Behavioral Planning for Automated Vehicles using Deep Reinforcement Learning

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering
by Research

by

Meha Kaushik
201302001
meha.kaushik@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
October 2018

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "**Behavioral Planning for Automated Vehicles using Deep Reinforcement Learning**" by **Meha Kaushik**, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Prof. K Madhava Krishna

To My Parents for their unconditional love

Acknowledgments

I would like to express my most sincere gratitudes, to my Research Advisor, Prof. K Madhava Krishna. I thank him for giving me the chance to work on the problem of my interest. His feedback on the work and insights on problem solving helped me learn things which could not be found in any book. His experience is invaluable. Above everything, I thank him for having constant faith in me and trusting me with important decisions. I look up to him not only for his knowledge, but his hard work, his positive attitude and the way he cares for his students.

Robotics Research Center would always be an integral part of my life now. I met some brilliant minds here, brilliant yet humble, who never thought twice in offering help and advice. I am obliged to Vignesh Prasad and Anurag Ghosh for his valuable comments, feedback and guidance. I thank Phani Teja Singamaneni for answering my silliest doubts with extreme patience. I am lucky to have Nayan as a friend with whom I have shared this entire research journey. I thank Parijat Dewangan and Josyula for sharing their domain knowledge and Bharath G. and Nazrul Haque for introducing me to this wonderful workplace and to this field.

I thank my institute, IIIT-Hyderabad for providing this amazing platform to choose and work on the domain you like. This institute is the reason I met few very important people in my life. I am grateful to Shubham Ghiya and Ayush Jhunjhunwala for patiently bearing up with me in my ups and downs. I thank Rashi Shrishrimal for teaching me the importance of never giving up and trying harder each time you fail. I am grateful to have a friend like her in my life. I would also like to thank Akanksha, Vatika, Dhriti, Itisha, Khawad, Aditya Bohra, Sanatan Mishra, Prateek Pandey and Mahtab for making this journey memorable.

Above everyone else, I thank my Mother Mrs Sunita Sharma and Father Mr BK Kaushik for making me the person I am, for believing in me in the times I did not believe in myself. I thank my friends Suhani Chordia, Sanyam Jain and Bhavya Bhandari for the faith they always had in me. Lastly and most importantly, I thank Aastikta Sharma and Harshit Kaushik for the love and comfort, they are always providing. Without my parents and my dearest friends nothing would have been the same.

Abstract

Autonomous Driving is one of the hot areas of research in the current world. Deep Machine Learning and Computer Vision together have shown promising results in this field. A relatively new domain is Reinforcement Learning. RL is associated with behavior based learning. An RL agent learns from experiences, by exploring it action and state space.

In this work, a Deep Reinforcement Learning algorithm, Deep Deterministic Policy Gradient(DDPG) is used to learn various driving behaviors. DDPG is an off-policy, continuous control framework. It is the off-policy nature of this algorithm which enables it to efficiently explore the environment. Curriculum learning and Intrinsic Motivation are used to extract advantage of the off-policy nature.

Different behavior based agents are trained. The important ones being Overtaking in highways and Opportunistic agent in dense unstructured traffic scenes.

For overtaking in highways, the agent learns in a Curriculum based approach. First, it learns to drive on an empty road and next it learns overtaking strategies. The reward function is handcrafted in a manner that the desired behavior is learned in least number of training episodes. Various experiments were conducted to handcraft the reward, a detailed analysis of which have been provided. The learned agent is able to overtake not only on straight but curved roads as well.

Another behavior that was learned was blocking. Blocking is an hostile behavior and should not be practiced in real life. By slightly changing the reward function for Overtaking on highways and by positioning the learning agent ahead of the opponent, blocking behavior was learned. The approach for overtaking and blocking is compared with existing RL based approach for the same.

For dense traffic scenes, we learn two behaviors: Opportunistic and Defensive. Opportunistic agent actively looks for free space ahead of it and navigates itself there, while defensive stays in its own lane and changes its speed each time any vehicle approaches it, to avoid collisions. There is no prior work which deals with end-to-end driving in dense unstructured traffic. Learning the behavior for defensive agent was not easy to achieve by simply using noise. We used intrinsic motivation based approach by explicitly showing it the rewarding actions.

For all of the behaviors the learning is scalable and robust in terms of the speeds of opponent vehicles and the number of cars in the surroundings.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last few decades, the area of autonomous driving has made significant progress owing to the rise of low cost sensors, availability of vast amounts of driving data, and the boom of Learning based methods.

Nevertheless, motion planning and decision making for autonomous vehicles is still a very challenging task. The environmental conditions are highly constrained because of dynamically changing situations and uncertainties in pose estimation of neighboring vehicles.

The behavior of neighboring vehicles cannot be predicted for real life on-road settings, restricting the use of trajectory based optimizations. Moreover, estimating their motion primitives (velocity, acceleration etc) is also unrealistic. Hence we require planning algorithms which plan for every time step and use only the state information of the ego vehicle.

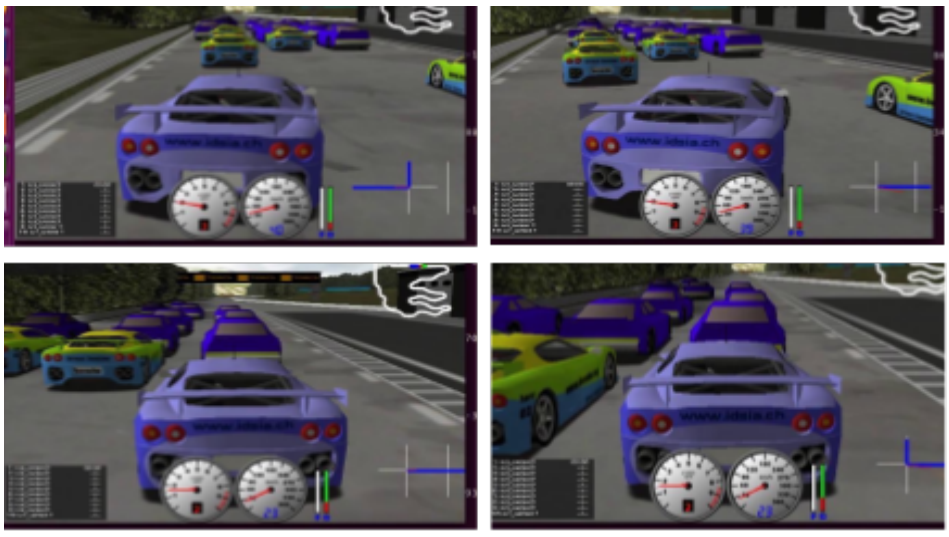

Figure 1.1: A view of the environment and traffic settings for our experiments. The scene consists of cars on three lanes moving with random velocities. The light blue car towards the end is our agent, rest all cars are the traffic components. They can steer in any direction with any speed. Our car is navigating from left most to rightmost lane.

1.1 Related Work

In this section we study the previous works which have approached the problem of planning for autonomous vehicles. One can see, how over the years paradigm have shifted from traditional path planning approaches to Deep learning based approaches. Recently, deep reinforcement learning based approaches for autonomous vehicles have also showed promising results.

1.1.1 Mediated Perception

Mediated Perception approaches segment the scene, derive useful parameters from it and then use those parameters to make the planning decisions.

Solutions started building up gradually, one of the initial works [17] targeted lane and car detection in Highway Driving. They collected data by driving for 14 days, few hours daily, in San Francisco Bay Area. The data was annotated with values of relative speed of all vehicles and the locations of lanes and vehicles. The authors use a modified version of Overfeat [45] CNN detector. Their proposed detector can operate at speeds greater than 10Hz, which is real time, hence they strongly conclude that Deep Learning can be used for developing Autonomous Cars.

Although [17] was a conclusive work, but dataset collection was still an irreplaceable tedious task. Authors in [38] formulate a system where the need of collecting huge training examples could be avoided. The authors aimed at creating a terrain classifier. They used the ImageNet dataset to learn the feature extraction layer. This layer appended to a fully connected neural network, that was trained on navigation sensor input coupled with labeled terrain classes was used as the classifier.

Instance level segmentation was another success story for the development of autonomous vehicles. Every pixel is labeled and the classification is done both class wise and instance wise i.e. for a road consisting of multiple cars and pedestrians, classification would be car1, car2.. pedestrian1, pedestrian2.. pedestrian n. Authors in [60] have developed a method for pixel wise instance level labeling of monocular images. Their results had a significant better performance than others at that time.

Another important research for Mediated Perception in autonomous driving was Deep Tracking, [34]. Authors here use sensor input to create a complete representation of the environment around the object in focus. It is an end to end method with final output being a full, unoccluded scene estimation. [8] is an extension of the previous work, where the authors now track static and dynamic obstacles in the environment. Both of the works use RNNs, which is able to encode the temporal relations between the states and hence enables the interpretation of highly complex dynamic scenes. Clearly, one can see the importance of these perception techniques for autonomous driving.

Tracking and Semantic Segmentation were the starting points for autonomous vehicle research. Authors in [34] presented an end-to-end framework to tracking the environmental components in dynamic surroundings. Authors use Deep Tracking to predict the space occupancy, on top of which they use the principle of inductive transfer for semantic classification. The main contribution here was the end-to-

end mechanism which used only the sensor inputs and was able to track occluded objects as well in dynamic complex environments.

Moving forward from object tracking to path planning, another important contribution in the field was when deep semantic segmentation was able to segment a proposed path. The method [4] generalized well for intersections and roundabouts. The segmented pathway could now be used with any path planner for decision making of the autonomous vehicle.

1.1.2 Behavior Reflex

The first idea following this approach was published in 1989, ALVINN [37]. The medical definition of the term Behavior Reflex is, "a reflex that is gradually developed by training and association through the frequent repetition of a definite stimulus". Reflecting the same, behavior reflex in autonomous driving refers to the end-to-end ML approaches where the input is the scene image and the output is one or all of the driving actuators (steering, brake or acceleration).

ALVINN (Autonomous Land Vehicle In a Neural Network) [37] was a neural network, wherein the vehicle learned how to follow the road. The input to the neural network are the frames of the scene and the range finder values and the output is the direction which the vehicle should take. Surprisingly, the neural network architecture of ALVINN was very simple, it consisted of a single hidden layer with 29 hidden units. For training the images used are snapshots from an artificial simulator. ALVINN was not claimed to be a state of art algorithm for planning in autonomous vehicles, but it was a start for the huge research area of autonomous vehicles with Neural Networks.

MANIAC [19] was developed using ALVINN units. It aimed at addressing the shortcomings of ALVINN, which included generalization to the road types which were not seen using training, navigation between different type of roads. MANIAC consisted of multiple ALVINN networks, each of which trained for one particular type of road. These ALVINN units act as road feature detectors. The final layer took output from these ALVINN layers as its input and outputted the steering commands.

With the success of CNNs [21] in computer vision, using images as input to understand high level scenes had become easier. Authors in [30] have used CNNs to understand the scenes and learned steering angle values for real time robots in a manner that the robot detects and avoids obstacles.

Work done in [30] was limited because of less data and computational power. In 2016, NVIDIA released an approach for End to End Learning for Self-Driving Cars, [6]. Their system, DAVE2, is able to detect features of the road and takes optimized steps to drive on the road. The data collected includes labels for road type, weather conditions and driver's actions. Data contains the frames from the driving process and the steering action chosen by the driver. This data contains only the positive examples, the authors augment the data with negative samples in which image is transformed, the car is shifted from the center of the lane and rotated from the direction of the road. The corresponding steering label is the one that would steer the vehicle back to the desired location. Now the images are fed to CNN with backpropagation error being difference between outputted steering angle and the true value of steering

angle. The evaluation was done first on a simulator and then on real roads. Conclusively, the CNN was able to learn steering commands for lanekeeping.

Previous works have learned the desired skills using dataset containing labels of image scenes and driver actions. A major disadvantage in such methods arises when the neural network is not able to generalize to new scenes and henceforth results in unexpected unsafe behaviors. When a policy function is used to map the state to the action a similar problem arises. DAgger [39], an Imitation Learning algorithm, is one solution to address the problem. It aggregates the dataset by including the trajectories generated by the policies learned in last iteration. Authors in [58] have extended DAgger to SafeDAgger which is used for end-to-end autonomous driving. SafeDAgger modifies DAgger by decreasing the number of queries to the reference policy. This end-to-end approach learns a policy function from sensory inputs to driving commands, by imitating an expert. The algorithm is tested on a well known simulator for autonomous cars, TORCS [55].

1.1.3 Direct Perception

This is the third paradigm in autonomous vehicle research. This approach described in [7] falls in between the two approaches mentioned above. Currently, the work done in [7] is the most successful work under this heading. The main motivation here is to learn features which would estimate the affordance for the task of autonomous driving. It is different from Behavior Reflex because it does not map the image to driving actions directly. When we map an image to driving actions, complexities arise when there are high correlations. Consecutive frames have similar looking images but in cases like lane changing, there are minute changes in the steer angles, which can be so insignificant in values that the algorithm is not able to learn. But the effect of those small changes in steer angles were huge. Behavior Reflex methods can result in unexpected behaviors in such cases.

The approach in [7] uses a CNN based standard architecture AlexNet [21]. It takes the image of the scene as input and outputs various affordance indicators(distances from lane markings, from cars in the same lane, cars in the immediate right and left lanes). The indicators are then used with any planner to determine the complete path profile.

The algorithm is trained with data collected by manually driving a car on each track in TORCS. The screenshots and labels are stored and later the ConvNet is trained using this collected data. Testing is done on various tracks on TORCS as well as on real-world data(KITTI dataset [11] and real driving video from Smartphone camera). Although real world driving is a completely different scenario, the system performed reasonably well.

