous driving: Common practices and emerging technologies,” *arXiv preprint arXiv:1906.05113*, 2019.

[11] M. Kaushik, V. Prasad, K. M. Krishna, and B. Ravindran, “Overtaking maneuvers in simulated highway driving using deep reinforcement learning,” in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1885–1890, June 2018.

[12] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[13] H. V. Hasselt, “Double q-learning,” in *Advances in Neural Information Processing Systems*, pp. 2613–2621, 2010.

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

[15] 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.

[16] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, pp. 26–38, Nov 2017.

[17] 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.

[18] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 387–395, PMLR, 22–24 Jun 2014.

[19] A. Y. Ng, S. J. Russell, *et al.*, “Algorithms for inverse reinforcement learning.,” in *Icml*, vol. 1, p. 2, 2000.

[20] L.-J. Lin, “Self-improving reactive agents based on reinforcement learning, planning and teaching,” *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

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

[22] D. Isele, “Interactive decision making for autonomous vehicles

in dense traffic," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 3981–3986, Oct 2019.

[23] Y. Lin, J. McPhee, and N. L. Azad, "Decision-making and control for freeway on-ramp merging using deep reinforcement learning," 2019.

[24] J. Lee, T. Kim, and H. J. Kim, "Autonomous lane keeping based on approximate q-learning," in *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pp. 402–405, June 2017.

[25] Q. Zhang, R. Luo, D. Zhao, C. Luo, and D. Qian, "Model-free reinforcement learning based lateral control for lane keeping," in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, July 2019.

[26] A. Sallab, A. Mohammed, P. Etienne, and Y. Senthil, "Deep reinforcement learning framework for autonomous driving," 2017.

[27] D. C. K. Ngai and N. H. C. Yung, "A multiple-goal reinforcement learning method for complex vehicle overtaking maneuvers," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, pp. 509–522, June 2011.

[28] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, ACM, 2009.

[29] Rui Zheng, Chunming Liu, and Qi Guo, "A decision-making method for autonomous vehicles based on simulation and reinforcement learning," in *2013 International Conference on Machine Learning and Cybernetics*, vol. 01, pp. 362–369, July 2013.

[30] C. You, J. Lu, D. Filev, and P. Tsiotras, "Highway traffic modeling and decision making for autonomous vehicle using reinforcement learning," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1227–1232, June 2018.

[31] B. Mirchevska, C. Pek, M. Werling, M. Althoff, and J. Boedecker, "High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2156–2162, IEEE, 2018.

[32] C.-J. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2148–2155, IEEE, 2018.

[33] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[34] P. Wang, H. Li, and C. Chan, "Continuous control for automated lane change behavior based on deep deterministic policy gradient algorithm," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1454–1460, June 2019.

[35] A. Chandramohan, M. Poel, B. Meijerink, and G. Heijenk, "Machine learning for cooperative driving in a multi-lane highway environment," in *2019 Wireless Days (WD)*, pp. 1–4, April 2019.

[36] A. Folkers, M. Rick, and C. Büskens, "Controlling an autonomous vehicle with deep reinforcement learning," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 2025–2031, June 2019.

[37] D. Isele, A. Nakhaei, and K. Fujimura, "Safe reinforcement learning on autonomous vehicles," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–6, Oct 2018.

[38] J. Kocić, N. Jovičić, and V. Drndarević, "Sensors and sensor fusion in autonomous vehicles," in *2018 26th Telecommunications Forum (TELFOR)*, pp. 420–425, Nov 2018.

[39] T. Rehder, A. Koenig, M. Goehl, L. Louis, and D. Schramm, "Lane change intention awareness for assisted and automated driving on highways," *IEEE Transactions on Intelligent Vehicles*, vol. 4, pp. 265–276, June 2019.

[40] D. M. Saxena, S. Bae, A. Nakhaei, K. Fujimura, and M. Likhachev, "Driving in dense traffic with model-free reinforcement learning," 2019.

[41] K. Makantasis, M. Kontorinaki, and I. Nikolos, "A deep reinforcement-learning-based driving policy for autonomous road vehicles," 2019.