Another direct perception based method was introduced in [1]. They use GoogLeNet for their CNN architecture and they learn 5 affordance parameters. They claim to outperform the previous existing techniques and the reason for the same being the removal of overlapping redundant affordance parameters.

1.1.4 Trajectory Optimization and Path Planning based approaches

In [3] authors use Model Predictive Control for path velocity decomposition for planning the trajectory of an autonomous car. Their method is based on the Time scaled Collision Cone [13].

[2] uses trajectory optimization for overtaking static obstacles. Authors here use a Receding Horizon Planner on top of which is a state machine which determines whether to overtake or not.

A very detailed analysis of the traditional path planners(RRT, Lattice Planners, Tree Search, Geometric Curve Optimization, Roll Out trajectories etc) and how can they be used for autonomous driving is given by authors in [20].

Authors in [12] also propose a method for autonomous lane changing. They claim that their method can be used as a fully autonomous system. Their method is based on the principles of Rendezvous Guidance.

Authors in [27] use non linear optimization techniques for generating overtaking maneuvers. They use principles calculus of variation and the Pontryagin's Minimum Principle to obtain conditions of optimality and then use the local along with dynamics of lateral motion as constraints.

Another approach that deals with overtaking motion is mentioned in [33]. To prevent any differences in behavior from actual cars the authors here wanted to avoid simplification of the optimization problem. Their planning algorithm optimizes over acceleration and steering change ratio.

There exist techniques which focus on complete on-road driving instead of one particular behavior. The planner developed in [15], used Dynamic Programming based algorithm to explore the space and generate multiple possible trajectories are generated. The most smooth one is then chosen for the path. This approach works well in dynamically changing scenarios as well.

1.1.5 Reinforcement Learning based approaches

On-road driving and developing overtaking skills have been a statement of interest for all domains. Researchers have used Reinforcement Learning to model learn driving for autonomous cars. Authors in [47] use Inverse Reinforcement Learning with Deep Q-Networks to learn driving in simulations. The action and state spaces are discrete here. Another work which uses Deep RL for controlling autonomous vehicles is [56]. Authors use the dataset of experts in TORCS and filter their experience replay during learning. The proposed method decreases the time taken and increases the stability of the trained network.

Our proposed work is inspired from [41]. Authors use DQN and DDAC for end to end learning for lanekeeping. They conclusively show that DDAC performs better than DQN. We build up on their solution of using DDAC for our behavior learning tasks.

Apart from driving control, RL has been used for generating overtaking maneuvers also. [16], [24] and [32] are dedicated approaches for overtaking. In the following chapters we compare our approach for overtaking with theirs.

1.2 Motivation

As we can see driving on-road has been approached by various methods. There are standalone methods for tasks like overtaking and there are end-to-end methods for complete driving task. Reinforcement Learning provides a framework for behavior based learning and their effectiveness algorithms can be verified in [56], [41] and [24]. A major limitation in all of these works is the discrete nature of action and state space. No Reinforcement Learning based motion planner for automated cars existed which used continuous spaces and did more than just driving on an empty lane. Our results show effective end-to-end driving in varied traffic scenes. We evaluated our approach for Highways, dense urban traffics and sparse slow moving traffic. Our approach is scalable and robust in terms of traffic density and speed of the opponent vehicles.

1.3 Our Contribution

- We develop an end-to-end method for learning behavior based autonomous agents. They are guided by their inherent behavior. We have shown results for Overtaking, Defensive, Blocking and Opportunistic Behaviors. Importantly, our agent decides its actions on the basis of its own state. It has no information of the opponent vehicles location or velocities.

- We show how training method can affect the learning process. For all the behaviors learned in traffic first, we train our agent to drive on an empty lane and later, to drive in traffic situations. Such curriculum based learning had significant effects on the results. We compare our Overtaking and Blocking agents with existing RL based overtaking and blocking agents. We could not compare the performance quantitatively due to lack of implementation details.

- We show how guided exploration can speed up the training process and affect the quality of results. For defensive agent, standard OU function as noise was not able to learn the desired behavior. We explicitly added constraints on brake and acceleration for first 50 training episodes, which drastically changed improved the quality of Defensive behavior.

- We also show how environmental settings can affect the behavior of the learned agent. For blocking and overtaking behaviors, the reward structure is almost same, yet the behavior differs hugely. The only change in learning process was the positioning of ego vehicle and the opponent vehicle. When the ego vehicle was placed infront of the opponent, it learned blocking behavior and when placed behind the opponent vehicle it learned overtaking behavior.

1.4 Background

Over the last years Reinforcement Learning has gained a lot of attention because of their impressive results in varied domains. The core of RL lies in learning from exploring the action space in the state

space. Positive reinforcements are given if the action taken at a given state leads to a new state which favors the achievement of long-term goal. RL based techniques have performed human like in domains like Atari Games [28], game of Go[49] and even in robotic manipulators. With the introduction of Deep RL, continuous domain problems are also well addressed now, which was the motivation of the presented work.

In Reinforcement Learning the agent learns from trial and error. A scalar term called reward marks the goodness of each action, given the state space. The agent is continuously interacting with the environment, for every action a_t it takes at time t, it receives a reward r_t and state s_t, the combination of r_t and s_t when fed to policy π decide the next step action, a_{t+1}.

1.4.1 Environment

1. **Fully Observable Environment**
   Agent observes the environment state.

$$O_t = S_t^a = S_t^e \tag{1.1}$$

   Here, O_t refers to the observation at time step t, S_t^a refers to State of agent at time step t and S_t^e refers to state of environment at time step t.

2. **Partially Observable Environment** Agent does not observe the environment state directly. It has its own representation for state.

$$S_t^a \neq S_t^e \tag{1.2}$$

1.4.2 Markov Decision Process

Markov Decision Processes are the environments for RL which are:

1. Completely Observable
2. Current state captures all the information of the history.

- **Markov Property** A state S_t is Markov iff:

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, S_2, ..., S_t] \tag{1.3}$$

- **Markov Process** A sequence of states $S_1, S_2, ..., S_t$ with Markov property.A Markov process (or Markov Chain) is a tuple $< S, P >$
  1. S is a finite set of states
  2. P is a state transition probability matrix,

$$P_{ss'} = P[s_{t+1} = s^{'}|s_t = s] \tag{1.4}$$

1.4.2.1 Markov Reward Process

A Markov Reward Process is a tuple $< S, P, R, \gamma >$:

- S is a finite set of states
- P is a state transition probability matrix

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, S_2, ..., S_t] \tag{1.5}$$

- R is the reward function,

$$R_s = \mathbb{E}[R_{t+1}|S_t = s] \tag{1.6}$$

- γ is the discount factor, $\gamma \epsilon$ [0,1]

1.4.2.2 Return

The return G_t is the total discounted reward from time-step t:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum \gamma^k R_{t+k+1} \tag{1.7}$$

1.4.2.3 State Value Function

The state value function $v(s)$ of an Markov Random Process is the expected return starting from state s:

$$v(s) = \mathbb{E}[G_t|S_t = s] \tag{1.8}$$

1.4.2.4 Action Value Function a.k.a Q-value

The expected return starting from state s, performing an action a, and then following the policy π,

$$Q_\pi = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] \tag{1.9}$$

1.4.3 Q-learning

The inherent aim of any RL agent is to maximize its cumulative reward. $Q(s, a)$ value gives an maps action and state to their corresponding expected reward.

Q-Learning[53, 54] is the most simple algorithm. It stores a mapping of various $< s, a >$ pairs and their corresponding rewards. Whenever an agent reaches a particular $< s, a >$, it simply looks up the table to know the corresponding expected return.

The update equation is given by:

$$Q(s_t, a_t) \leftarrow Q(s_{t-1}, a_{t-1}) + \beta(\Delta_t - Q(s_{t-1}, a_{t-1})) \tag{1.10}$$

here, β is the learning rate, Δ_t equals $r_t + \gamma \max Q(s_t, a)$, r_t is the reward at time t and γ is the discount.

1.4.4 Deep Q-Networks

Q-Learning[53, 54] is one of the most widely used techniques in Reinforcement Learning. It works very well for discrete action and state spaces but gets difficult to train if the action and state spaces increase in size or become continuous in nature. Approximating the Q-function using a Neural Network leads to unstable learning with little convergence.

It should be noted that training in reinforcement learning takes place by sampling states from trajectories and states from one trajectory tend to be similar, training such samples were high correlations lead to the instability. Another breakthrough in Reinforcement Learning algorithms were the Deep Q-Networks[28, 29]. DQNs solve this issue of correlated samples by introducing the experience replay, it is a cache which stores previously encountered states and transitions and the network samples data points from it while learning. Another important proposal of their work was the target network. Since the network from which we are sampling is same as the one getting updated, there arises instability, hence we update the weights once in a while and call this as target network.

1.4.5 Deterministic Policy Gradient

The Policy Gradient Theorem[51] for stochastic policies is very popular in continuous spaces. The deterministic version, Deterministic Policy Gradient Theorem[48], reduces the performance gradient $\nabla_\theta J(\mu_\theta)$ of a deterministic policy μ_θ to a simple expectation, as seen in Eq. 1.11.

$$\begin{aligned} \nabla_\theta J(\mu_\theta) &= \int_S \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)} \, ds \\ &= E_{s\sim\rho^\mu}\Big[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\Big] \end{aligned} \tag{1.11}$$

where $\rho^\mu(s)$ denotes the discounted state distribution by following the policy and $\nabla_a Q^\mu(s,a)$ is the gradient of the Q values $Q^\mu(s,a)$ w.r.t the action a taken in state s, obtained from the policy μ. $E_{s\sim\rho^\mu}[.]$ denotes the expected value with respect to discounted state distribution.

DPGs are can either be On-Policy or Off-Policy, but deterministic policies have inherent incapability of exploring the action space sufficiently. Off-Policy DPG use a stochastic policy to explore, hence they converge faster. The algorithm we use in our approach is a type of Off-Policy DPG.

1.4.6 Deep Deterministic Policy Gradients

Deep Deterministic Policy Gradients[23] is a deep version of DPGs inspired from the success of DQNs. They use the two concepts introduced in DQNs along with a third one called Batch Normalization[18].

1. **Replay Buffer**: Transition Tuples, (s_t, a_t, r_t, s_{t+1}), are sampled from the environment as per the exploration policy and stored into a replay buffer. Here s_t, r_t and a_t denote state, reward and action respectively, at timestep, t. We use this representation in the rest of the paper, with the omission of the subscript t in some cases for brevity. The correlation between the states of

similar trajectories does not allow a stable and convergent learning. Sampling mini-batches of experiences randomly from the buffer solves this issue for both the actor and the critic.

2. **Target Networks**: Instead of directly copying weights, target networks use "soft" updates. A copy of the networks for both the actor and critic are created, denoted by $Q_T(s, a)$ and $\mu_T(s)$ respectively, but their weights are updated by slowly tracking the learned network. This helps in improving the stability of the learning by constraining the weights to change slowly.

$$\begin{aligned} \theta^{Q_T} &\leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q_T} \\ \theta^{\mu_T} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu_T} \end{aligned} \tag{1.12}$$

   where θ^{μ} & θ^{Q} are the network parameters for the actor and critic networks respectively, θ^{μ_T} & θ^{Q_T} are their corresponding target network parameters and $\tau << 1$, is the learning rate.

3. **Batch Normalization**: Different components of a input to a neural network, usually have different units and scales. This results in slower and inefficient training. Batch Normalization was a solution to resolve this. It normalizes each dimension across the samples in a minibatch to have unit mean and variance. It also maintains a running average of the mean and variance to use for normalization during exploration.

DDPG is an off-policy algorithm, hence the exploration technique is completely independent from the learning policy. It allows us to use simple techniques like adding noise into our actor policy for exploration.

Similar to Q-learning[53, 54], weights of Critic Network are learned using a loss obtained from the Bellman-equation:

$$\begin{aligned} L &= \frac{1}{N}\sum_{i}(y_i - Q(s_i, a_i))^2 \\ y_i &= (r_i + \gamma Q_T(s_{i+1}, \mu_T(s_{i+1}))) \end{aligned} \tag{1.13}$$

where r_i is the reward at the i^{th} timestep, $Q_T(s_{i+1}, \mu_T(s_{i+1}))$ is the target Q value for the state-action pair $(s_{i+1}, \mu_T(s_{i+1}))$ where $\mu_T(s_{i+1})$ is obtained from the target actor network, $Q(s_i, a_i)$ is the Q value from the learned network, N is the batch-size and γ is the discount factor.

The Actor network is updated as given below:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_{i} \nabla_a Q(s, a)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s)|_{s=s_i} \tag{1.14}$$

where N is the batch-size, θ^{Q} are the critic network parameters and θ^{μ} are the actor network parameters. The rest of the terms have the same meaning as those in Eq. 1.13. DDPG has been effectively used to solve problems in various domains, examples include [36], [41], [14].

Algorithm 1 Behavior Learning using DDPG

$Randomly\ initialize\ Actor\ and\ Critic\ Networks$
$TargetActor \leftarrow ActorNetwork$
$TargetCriticNetwork \leftarrow CriticNetwork$
for $i = 1\ to\ NumEpisodes$ **do**
  $s \leftarrow ResetTORCS()$
  for $j = 1\ to\ MaxStep$ **do**
    $action \leftarrow Policy(s)$
    $action \leftarrow action + N$
    $s', r, done \leftarrow Step(action)$
    $Buffer \leftarrow Store(s, a, s', r)$
    if $size(Buffer) > BufferSize$ **then**
      $batch \leftarrow Sample(Buffer, BufferSize)$
      $Q_T \leftarrow Update(Critic, batch)$
      $Policy \leftarrow Update(Actor, batch, Q_T)$
      $Update\ Target\ networks\ using\ \tau$
    end if
    if $done$ **then**
      $break$
    end if
  end for
end for

1.4.7 Curriculum Learning

Just like humans, machine learning algorithms learn better when the training samples are provided in a progressively increasing difficulty levels, instead of any random manner. Learning to perform in simpler situations first and eventually building up more difficult ones is faster than learning all of the situations at once. Performance of the system is increased in terms of the speed of convergence and quality of the local minima or maxima. This manner of training in which simpler situations are trained first and complex ones later is called Curriculum Learning [5]. Results of[31] prove the effectiveness of Curriculum learning in Reinforcement Learning techniques as well.

1.4.8 Intrinsic Motivation

Intrinsic motivation in living animals refer to the driving force which comes from inside, to act in a particular manner. It is not the reward of doing the act which is motivating, but the action itself is pleasurable. In [50] the evolutionary aspect of Intrinsically motivated RL is shown. Reward function for adaptive agents are evaluated according to their expected fitness, where explicit fitness function is given along with the distribution for the state of interest.Here the authors search for a primary reward function that maximizes the expected fitness of the RL agent learning through that reward function.

1.5 Chapter Organization

The following chapters in the thesis are organized according the learned behavior of the agent. Chapter 2 shows the simplest behavior, lanekeeping. Chapter 3 talks about the overtaking behavior, how we modified the reward function and introduced Curriculum Learning in the training process. Chapter 4 is dedicated to the blocking agent. Chapter 5 then talks about behaviors in unstructured traffic scenes: Opportunistic and Defensive. Opportunistic agent searches for free spaces in dense traffic and takes them up as soon as possible. Defensive on the other hand tries to save itself from any possible collisions by slowing down. Conclusions are present in Chapter 6.

Chapter 2

Learning to follow lane : Lanekeeping

2.1 Introduction

Lanekeeping refers to the behavior where a car drives straight on the lane, without any collisions or jerky behavior and preferably in the middle of the road. In this case we do not consider any opponent vehicles, because our only aim is to make the agent learn driving on an empty lane. The agent has to avoid collisions with walls and maintain a safe velocity during curves.

We learn two type of lanekeeping behavior:

1. Unrestricted Velocity: The car can increase its velocity till the point the mechanical constraints of the vehicle allow, which usually goes up to 150-200km/hr in our simulator.

2. Restricted Velocity: When the maximum velocity is restricted, the agent navigates more safely, especially on turns. Restricted velocities are more real world like, driving at 200km/hr in traffic scenes(considered in following chapters) is not feasible.

   Lanekeeping behavior on TORCS was one of the tested behavior for DDPG in its original paper [23]. It has been replicated by multiple groups, two of them being, [42] and [22].

2.2 Simulator and Environment Setup

We have used TORCS [55] for all our experiments and development. A modified version called Gym-TORCS [57] is available freely, which facilitated the use of RL algorithms easily with traditional TORCS.

Our agent car is of type scr_server [25], which was developed to be used with TORCS. Unlike the other bots in TORCS, this bot does not have its own intelligence, it rather waits for a client to send it the actions to pursue. In our case the actions are decided by the DDPG algorithm.

We use scr_server1 car on Aalborg track in presence of no other opponents for training of the lanekeeping behavior. We tested on all available TORCS tracks, the agent performed reasonably well on each of them.

We observed the agent learned a very robust behavior on Aalborg track, because the track offered diverse road features, sharp curves, smooth curves and different degrees of slopes over the entire track.

2.3 State Vector and Action Vector

The state vector consists of 29 elements:

1. Angle between the car and the axis of the track.
2. Track Information: Readings from 19 sensors with a 200m range, present at every 10° on the front half of the car. They return the distance to the track edge.
3. Track Position: Distance between the car and the axis of the track, normalized with respect to the track width.
4. SpeedX: As the name suggests, speed of the car along the longitudinal axis of the car.
5. SpeedY: Lateral speed of the car.
6. SpeedZ: Vertical speed of car, indicates bumpiness.
7. Wheel Spin Velocity of each of the 4 wheels(4 readings).
8. Rotations per minute of the car engine

The actions expected by scr_server client include: steer, acceleration, brake and gear. Gym-Torcs has manually controlled the value of gear on the basis of velocity in X direction. Gear control is given by equation 2.1.

$$Gear = \begin{cases} 2, & \text{if } SpeedX \geq 50 \\ 3, & \text{if } SpeedX \geq 80 \\ 4, & \text{if } SpeedX \geq 110 \\ 5, & \text{if } SpeedX \geq 140 \\ 6, & \text{if } SpeedX \geq 170 \\ 1, & \text{otherwise} \end{cases} \tag{2.1}$$

Since gear is manually decided, the Action Vector consists of continuous values, the ranges of which are given below:

1. Steer: This represents the steering angle and ranges from -1 (complete right) to 1 (complete left).
2. Brake: This indicates the strength of braking and ranges from 0 (no brake) to 1 (complete braking).
3. Acceleration: This is like the opposite of brake in the sense that it ranges from 0 (no acceleration) to 1 (full acceleration).

2.4 Neural Network Architecture

The Actor and Critic Networks are shown in figure 2.1 and 2.2. These are same as in [22].

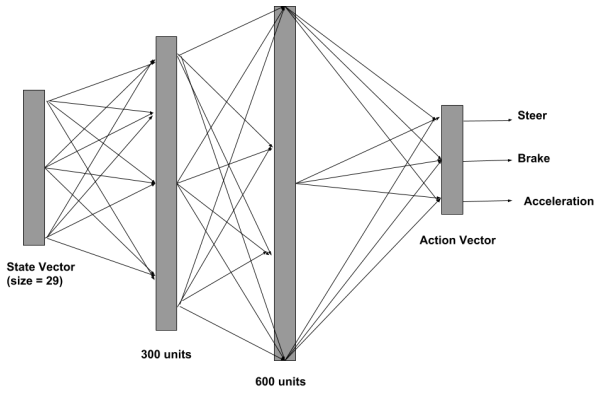

Figure 2.1: Architecture of Actor Network for lane-keeping task

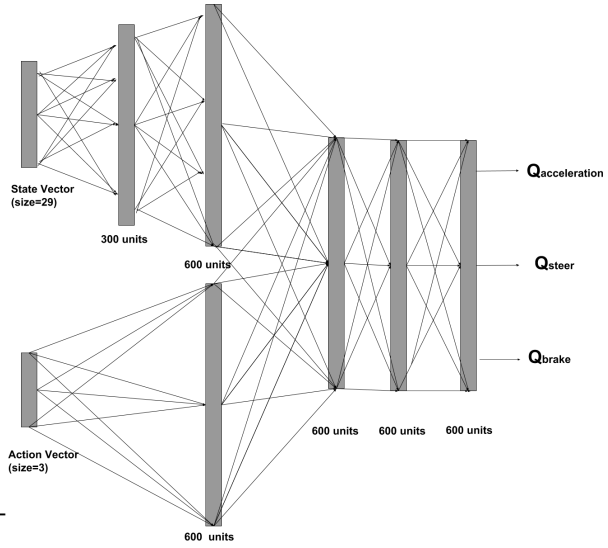

Figure 2.2: Architecture of Critic Network for lane-keeping task

2.5 Exploration

For exploration purpose Ornstein-Uhlenbeck [52] noise is added to all three actions.

In the beginning, for better exploration, noise is kept high. As the agent starts learning the desired behavior, noise is reduced, for which we have kept a multiplier $\epsilon = 1$ and is reduced by 0.00001 every time step. as:

$$\epsilon = \epsilon - 0.00001 \tag{2.2}$$

We train the lane keeping behavior for 2000 episodes with learning rate for the actor network being 0.0001 and for the critic being 0.001, buffer size was 100000, batch size 32 and γ being 0.99. The value of τ for learning the target network weights is 0.001 in both the actor and the critic.

2.6 Reward function

Reward function is the most important property of any RL algorithm. It is the driving motivation for any algorithm to learn a particular behavior. The details of reward functions for the two cases are in the following sections.

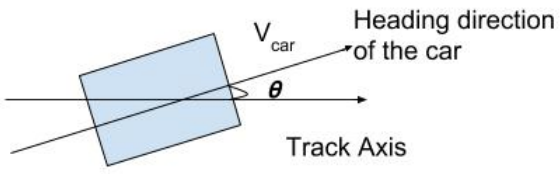

Figure 2.3: Figure illustrating the car's position w.r.t track axis, the cos component of SpeedX is in forward direction and sin component is in lateral direction.

2.6.1 Unrestricted Velocity

Our inherent aim is to make the agent move along the track axis, with maximum possible safe speed and least possible lateral translation. For each time step, the distance traveled along the track axis equals the component of velocity along track axis($v_x \cos\theta$), the lateral drift is the component of velocity perpendicular to the direction of track axis($v_x \sin\theta$). We introduce a term, $progress$ in equation 2.3, indicating the distance traveled along the track direction at each time step.

$$progress = V_x \cos(\theta) - V_x \sin(\theta) \tag{2.3}$$

TORCS provides a sensor value, $trackPos$ which is the distance between the track axis and car's position on track, normalized w.r.t track width. To penalize the agent for deviating from the center of the lane we use the value $trackPos$. Higher the deviation from central axis, lower the reward, hence we negate the value of $trackPos$. Instead of using the exact value of $trackPos$, we use its absolute value so as to punish both right and left shift from the central axis. Not using the absolute value would mean, the deviations towards left side of the lane are favored and towards right are penalized.
The first part of our reward would have values almost equal to velocity, which is in range 10-200km/hr, but $trackPos$ is a normalized value in the range [-1,1]. For $trackPos$ to impact the reward equivalently, we multiply it with $SpeedX$. Hence our final reward function becomes:

$$R_{Lanekeeping} = V_x \cos(\theta) - V_x \sin(\theta) + V_x abs(trackPos) \tag{2.4}$$

2.6.2 Restricted Velocity

We achieved the velocity restrictions using two methods:

1. **Manually limiting the applied acceleration:** The reward function, state vector and action vector are described in section 2.6.1 and 2.3 respectively. On top of the standard conditions we added one extra constraint. Whenever velocity exceeds the maximum allowed velocity, acceleration value is manually set to zero.

2. **Modifying the reward function:** Here as well, the state vector and action vector remained same as in previous case, but reward was modified as in eq. 2.5:

$$Reward = \begin{cases} R_{Lanekeeping}, & \text{if } speedX < maxSpeed \\ -900, & \text{otherwise} \end{cases} \tag{2.5}$$

Here, $R_{Lanekeeping}$ is same as in eq. 3.1, $maxSpeed$ refers to the maximum allowed speed in X direction, this can be set to any reasonable positive value of choice. We have used $speedX$ and not speed, simply out of ease of implementation. $speedX$ is provided as one of the state information for in both cases.
One can use the absolute value of velocity by calculating square root of squared sums of the x and y components of velocity.

We have chosen -900 as the otherwise reward, -900 can be replaced by any large(compared to the values of $R_{Lanekeeping}$) negative value. We took $maxSpeed$ as 30km/hr, hence -900 was a huge negative value compared to $R_{Lanekeeping}$, whose value would be less than or equal to 30 at any given time-step.

Figure 2.4 shows screenshots from the simulator, when lanekeeping with restricted velocity was learned. Our $maxSpeedX$ value was 30km/hr, as consequence of which agent's speedometer shows 24-25km/hr values.

2.7 Extra reward conditions

For both of the above cases, we had extra discrete rewards, to handle special cases.

Condition	Reward
Collision	-100
Off track drifting	-50
No Progress	-10

Table 2.1: Extra Rewarding Conditions

The extra reward conditions make sure that the agent is penalized for wrong actions. We terminate the episode every time any of these conditions is met. When we did not terminate the episode, agent learned that even after collision if it gets positive reward in future episodes, it would go ahead and take those such actions. Although after more number of episodes of training it would eventually learn the expected behavior. Terminating ensures faster learning of the desired behavior.

Also note, not giving the extra reward functions would not affect the quality of the results if the agent is trained for sufficient number of episodes. In absence of these discrete rewards, the agent has another task of learning that when it collides or drifts from track or shows zero progress, it's long term reward

is decreased. Learning these would require more number of training episodes. To sum up we can say, by adding the extra rewards, we are just speeding up the learning process.

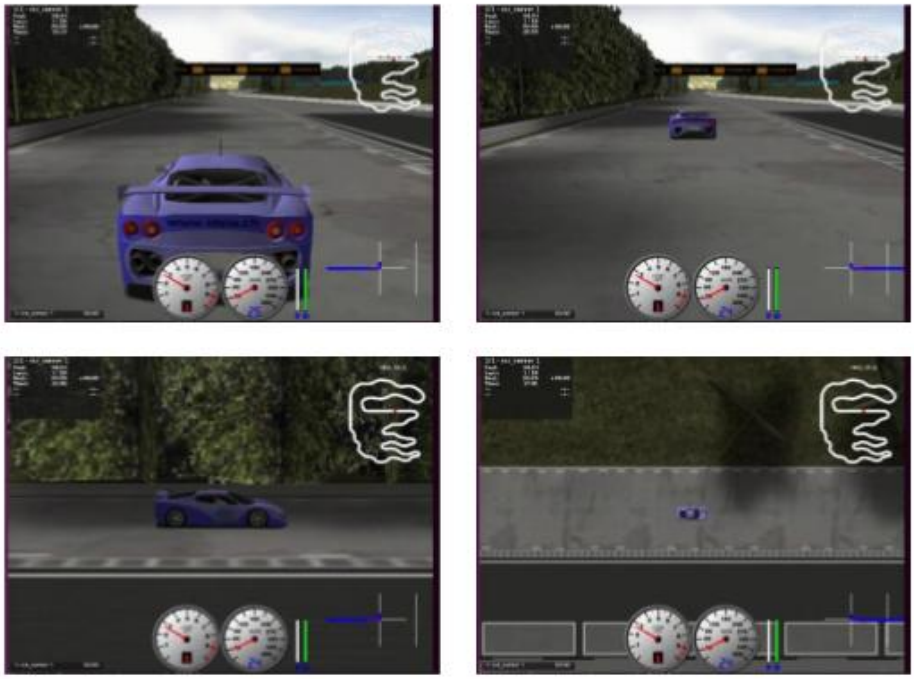

Figure 2.4: An agent trained for lanekeeping with restricted velocity. It's value of velocity is almost constant equal to 24-25km/hr. We provide the view of our agent car from different camera points.

2.8 Results

We observed, 300 episodes onwards, the agent learned to move forward on the track, the motion is not very smooth at this point and sometimes it collides into the wall. 500 episodes onwards the motion is comparatively more stable and the car can easily turn around on simpler curves, but sharp curves are difficult to maneuver. By the end of 1k episode the car is performing very well, learns to drive on lane and navigate smoothly on curves, but it faces some collisions, approximately once in 5 times. We then train it for another 1k episode, after which the motion model is very stable and the agent traverses all type of track maneuvers without any collisions.

In the beginning of the training, the agent has not learned the desired behavior which can be exploited. At this time, exploration is important for the agent, so that it builds up experiences and learns the good and the bad actions.

It is also to be noted that the OU noise added to the actions is highest in the beginning of the training. This is the exploration part of the learning process, eventually when the agent started to learn the desired

behavior, the OU noise is decreased and exploitation of the learned behavior part begins. Maintaining a balance between exploration and exploitation is another important aspect of Reinforcement Learning.

2.9 Conclusion

Lanekeeping in TORCS was shown previously in [23], [22] and [40]. We show two methods by which we restrict the maximum speed of the car. Both of the methods give equivalently good results.

One can see, how the reward function can control the behavior of the learned agent. We gave high negative reward to restrict the highest velocity attained, and now with the restricted velocity the car behaves in a comparatively safer manner.

In the following chapters we will learn more complex behaviors. We initialize our networks with the weights for lanekeeping. We observed that this initialization impacted the results hugely. Without this initialization, we were not able to get the desired behavior from our agent.

Chapter 3

Learning Overtaking Maneuvers for Simulated Highway Environments

3.1 Introduction

Overtaking is a very prominent driving behavior on highways. Learning this behavior is important for any autonomous vehicle. We present a novel approach using Deep Deterministic Policy Gradients (DDPGs) to learn this behavior. Our approach achieves this in a curriculum[5] like setting. We make the agent learn a simpler task first and later we move to learning a complex task.

Our methodology of training the agent is similar to how humans learn to drive. We are first taught to drive a car straight on an empty road, and keep it from straying away from its path. Once this is done, the driver is made to drive in traffic among other cars to learn to navigate in complex scenarios. In a similar fashion, we first train the agent for performing lane keeping. The learned behavior is then augmented to learn overtaking maneuvers. To the best of our knowledge this is one of the first works to approach the problem of overtaking in a continuous manner, and one of the first to show the effectiveness of adopting a curriculum learning approach for tackling such problems. Our learned behavior is scalable and robust in terms of the density and speed of the opponent vehicles.

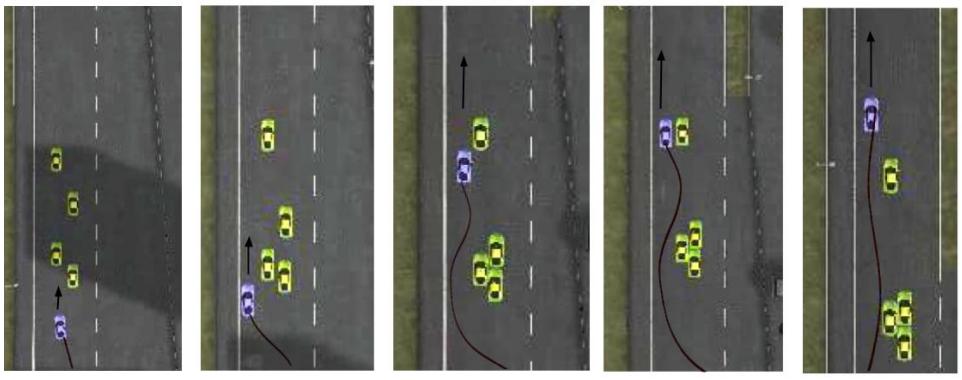

Figure 3.1: Results of our approach. The yellow cars are the opponent cars while the purple car is our agent, its trajectory shown in red. Observe that our agent successfully learns to overtake cars in front of with without any collisions, or going off the road.

3.2 Simulator and Environment Setup

We have used TORCS [55] for all our experiments and development. A modified version called Gym-TORCS [57] is available freely, which enabled us to use RL algorithms at ease with traditional TORCS.
Our agent car is of type scr_server [25], which was developed later to be used with TORCS. Unlike other bots in the simulator, this bot does not have its own intelligence, it rather waits for a client to send it the actions to take. In our case the actions are decided by the DDPG algorithm.

The opponent cars start ahead the agent, their velocities are set to be random values between 20 km/hr to 60 km/hr. During training, there are 4 opponent cars in the scene. Their motion primitives(steer, brake and acceleration) are guided by the controller provided in [10].

3.3 State Vector and Action Vector

The State Vector is a 65 sized array consisting of the following sensor data:

1. Angle between the car and the axis of the track.
2. Track Information: Readings from 19 sensors with a 200m range, present at every 10° on the front half of the car. They return the distance to the track edge.
3. Track Position: Distance between the car and the axis of the track, normalized with respect to the track width.
4. SpeedX: As the name suggests, speed of the car along the longitudinal axis of the car.
5. SpeedY: Lateral speed of the car.
6. SpeedZ: Vertical speed of car, indicates bumpiness.
7. Wheel Spin Velocity of each of the 4 wheels.
8. Rotations per minute of the car engine
9. Opponent information: Array of 36 sensor values, each corresponding to the distance of the nearest opponent in the range of 200 meters, located at a difference of 10°, spanning the complete car.

The first 29 values are same as in section 2.3. It is the opponent information(36 values) which is added for this case.

The Action Vector is exactly same as in section 2.3.

3.4 Neural Network Architecture

We use the DDPG framework to train the actor and critic networks with architectures as shown in Fig. 3.2 and 3.3.

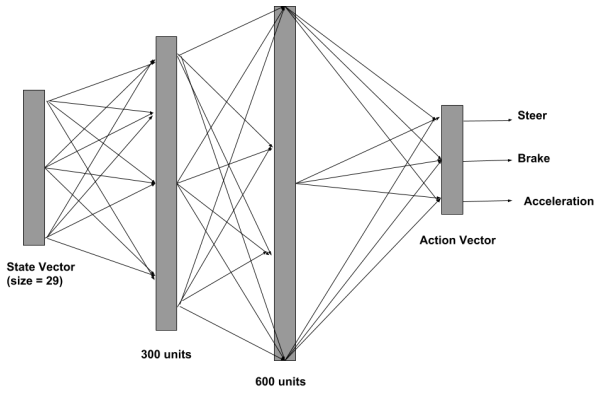

Figure 3.2: Neural architecture of the Actor Network for Overtaking behavior

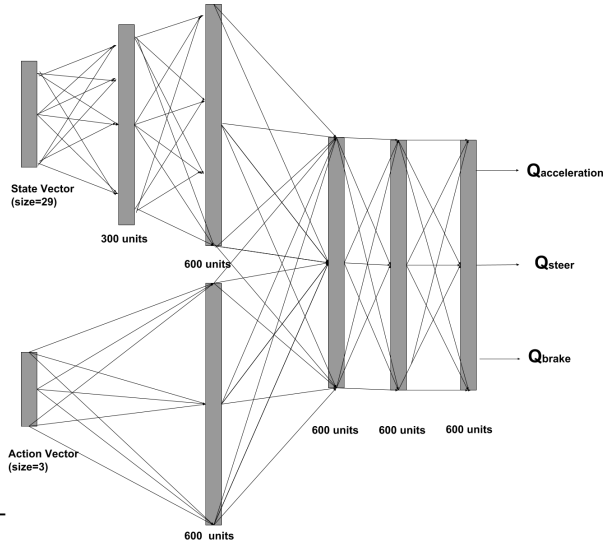

Figure 3.3: Neural architecture of the Critic Network for Overtaking behavior

3.5 Approach

3.5.1 Lane Keeping Behavior

In order to better learn the required behavior, we have two phases of learning. In the first phase, the agent is made to learn to drive smoothly on the road in a single lane. The reward function for achieving this task, is given below,

$$R_{Lanekeeping} = v_x(cos\theta - sin\theta) - v_x abs(trackPos) \tag{3.1}$$

this reward is same as described in section 2.6.1. The difference in the trained network is the size of the state vector. In section 2.6.1, state vector was of 29 length and now it has length as 65.

3.5.2 Overtaking Behavior

Once the agent achieved suitable performance for lane keeping, we added more cars in the simulation and augment the reward function to teach the agent to overtake neighboring cars and navigate in a traffic like scenario. The reward for the second phase of training included the above reward along with a reward for being ahead of other cars so as to favor overtaking behavior. This reward function is given below.

$$R_{overtaking} = R_{Lanekeeping} + 100 * (n - racePos) \tag{3.2}$$

Here n denotes total number of cars in a given episode and $racePos$ denotes the position of car in the race, which is obtained from the simulator. If the car is behind other cars, the value of $racePos$ will be higher, thereby decreasing the value of $n - racePos$ giving a lower reward. Once it overtakes a car, the value of $racePos$ decreases, increasing the value of $n - racePos$, giving a higher reward. For implementing this approach on a real system, vision based approaches can be used to calculate when does the agent overtakes another car.

We multiply the second term by 100, because the first $R_{Lanekeeping}$ is in the range of velocity, i.e 10-200 units. So as to make the second term of equivalent value we multiply it by 100.

Apart from equation 3.2, explicit rewards were given to handle some special cases:

Condition	Reward
Collision	-1000
Off track drifting	-1000
No Progress	-500
Overtaking	$R_{overtaking} + 2000$
Overhauling	$R_{overtaking} - 2000$

Table 3.1: Extra Rewarding Conditions

The explicit conditions of colliding with another car, drifting off the track and not making any progress are necessary, as our reward contains no penalty for them. The extra conditions for Overtaking and Overhauling helped in improving the training rate i.e with these conditions in place the agent learns in less number of episodes.

3.6 Exploration

For exploration purpose Ornstein-Uhlenbeck [52] noise is added to all three actions.

In the beginning, for better exploration, noise is kept high. As the agent starts learning the desired behavior, noise is reduced, for which we have kept a multiplier $\epsilon = 1$ and is reduced by 0.00001 every time step. as:

$$\epsilon = \epsilon - 0.00001 \tag{3.3}$$

We train the lane keeping behavior for 2000 episodes and overtaking behavior for 1000, with the learning rate for the actor being 0.0001 and for the critic being 0.001, buffer size was 100000, batch size 32 and γ being 0.99. The value of τ for learning the target network weights is 0.001 in both the cases.

3.7 Results

3.7.1 Results of our Approach

The important findings of our work are mentioned below:

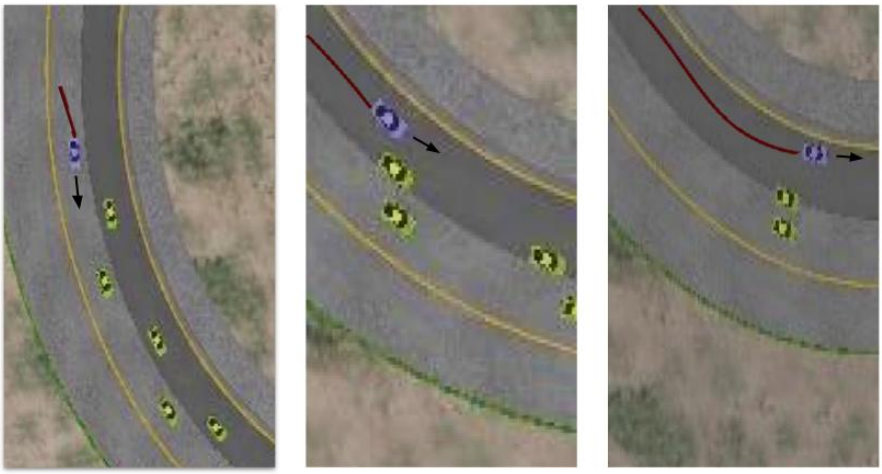

Figure 3.4: Results of our approach on curved track, as viewed from top. The yellow cars are opponent cars and blue is our trained car. Unlike previously existing approaches, [24] our method does not require special assistance for curved roads.

- The highlight of our work is the robustness and scalability we achieve because of using Deep RL. We had trained our agent with only 4 neighboring cars, while testing, our agent was able to successfully drive and overtake on various tracks with as high as 9 neighboring cars.

- Robustness is achieved not only in terms of the number of opponents, but also in terms of the speeds of the opponent cars. During training, speeds of the opponent cars were in the range 10km/hr to 60km/hr. When tested with the opponents with speeds in the range of 10km/hr to 160km/hr, the trained agent was able to overtake seamlessly.

- Our agent was trained on Alpine1 track of TORCS, and we tested it on all available road tracks, it performed significantly well on other tracks. The statistics of performance are provided in table 3.2.

- Conclusively, all of this is possible, because our RL agent has mapped the state vector to action vector, and our state vector does not contain information related to speed of other vehicles or their count. It only contains the distance of obstacles(in all directions) from the agent. Our agent henceforth has learned how to react when anything is present in its vicinity. This is pretty much the same how humans drive, they make decisions at every instant depending on what the environment condition is at that instant.

We analyze our results by two methods:

1. Table 3.2 analyses :

- On all the available Road tracks we evaluate the performance, by calculating the time steps when our agent collides with either the walls or other cars. We calculate the percentage of time steps when any collision takes place. It has to be noted that on average, no collision was a high risk collision, most of the colliding time steps involved the agent car gliding very close to other opponent cars.
- We calculate the number of opponent cars our agent overtakes on an average basis.
- We also calculate the percentage of episodes where our agent overtakes all other opponent cars in the scene.

According to table 3.2, out of 9 cars, our agent was able to overtake 7-8 on average. The values are calculated over 20 episodes, where each episode marks the beginning of all cars starting at 0km/hr from their initial positions. The episode is terminated either when our agent is either out of track or collides disastrously with the walls or other cars, or when it overtakes all other cars. The second column in the table refers to the percentage of timesteps when there was a collision between the agent and another car. In most cases, this value is very low, indicating the collision avoidance nature of our learned agents.

2. TORCS provides multiple cars with inbuilt AI implementations. We race our agent against those cars to evaluate the performance. The parameters we consider for this evaluation as shown in tables 3.3 and 3.4 are:

   - Percentage of time steps when our agent stayed ahead of the AI car.
   - Number of times our agent overtakes the AI car. Since the two cars do not start in line with each other, there are two separate tables, table 3.3 where the agent car starts the first and table 3.4 where the AI car starts the first.
   - Number of times AI car overtakes our agent.
   - Maximum Speed attained by our agent. As mentioned previously, our agent car was trained with opponent cars with speeds in the range 0 to 60km/hr, but for the AI cars, speed is not restricted and while our agent was competing them, it attained very high speeds. In some cases speed was as high as 198km/hr.
   - Average value of damage points of our agent. TORCS provides a information variable for scr_server cars indicating the number of points on car surface, which have experienced any damage.

It is to be noted that we did not train our agent for racing conditions, our agent was trained for real world like highway scenes, where safety was a priority. Hence we did not expect our agent to overtake the AI car always. But the results we got were impressive.

With AI agents: Berniw, BT, Damned, Olethros our car was able to stay ahead almost 50% of the times. If the AI cars overtook our agent, our agent was able to overtake them again, this can be inferred from column 2 and 3 of 3.3.

But when our car started from second place, it was not able to overtake Berniw, Damned, Illiaw and Olethros. For rest all cars our agent overtook them and stayed ahead them for rest of the episodes. The major reason for the inability to overtake was the lead the AI agents took in the beginning.

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

Table 3.2: Analysis on Various Tracks with 4 opponent cars and 9 opponent cars in scene

Name of the AI Car	% timesteps our agent was ahead the AI car	No. of time our agent overtook the AI car	No. of time AI car overtook our agent	Maximum Speed attained by our agent	Average value of Damage
Berniw	46.0508	0.35	1.35	177.6893	1119
BT	44.9948	1.45	1.65	186.3228	1267.5
BerniwHist	100	0	0	177.0957	751.4444
Damned	56.1375	0.25	0.8	188.5394	1071.8
Inferno	100	0	0	198.771	400
InfHist	100	0	0	197.6991	1012
Illiaw	100	0	0	198.771	671.1
Olethros	46.5649	1	1.1	157.7065	2557.9
Tita	94.5806	1	1	171.5821	1165.2

Table 3.3: AI versus our agent, our agent starts from ahead

Name of the AI Car	% timesteps our agent was ahead the AI car	No. of time our agent overtook the AI car	No. of time AI car overtook our agent	Maximum Speed attained by our agent	Average value of Damage
Berniw	0	0	0	171.2543	541
BT	18.4225	0.2	0.2	190.9156	987.6
BerniwHist	93.2503	1	0	184.4135	1050.4
Damned	0	0	0	185.0657	759.8
Inferno	26.4706	1	0	167.6175	3615
InfHist	96.6403	1	0	171.1092	926
Illiaw	0	0	0	174.3621	2077
Olethros	0	0	0	183.5015	813.8
Tita	34.6154	1	0	170.3025	1577

Table 3.4: AI versus our agent, our agent starts from behind

3.7.1.1 Observations and Other Experiments

We gradually build up the solution for our approach. Before we found the approach mentioned in section 3.5, we experimented using the different elements of the final reward function in various combinations. Table 3.5 shows the experiments we conducted and our observations.

Description	Reward	Preloaded lanekeeping rewards	Observations after 3k-4k episodes
Without curriculum learning	$R_{Lanekeeping} + 100(n - racePos)$ and extra reward conditions as per table 3.1	no	Even after 4k episodes of training the agent is not able to learn smooth overtaking trajectories. It changes steer angle to prevent high damage collisions
Wthout $R_{Lanekeeping}$	$100(n - racePos)$ and extra reward conditions as per table 3.1	yes	It doesnt learn to properly overtake, it just saves itself from high damage sometimes
Without +-2000	$R_{Lanekeeping} + 100(n - racePos)$ extra reward conditions 3.1 do not contain overtaking and overhauling conditions	yes	By the end of 4k episodes, the agent is able to learn overtaking, but the number of collisions is extremely high
No extra term, only $R_{Lanekeeping}$ as reward	$R_{Lanekeeping}$ and extra reward conditions as per table 3.1	yes	The agent has began to learn to alter its steer angle to avoid collisions, although it cannot avoid all collisions completely. By the end of 4k episodes, agent has learned to overtake other cars with few collisions, results comparable to Highway overtaking simulation, intuitively results will improve with higher number of episodes

Table 3.5: Observations of various experiments conducted in process of handcrafting the reward for overtaking on highways. For all of the above cases, by the end of 1k episodes, the agent has learned lanekeeping behavior, but it hasn't learned how to react in presence of other cars, it drives straight on the road colliding with whatever comes in between. By the end of 2k episodes it learns how to change steer, it prevents itself from very high damage but nothing more.

- Without the high reward for overtaking(+2000) and the high penalty for overhauling(-2000), the car was able to learn the overtaking maneuvers after number of episodes as high as 4k. These

can be regarded as the differential of $(n - racePos)$ term. When the high positive and negative rewards mentioned above, were included in the reward function, the car was able to learn by 1000 episodes. This indicates, that high positive reward for a behavior at one time step helped in faster learning.

- Without pre-training for lane keeping, the car was not able to learn the overtaking maneuvers, up to 4000 episodes. This can be reasoned on the fact that, in the first 1000-1500 episodes the car learned lane keeping behavior, next when it was expected to learn overtaking behavior, the OU noise was reduced greatly and the car was not able to explore sufficiently enough to learn.

3.7.1.2 Effectiveness of Curriculum Learning

Our approach uses Curriculum Learning styled training i.e we train our network for simple case like lanekeeping. Next we train it for overtaking.

Our agent learned the overtaking behavior in 1k episodes of second phase of training.

To show the effectiveness of Curriculum Learning we compare the results obtained by training 2k episodes without Curriculum styled Learning and 1k with Curriculum styled learning.

Table 3.6 shows the percentage time learned agents overtake all the cars in the scene. We have shown the result for 4 tracks only. It is clear from the bar graph that without curriculum learning the percentage of overtakes is very low. The reason of some percentage of overtakes in the case without curriculum learning is because the agent drives straight onto the road, it overtakes but with collisions. The high number of collisions are apparent from 3.5.

Clearly, curriculum learning was an effective method to increase the quality of results within limited episodes.

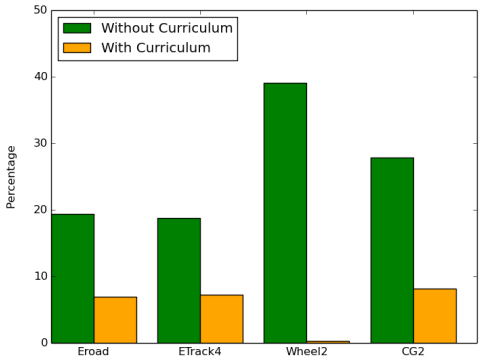

Figure 3.5: Percentage of timesteps in which the agent was involved in a collision (lower the better).

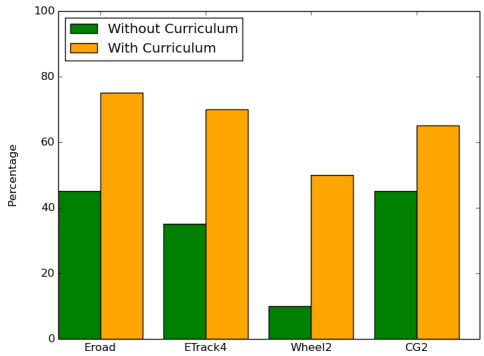

Figure 3.6: Percentage of episodes where agent overtook all cars (higher the better).

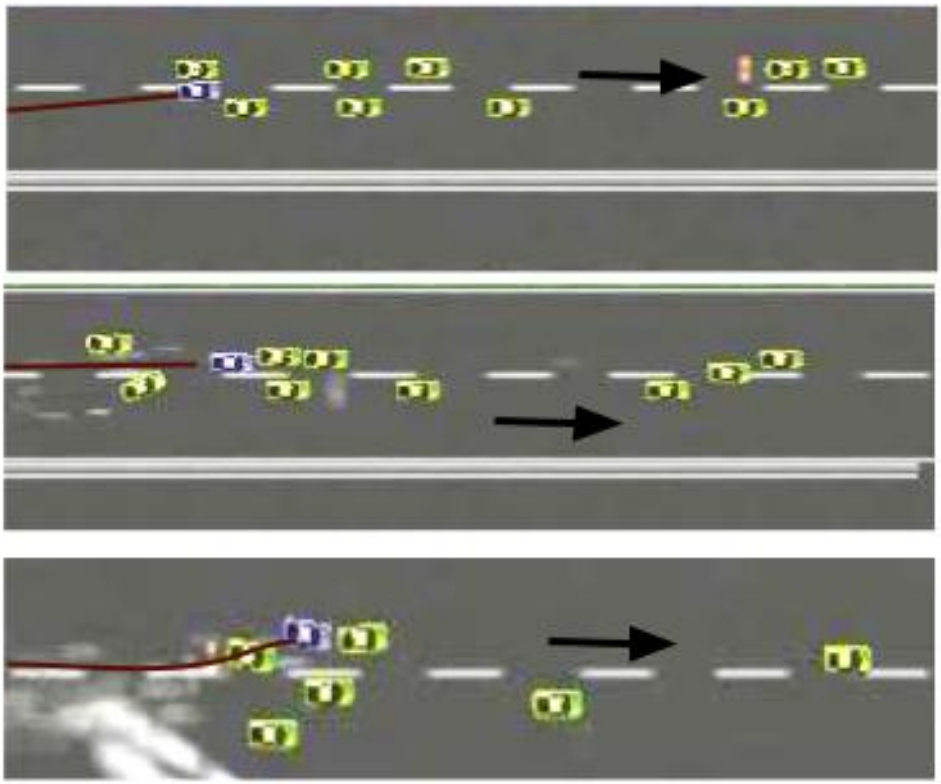

Figure 3.7: Agent's Trajectory when trained without curriculum learning. Neither does the agent properly overtake neighboring cars, nor does it avoid collisions. When the agent is trained without curriculum learning, even after 4k episodes it cannot learn the overtaking behavior. For first 2k episodes it learns the lanekeeping behavior and by the end of 2k episodes the OU noise multiplier decreases hugely, limiting the noise added in the action space. This results in limited exploration, the lack of which is reason of inability of the agent to learn overtaking behavior.

3.7.2 Experiment1: Using previous states of Opponents in the state vector

We modified the state vector from 65 space to 173(29 + 36x4, 29 is the state vector size without opponent information and 36 is the size of opponent information vector) space. Instead of including the opponent information for the current step only, we include the opponent information for current step as well as for previous 3 steps. The state vector in our original approach does not incorporate the opponent information in a temporal manner. To estimate the motion of opponent cars, temporal information is a logical requirement. In an attempt to do so, we have added the previous three opponent information in the state vector.

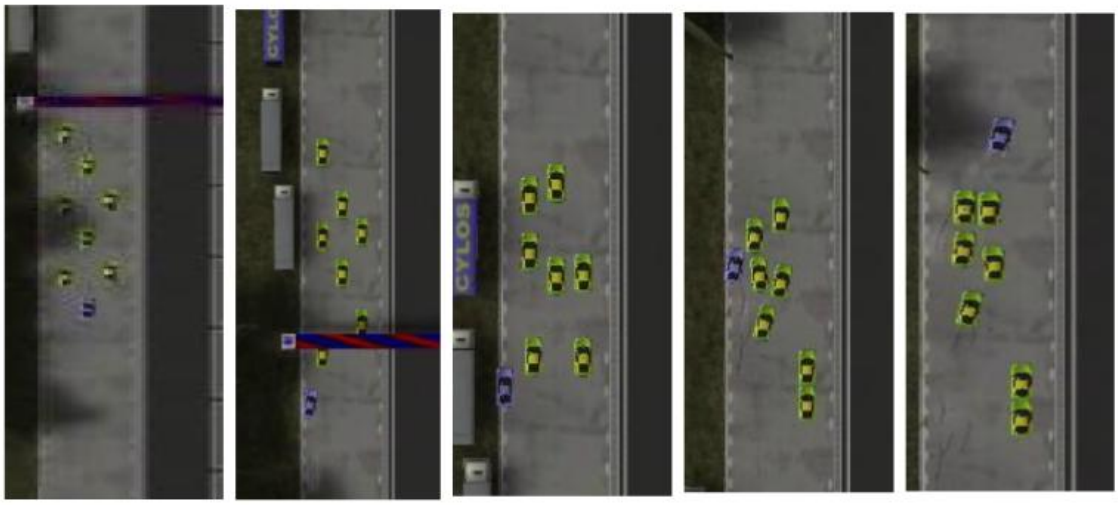

Figure 3.8: Results for highway overtaking using approach mentioned in section 3.7.2. Our agent(blue) starts from the last and overtakes all other cars(yellow), till the last frame.

While training we have kept 4 other cars in front of the agent.
Extra reward conditions are same as in table 3.1.

Table 3.6 shows a comparison between approach in section 3.7.2 and the method used in section 3.5. As it can be inferred from the 3.6, collisions decrease hugely in current method. This can be reasoned on the fact that, last four step opponent information is able to provide velocity estimate of other vehicles. Another inference from 3.6 is the quality of overtaking trajectories. The average number of cars overtaken is lesser in our case. This clearly shows that the agent did not learn sufficient enough. Although it was trained for 1500 episodes which is 500 more than the training episodes of section 3.5. The reason 1500 training episodes is not sufficient is because of the increased state space. The state space increases by more than double, from 65 to 173.

3.7.3 Experiment2: Using Curriculum Learning with $R_{Lanekeeping}$ as reward function

Logically, the reward of lanekeeping indicates moving forward on the track, but if there are cars ahead the agent will collide and its movement along the track will be hindered, hence slowing down the vehicle. To avoid this situation, the agent should overtake the opponent. Based on this understanding, we trained our agent without $R_{overtaking}$ but with Curriculum Learning. After 4k episodes of training the agent was able to learn overtaking maneuvers.

We analyze the trained agent in table 3.7.

Track Name	Avg. no of cars overtook		%of colliding timesteps		% episodes where agent overtook all other cars	
	4cars	9 cars	4cars	9 cars	4 cars	9 cars
Wheel2	3.3	7.7619	0	0	80	85.7143
Forza	3.6	2.75	0	0	95	0
CG2	2.8	6.85	0.2117	0.494	65	65
CG3	0.45	0	0	0	5	0
Etrack1	3.45	8.4	0	0	95	90
Etrack2	3.3	8.15	0	0	90	100
Etrack3	3.1	7.25	0	0	60	85
Etrack4	3.65	8.2	0	0	95	100
Etrack6	3.55	6.8	0	0.1537	90	55
ERoad	3.75	5.4	0	1.3537	95	40
Alpine1	3.4	8.1	1.0067	36.5944	75	95
Alpine2	1.9	5.4	0.9969	1.3537	20	40
Olethros	0.25	2	0	0	5	10
Spring	3.65	8.6	0.2283	0	90	100
Ruudskogen	3.5	8.25	0	0	85	90
Street1	2.7	6.9	0	0	65	75
wheel1	3.8	7.35	0	0	90	75
CG-Speedway1	3.25	1.7	0	0	75	5

Table 3.6: Results of overtaking behavior when learned by using 4 previous opponent information in state vector. These results are comparable to the values in table 3.2. The only difference is this method takes 4k episodes of training while the one in 3.5 took 1k episodes of training. In the absence of the parameter $racePos$, we can use this method for training overtaking behavior.

3.8 Comparison with existing techniques

The presented work is not the first which uses Reinforcement Learning to generate overtaking behaviors. Authors in [16] and [24] have approached the problem using Q-Learning. Authors in [24] develop the complete driver using a structured, layer based architecture, shown in fig. 3.9. As we can see in the figure overtaking strategy is broken down into two tasks : Brake Delay and Trajectory. Trajectory behavior deals with overtaking on a straight path while Brake Delay is for overtaking in tight bends, where the agent would have to delay applying brakes to overtake the other car, which has slowed down because of the curve. The agent uses RL only for overtaking behavior, in all other forms of driving it uses the implementation provided by TORCS under the bot name Berniw. The authors have used Tabular Q-learning, where the action and state space are discrete in nature. The sensors and effectors used are :

Here, one episode is the duration from when the agent car behind the opponent car till the point when either the agent car overtakes the opponent car or collides with it or goes off-track. The reward structure

Track Name	Avg. no of cars overtook		%of colliding timesteps		% of episodes where agent overtook all other cars	
	4cars	9 cars	4cars	9 cars	4 cars	9 cars
Wheel2	3.8	8.1	0.9569	33.1946	100	100
Forza	3.7	8.55	24.8996	15.8672	100	100
CG2	3.75	8.4	30.9051	6.2833	100	95
CG3	3.8	7.4	50.3257	24.6032	95	80
Etrack1	3.95	8.15	0	24.9246	100	100
Etrack2	3.75	8.7	24.6835	47.0948	100	100
Etrack3	3.95	8.35	0.4762	30.4991	100	100
Etrack4	4	8.25	0.1567	29.1467	100	95
Etrack6	3.75	10	23.427	7.6053	100	100
ERoad	3.85	8.55	34.9708	2.5611	100	100
Alpine1	3.55	7.7	18.6765	38.6148	100	100
Alpine2	3.85	8	10.7285	24.4591	100	80
Olethros	3.6	7.85	0.817	22.2113	95	95
Spring	3.8	8.1	0.4934	16.8502	100	100
Ruudskogen	3.3	7.95	7.1325	8.5495	85	100
Street1	3.9	8.15	2.7027	20	100	100
wheel1	4	9	3.0257	14.4873	100	100
CG-Speedway1	3.75	7.35	0	19.3217	100	80

Table 3.7: Results of overtaking behavior when trained with $reward = R_{Lanekeeping}$ for 4k episodes

Name	Range	Representation (discretized)
$dist_y$	[0,200]	{[0,10), [10,20),[20,30), [30,50),[50,100),[100,200)}
$dist_z$	[-25, 25]	{[25, -15), [-15, -5), [-5, -3), [-3,-1), [-1,0), [0,1), [1,3), [3,5), [5,15), [15,25)}
pos	[-10, 10]	{[-10,-5), [-5,-2), [-2,-1), [-1,0), [0,1), [1,2), [2,5), [5,10]}
Δ_{speed}	[-300, 300]	{[-300,0), [0,30), [30,60), [60,90), [90,120), [120,150), [150,200), [200,250), [250,300]}

Table 3.8: Sensors used in [24]. Range is the range of sensor values and Representation indicates the discretization applied to the values to represent them in a lookup table

Action	Move 1mt to left	Keep position	Move 1mt to right
Value	-1	0	1

Table 3.9: Actions used in [24]

is

$$reward = \begin{cases} 1, & \text{if goal was reached} \\ -1, & \text{if car crashed} \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

Authors have considered three cases:

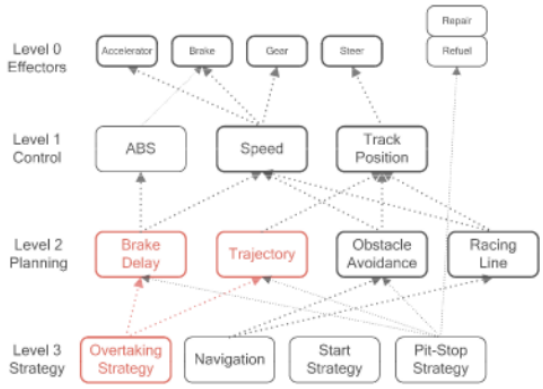

Figure 3.9: Structured behavior generated from the first behavior analysis, as described in [24].

1. Opponent on Fixed Trajectory and Standard Aerodynamics
2. Opponent on Fixed Trajectory and Challenging Aerodynamics
3. Opponent on Random Trajectory and Standard Aerodynamics

For all the three cases number of training episodes were 10000. The quality of the learned behavior is judged by two properties: Overtaking time and Highest Speed attained. Using Wilcoxons rank-sum test, they show the difference in the behavior of learned agent and Berniw is statistically significant with 99% confidence.

As we can see in 3.9, overtaking has two subtasks : trajectory and brake delay. Brake Delay is used in special cases of overtaking in tightly bend curves. This is applied on top of the previous case. Sensors information for this case has an extra variable, $dist_{turn}$ which is the distance between the current position of the car and the next turn along the track axis. $dist_y$ and pos are no more required in sensor information. $dist_{turn}$ was discretized by mapping into {[0,1), [1,2), [2,5), [5,10), [10,20), [20,50), [50,100), [100,250]}. The Action space for Brake Delay System comprises of two possible actions : Inhibit the breaking action(mapped as 1) or leave the decision of braking to other components(mapped as 0). Reward function is same as in previous case, eq. 3.4. Results indicate statistically significant difference as 99%. Although there is a risk of collisions with probability 5.2%.
Unfortunately, due to lack of implementation details for the experiments conducted in [24], we could not compare the results quantitatively. But following is the qualitative analysis.

3.8.1 Main points of difference

- Our approach is robust and scalable. We have trained our agent with four other cars and we have tested it with number of cars ranging from 1 to 15. Our agent successfully overtakes the other cars with few collisions. Our motivation was to develop overtaking trajectories in highway traffic scenes.

- Our approach is end-to-end, we not just give overtaking trajectories, but in absence of other cars we do not use other algorithm to detect the actions. The various traffic scenes do not need to be handled differently, curves, straight paths, no opponent cars all situations are handled in one single approach.
- A very prominent difference is the use of Deep RL with continuous space in our approach and their approach uses tabular Q-Learning with discrete actions and states.

3.9 Conclusions

We present a novel approach for learning overtaking maneuvers in a highway like scenario. Inspired from how humans learn to drive, our approach resembles that of curriculum learning, where we first make the agent learn the simple task of lane keeping followed by adding rewards for learning to overtake. The proposed approach is different from most existing methods in the fact that we gradually build up to the task rather than training it for the final task right away. This type of systematic training not only yields more favorable results but does so in lesser time. The learned agent is able to navigate on various tracks while efficiently overtaking neighboring cars at speeds as high as 160 km/h. We analyze the various combinations of reward function elements and show their effect. This analysis develops an understanding of how RL works and how our agent is reacting to the environment. In the following chapters we directly use this knowledge to generate behaviors of our choice.

Chapter 4

Learning Blocking Behavior for Automated Vehicles

4.1 Introduction

Blocking behavior is a type of aggressive behavior where the driver does not allow the car behind it to overtake. Such behavior makes sense in Championship scenarios but in civilized traffic such behavior is considered hostile.

This behavior may be unnecessary for real world situation, but in simulations it can facilitate generation of varied traffic scenes. Using a blocking car in the front of a dense traffic can further decrease the overall flow of the traffic and make the situation more complex.

Apart from creating diverse traffic scenes, blocking behavior can be used to judge the efficiency of overtaking behaviors of race cars.

4.2 Environment Settings

Similar to the previous cases, here also, we have used TORCS along with scr_server as our agent. To create difficult training conditions, we trained the agent with inbuilt TORCS AI car, Berniw. Berniw is claimed to be one of the most intelligently implemented car of TORCS.

One single car blocking multiple cars is an over expectation, hence for the trainings as well as testings, there is a single opponent car. The training starts with our agent positioned ahead of the AI car.

Similar to previously learned behaviors, we use DDPG to learn blocking behavior. The neural network architecture of Actor and Critic is same as shown in fig. 3.2 and 3.3 respectively. Interestingly, the action and state space are also same as in section 3.3. The state space is a size 65 vector and action is 3.

The most important difference between the training condition for Overtaking and Blocking behavior is the start position of our agent. When our agent starts from behind the opponent car, it learns overtaking behavior and when its starting position is ahead of the opponent car, it learns blocking behavior.
Secondly, we do not train this behavior in a two-fold manner, like overtaking. It is trained in a single fold training, now we do not require the car to learn driving behavior for any type of faster exploration.

The reward function is:

$$R = R_{Lanekeeping} + 100(n - racePos) \tag{4.1}$$

Extra Reward condition are mentioned in table 4.1.

Condition	Reward
racePos == 1	+2000
Collision	-1000
No Progress	-50
Off Track Drifting	-200
Turns around Backward	-100

Table 4.1: Extra Reward Conditions for Blocking Behavior

The reason of the above settings for reward lies in two factors: First, our car should always be moving forward, it should drive on lane and follow all basic behaviors of lane driving. Second, we give our agent huge positive reward(+2000), every time step it is ahead of the opponent car, this huge reward facilitates learning the fact that staying ahead of the opponent car is beneficial behavior. As a result of which our agent learns the blocking behavior.

4.3 Results

We observed that our agent learned how to change its position on the track, so as to come directly in path of the other vehicle and never let it overtake. Our agent was not ruthless and did not collide with the other car.

Figure 4.1: Results of blocking behavior, in first three images, the purple car moves towards right to overtake our(blue) agent, our agent also moves towards the right to prevent the overtaking. In last two frames, the purple car is translating towards left and back to center and our car, also translates in center to block it from overtaking.

Table 4.2 shows that AI cars BT, Damned and Olethros could not be blocked by our Blocking Agent, on the other hand Berniw, Inferno, and Tita could be blocked with 65% chances and Lliaw, InfHist, BerniwHist are easily blocked most of the times.

Name of the AI Car	% of colliding timesteps	%of overhauls
Berniw	1.6473	35
BT	2.1277	100
BerniwHist	0.6547	0
Damned	2.2901	100
Inferno	1.6473	35
InfHist	2.1672	0
Illiaw	0	0
Olethros	0.9202	100
Tita	2.0032	35

Table 4.2: Analysis of Blocking Behavior. % of colliding timesteps indicate the timesteps out of total timesteps where the agent experienced a collision. %of overhauls indicate how many time did the AI car overtook our agent.

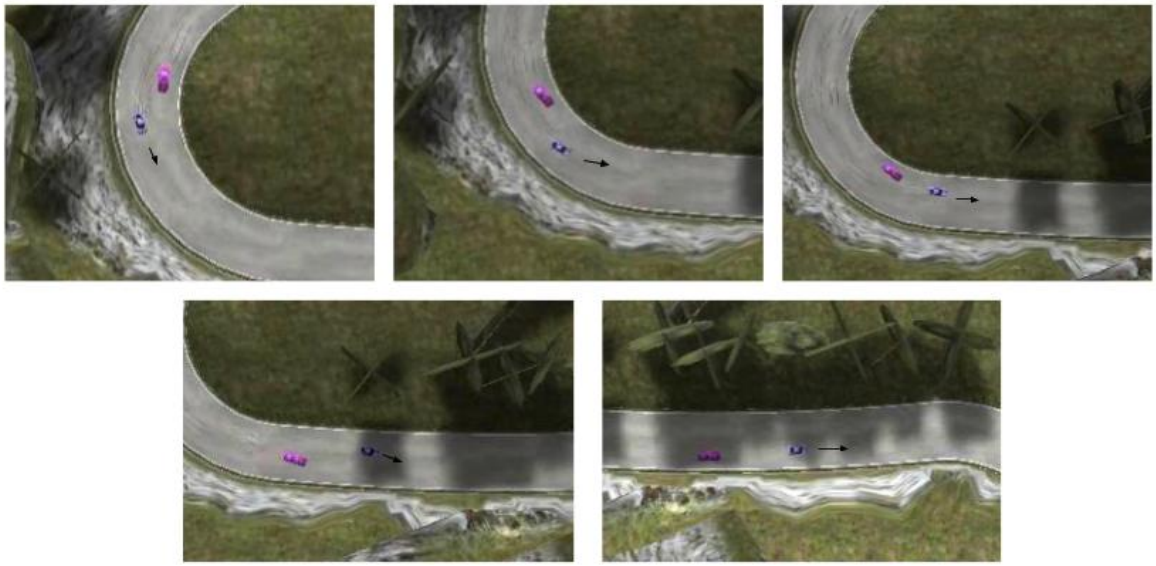

Figure 4.2: Results of blocking behavior on curved roads. The previously existing approach [16], learns blocking on curved tracks by Delayed Braking technique which is applied on top of the main learned method. In our approach we do not have to learn separate behavior for separate type of tracks. Our method offers the advantage of robustness and end-to-end nature.

4.4 Comparison with existing approaches

Blocking behavior has been targeted using Reinforcement Learning in [16]. Their approach is derived from work shown in [24]. The authors have used Berniw as their Base AI car i.e. when the agent

does not need to perform the Blocking characteristics, it will use Berniw's driving implementations. Their approach uses tabular Q-learning with discrete values of action and state space. The state information comes from sensors which has been discretized similar to table 3.8, but with tighter bounds for better modeling. The variables required for this approach are shown in table 4.3. Actions are same as shown in table 3.9.

Variable	Intervals
d_y(m)	[0 5), [5 10), [10 15), [15 20), [20 25), [25 30)
d_x(m)	(- -10), [-10 -5), [-5 -3), [-3 -1), [-1 0), [0 1), [1 3), [3 5), [5 10), [10)
d_v(km/h)	[0 10), [10 20), [20 30), [30 50), [50 70), [70 100), [100 150), [150 200), [200)
p_x(m)	(- -3), [-3 -2), [-2 -1), [-1 0), [0 1), [1 2), [2 3), [3)

Table 4.3: Intervals of Variables in Q-Learning as shown in [16]

Reward function is changed to:

$$reward = \begin{cases} -1, & \text{if AI car overtakes} \\ +1, & \text{if AI car cannot overtake for 60 seconds} \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

The results shown in [16] qualitatively indicate the blocking capabilities of their learned agents. They also show that if though training was done on one agent, the learned agent generalized well on other agents as well.

Authors in [16] show their blocking behavior as a difference in laps covered by two agents, for us most of the time the laps covered were similar, hence such quantitative analysis did not make sense.

Results in [16] compare blocking skills with Lliaw agent on CG-Speedway 1 and Ruudskogen tracks. To compare our approach with theirs we did the same thing. Our agent was able to block Lliaw on both of the tracks for all the time steps. Our agent did not collide with Lliaw even a single time over a period of 50 episodes. For them the damage value ranged from 500-1000. In this comparison our agent performs better than theirs.

The other analysis done by authors in [16] involve use of tracks created by them and QLearning agents trained by them. The implementation details were not clearly mentioned, hence we could not compare quantitatively.

4.4.1 Differences between the two approachs

- Our approach is end-to-end, we not just give overtaking trajectories, but in absence of other cars we do not use other algorithm to detect the actions. The various traffic scenes do not need to be handled differently, curves, straight paths, no opponent cars all situations are handled in one single approach.
- A very prominent difference is the use of Deep RL with continuous space in our approach and their approach uses tabular Q-Learning with discrete actions and states.

4.5 Observations and Conclusions

The most important observation here is how the environment settings can change the behavior learned. The reward function is almost same for both Overtaking and Blocking case, yet the behaviors are different. Our blocking car, once overtook cannot overtake the other car easily. Intuitively, when trained for very high number of episodes, with an opponent who our agent, scr_server can overtake while exploration, then our agent can learn overtaking and blocking behavior simultaneously and can act according to the environment conditions.

Our motivation is not the claim that blocking behavior is best learned by our approach, instead we want to show the powerful nature of Deep RL, which made agents with similar reward learn different behaviors because of environmental settings.

Blocking behavior can be used to help generate complex traffic scenarios, especially situations like Indian road scenes, where vehicles do not drive in a fixed lane and tend to change their behavior randomly.

In the next chapter we would move to trickier traffic scenes. The lanekeeping behavior with restricted velocity, we had learned earlier will come to use in the next chapter.

Chapter 5

Driving in Unstructured Traffic Conditions

5.1 Introduction

Driving in cluttered unstructured environments is not an easy task. By unstructured we imply that the scene is continuously changing and we cannot model the behavior or motion model of the other cars. Different cars are moving at different speeds and with a different motivation. Some are motivated by the need to reach a destination at the earliest possible time and some aim at driving safely without any possible risks. In real time traffic, vehicles are guided by the drivers behavior and very importantly the behavior of nearby cars. The behavior or motion planning decisions of any of the car cannot be decided in advance, any decision taken in past, can be changed at any instant.

Methods which plan in a centralized manner cannot work in real time scenarios, because all the cars are completely independent without any major communication channel. Methods which plan in advance for next few time steps cannot guarantee successful planning because of the dynamic nature of the environment. We need a method which plans for each time step using only the information that is available at the current time step.

We propose a solution to drive in such unstructured environments. We use Deep RL, the input to our algorithm is the sensor readings and velocity details at current time step, of our agent. Actions(steer, acceleration, brake) for each time step are returned. Unlike many popular algorithms for driving our current method does not need the information states for the other cars, our agent learns takes only the current step information vector and learns from experience(training/exploration), how to map the state vector to action vector in a way that reward is maximized. It learns similar to humans, how we approximate distances and take actions at current time step and dynamically decide the actions for next time steps according to the new predicted distances.

Our work targets to learn to navigate in unstructured environments. We have learned different behaviors, with two(Opportunistic and Defensive) of them focused only on how to tackle the congested unstructured dynamically changing environments.

5.2 Environment Settings

During exploration or training, there were 4 opponent cars and our car started from the end. We also explored with more than 4 opponents and with varied traffic scenes, but the best results were obtained with 4 opponent cars. This can be reasoned on the fact that 4 opponents create some good experiences for our agent to learn the expected behavior and introducing more opponents confuses the agent because navigation now gets really difficult. The agent is not able to see good experiences and hence is not able to learn the expected behavior.

While exploitation or testing, we tested on various road conditions, simpler ones included 3 other opponent cars on a two lane structure and the complex ones being three lane structure with our agent surrounded by cars on all 4 sides. The opponent cars had velocities in the range 10km/hr to 40km/hr and the each one was assigned a random velocity in this range. The opponent cars were following the lane for most of the time, to introduce random chaos in the environment, we randomly changed the opponent cars' directions for some random time steps. All of these combined factors resulted in dense unstructured traffic scenes.

5.3 Opportunistic Behavior

Opportunistic behavior refers to the attitude where driver wants to grab every opportunity to move forward in the traffic. The agent would search for free spaces ahead of itself in any of the lanes and whenever possible it will traverse there. In scenes like Indian traffic conditions, many drivers behave in this way. It is a common driving behavior in dense situations.

5.3.1 Reward function

While driving on highways, velocity allowed for the agent is not restricted, which makes the agent ruthless and nasty. Once we restrict the highest attainable velocity, agent is able to learn safe maneuvers in dense traffic conditions. Hence for traffic situations we restrict the velocities as done in section 2.6.2 We train the agent in a manner similar to highway overtaking case i.e using Curriculum Learning based training and preloading the weights of Lanekeeping agent, as mentioned in section 3.5.

$$Reward = \begin{cases} R_{Lanekeeping} + R_{overtaking}, & \text{if } velocity < maxVelocity \\ -900, & \text{otherwise} \end{cases} \tag{5.1}$$

We do not modify the $R_{overtaking}$ because our inherent aim which is to move in a way to occupy any available free space, is equivalent to overtake or to attempt an overtake by lane change. Again, the extra reward conditions are same as in table 3.1. During training, there exist 4 other opponent cars which move with velocities ranging from 5km/hr to $maxVelocity$.

5.3.2 Exploration

To facilitate faster learning we attempt to explore the good actions first, for the same we do not add any noise for first 30 episodes of training, this way the agent tries to drive straight on road and learns how his interactions with other cars affects his rewards. It would collide in many cases but would also occupy free spaces in some cases, seeing such experiences at a early stage itself, would facilitate faster learning. This method of exploration comes under surprise based intrinsic motivation. After the first 30 episodes exploration was done by adding OU noise, as done in all previously learned behaviors.

5.3.3 Results and Observations

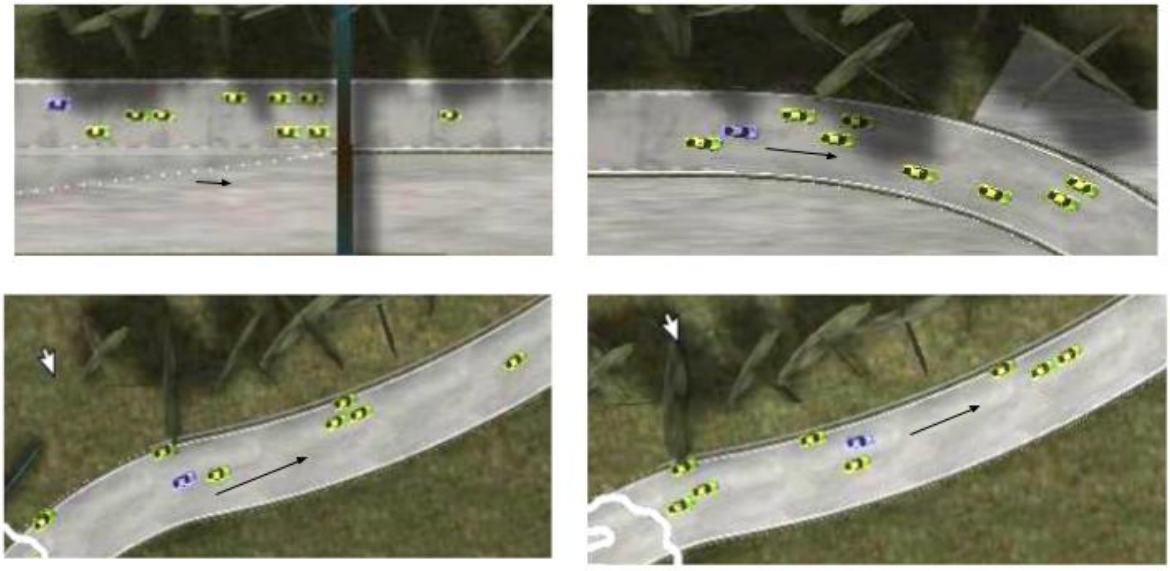

Figure 5.1: Results of opportunistic driving in relatively sparse traffic conditions. One can notice that the blue agent occupied the spaces infront of it. The agent behaves in the desired manner both on curved and straight tracks.

Our results indicated smooth trajectories, where the agent remains under the speed limit and whenever possible, changes its lane to occupy the nearest free space available.
This behavior is representative of how humans behave in very dense traffic situations like traffic jams. Wherever any free space is available, our agent navigates to go there. Such scenes are typical in Indian Roads.
The opportunistic behavior is our first step towards solving decision problems in very dense, unstructured environments. We got the best(collision free and readily occupying free spaces) results when we trained a single agent in presence of 4 other agents.
The number of training episodes after which we got convincing results were 2500.

We experimented increasing the state vector by including the information of previous three steps of opponents. Unfortunately, even after 4500 episodes of training we did not see any significant results. The logical explanation behind the difficulty in learning is the increased size of state space.

Figure 5.2: Opportunistic behavior shown by our agent(blue) in presence of dynamically changing traffic. Our agent detects the free spaces and navigates in between the other cars and takes up the free spaces. This behavior is typical in scenes like on Indian roads.

No. of Agents	Total number of steps in episode	Total no.of colliding steps	% of colliding steps	Structure of the environment
30	310	70	22.5	Highly unstructured, cars surround agent from all four sides.
20	282	44	15.6	Highly unstructured, cars surround agent from all four sides.
15	251	44	17.5	Unstructured, less dense
10	517	22	4.25	Structured, cars follow lanes for majority time
5	300	29	9.6	Less dense
3	200	8	4	Not dense

Table 5.1: Table analyzing Opportunistic behavior in different levels of traffic conditions. Top to down, structured nature of traffic increases.

5.4 Defensive Behavior

Driving in dense traffic does not always mean to try and overtake others, some drivers let others take up the front space and focus on safer driving. Defensive behavior implies such driving where the agent

will focus primarily on avoiding collisions. It will slow down whenever required to avoid collisions from the vehcile ahead.

5.4.1 Action and State Space

The state space is 65 length vector as described in all previous approaches. This is relatively different from the previous approaches. Here, we learn only the brake and acceleration actuators. Steering angle is fed manually and is calculated using SnakeOil [10] agent's steer calculation:

$$steer = (10/PI) \times trackAngle - (0.10) \times trackPos \tag{5.2}$$

here $trackAngle$ is angle between car's heading angle and track axis, $trackPos$ is the relative position of car on the track. It cannot overtake because it cannot manipulate its steering angle. Steering angle values align with the track angle values. We did not preload any weights, this was a faster training because of decreased size of action space.

5.4.2 Reward function

The agent is for motivated moving forward on the lane and for avoiding collisions. We use the $R_{Lanekeeping}$ for moving forward motivation and damage at current time step for avoiding collision. Damage is another sensor information provided by TORCS for scr_server type car.It indicates the number of points on the car, damaged due to collisions. Reward considering the above two factors was

$$reward = R_{Lanekeeping} - damage_{currentTimeStep} \tag{5.3}$$

The extra reward conditions are same as in table 3.1.

5.4.3 Exploration

When this agent was trained with standard OU function as exploration noise, it could not learn the desired behavior. This observation can be reasoned to the fact that applying complete brake and applying zero acceleration would not be generated very frequently by OU noise. Hence, the agent was lacking the experiences where it receives higher reward(in the longer run) by slowing down.
To help the agent see situations where it is rewarded on slowing down, we manually set the acceleration as 0 and brake as 1, whenever the agent collided with any other agent.
This was one most important contribution of intrinsic motivation, in our work, we intrinsically showed it examples of good behavior and eventually it was able to learn from them. After 500-700 episodes of training the agent learned to slow down whenever opponents were detected ahead of it.

5.4.4 Results

The agent follows smooth trajectories, stays in the middle lane and slows down whenever any opponent is approaching in any of the lane, from where it can collide into the agent.

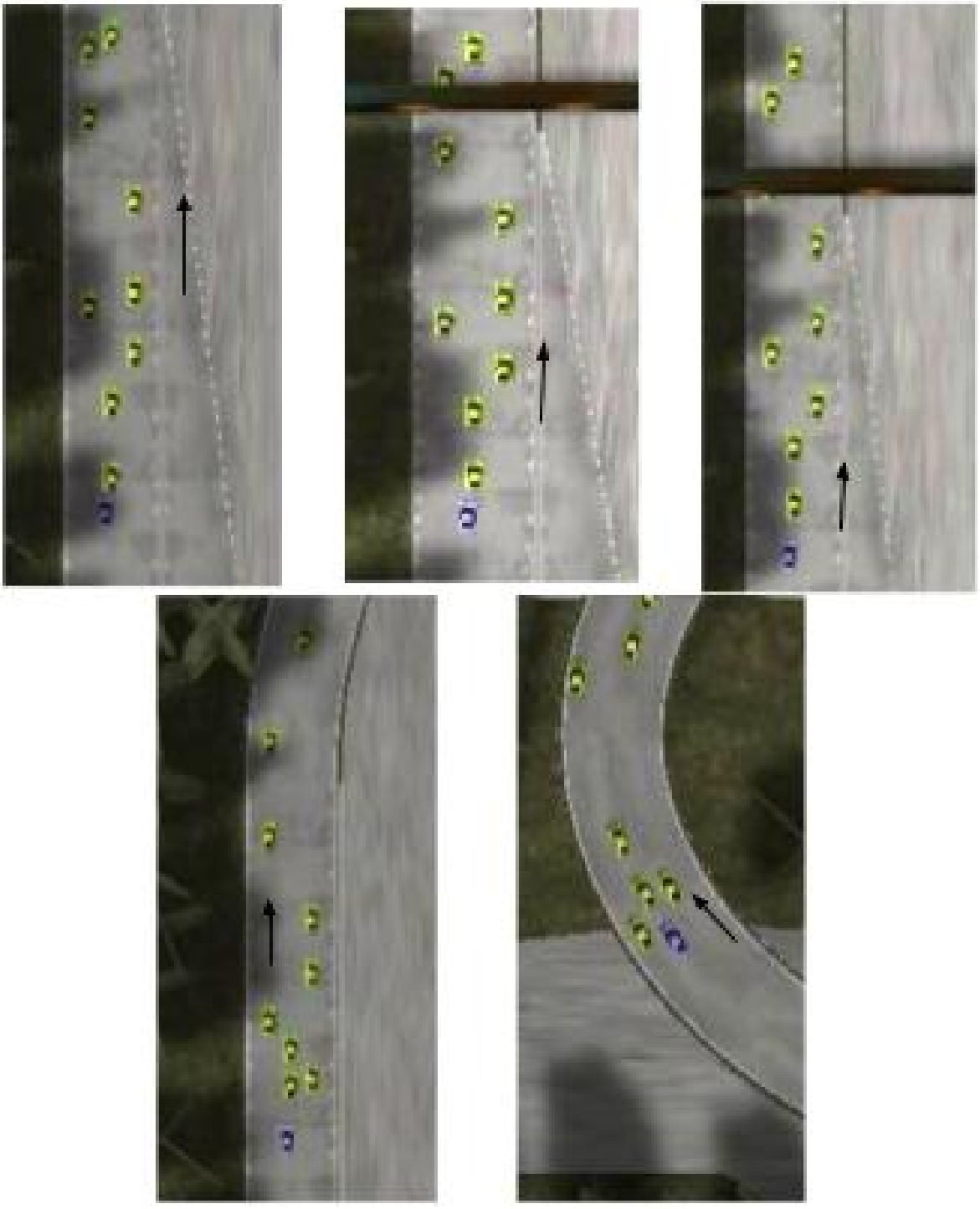

Figure 5.3: This is an example of defensive behavior, whenever our agent(blue) is at a risk of colliding with any other car(green), it slows down. It takes care to not collide with cars in same lane as well as in adjacent lanes. The defensive agent behaves well both on straight and curved tracks.

5.5 Conclusion

Driving in traffic and particularly when the other cars are not following a particular behavior is a challenging task. We attempt to approach the problem using Deep RL, the results we get have a significant performance in varied traffic scenes. The agent is driven by its behavior and hence navigates in complex dynamically changing environments. We propose two behaviors for such complex situations: Defensive and Opportunistic. Opportunistic agent behaves rogue at times as it is driven by the motive to occupy any free space available ahead of it, in real world this can cause tension between the various drivers, hence we propose another behavior. Defensive agent defends itself from any collisions by slowing down. It cannot overtake, it has to stay in its own lane and prevent collisions. We believe defensive agent is safer than opportunistic.

Chapter 6

Conclusions

In this work a relatively unexplored domain, deep reinforcement learning, is explored for autonomous driving. This is one of the first work which targets behavior based driving maneuvers instead of targeting tasks like reaching a goal or overtaking an obstacle. Additionally, this is an end-to-end framework in continuous spaces.

Using an end-to-end DRL based solution for driving in real time is neither safe nor easily achievable. In simulators we have the leverage of colliding into walls and other agents, in real life this is a lot to ask for. One way forward is to learn in simulators and transfer the learning into the real world. Other way forward is to use traditional planners for complete path planning and to use RL for complex decision making. For example, the traditional planner plans the route and suppose there are two equally good paths, the planner outputs both of them and now RL layer will select one of the two path.

In current times, autonomous driving is a popular area of research, unfortunately, there exists no simulator which models behavior based dense traffic conditions. Either the traffic is sparse or it follows a pattern in order to avoid collisions, or the agents have the complete information state of all other agents (which is not realistic). Our work can be extended to generate traffic simulations, for purpose of autonomous driving research. Our approach is easily transferable in other simulators like CARLA [9], AirSim [46], SUMO [35].
A simple extension using the current agents would involve placing different behavioral agents in different relative orientations and get varied traffic scenes.

Another approach to create traffic scenes would involve changing the behavior of the agents depending on their position with respect to other agents. For example, the agent leading the traffic can simply follow lanekeeping behavior, agents in surrounded by many other agents can follow the opportunistic and defensive behaviors and agents which are in sparse density traffics can follow the overtaking behaviors.

Currently, all the agents have learned their behaviors separately. Approaches like Distributed Deterministic Policy Gradients [59] would increase the learned network stability and decrease the training time hugely. For creating varied traffic situations Multi Agent DDPG [26] also looks promising. It aims learning a policy where the individual agents also receive high rewards and the cumulative reward of

the complete system is also attempted to maximized. MADDPG and D3PG are promising directions for behavior based traffic for multi agents.

All of the behaviors learned are standalone in the present state. Learning a meta decision policy to choose which behavior to follow in what conditions can help to create an end-to-end driving framework. Since RL does not use any data or any expert driver, this framework is expected to perform better than the ones based on data driven approaches. Importantly the exploration policy would decide the performance of the training.

Another important contribution can develop out of Inverse Reinforcement Learning. The tricky part in methods involving Inverse RL is the collection of data. One can use the AI agents coded in TORCS to generate the expert trajectories. In IRL based methods we use expert trajectories to learn a reward function, now we use the learned reward function with RL algorithm.

RL has a lot of potential in varied domains, when trained with intelligently chosen reward function and exploration policy, RL can outperform traditional methods.

We explored Deep RL with one method DDPG, recently few other Deep RL algorithms have also shown promising results in varied domains. Proximal Policy Optimization [44], Trust Region Policy Optimization [43] are also continuous control deep RL methods.

Conclusively, this work is an attempt to show the effectiveness of Deep RL in the field of autonomous agents and behavioral learning. Many interesting contributions are possible using Deep RL in the field of autonomous vehicle research as well as in other domains. Optimistically, problems which could not be solved by traditional methods can give significant results with RL with the correct choice of algorithm, reward function and exploration policy.

Related Publications

1. Meha Kaushik, Vignesh Prasad, K Madhava Krishna and Balaraman Ravindran, **"Overtaking Maneuvers in Simulated Highway Driving using Deep Reinforcement Learning"**, in 2018 IEEE Intelligent Vehicles Symposium (IV'18)

2. Meha Kaushik, K Madhava Krishna **"Learning Driving Behaviors for Automated Cars in Unstructured Environments"**, *In review*

Bibliography

[1] M. Al-Qizwini, I. Barjasteh, H. Al-Qassab, and H. Radha. Deep learning algorithm for autonomous driving using googlenet. In *IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017.

[2] H. Andersen, W. Schwarting, F. Naser, Y. H. Eng, M. H. Ang, D. Rus, and J. Alonso-Mora. Trajectory optimization for autonomous overtaking with visibility maximization. In *Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on*, pages 1–8. IEEE, 2017.

[3] M. Babu, Y. Oza, A. K. Singh, K. M. Krishna, and S. Medasani. Model predictive control for autonomous driving based on time scaled collision cone. *arXiv preprint arXiv:1712.04965*, 2017.

[4] D. Barnes, W. Maddern, and I. Posner. Find your own way: Weakly-supervised segmentation of path proposals for urban autonomy. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 203–210. IEEE, 2017.

[5] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, 2009.

[6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[7] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[8] J. Dequaire, D. Rao, P. Ondruska, D. Wang, and I. Posner. Deep tracking on the move: learning to track the world from a moving vehicle using recurrent neural networks. *arXiv preprint arXiv:1609.09365*, 2016.

[9] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[10] C. X. Edwards. 2015 snakeoil competition entry. `http://xed.ch/p/snakeoil/2015/`, 2015.

[11] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[12] U. S. Ghumman, F. Kunwar, B. Benhabib, et al. *Guidance-Based on-line motion planning for autonomous highway overtaking*. University of Toronto, 2008.

[13] B. Gopalakrishnan, A. K. Singh, and K. M. Krishna. Time scaled collision cone based trajectory optimization approach for reactive planning in dynamic environments. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 4169–4176. IEEE, 2014.

[14] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017.

[15] T. Gu and J. M. Dolan. On-road motion planning for autonomous vehicles. In *International Conference on Intelligent Robotics and Applications*, pages 588–597. Springer, 2012.

[16] H.-H. Huang and T. Wang. Learning overtaking and blocking skills in simulated car racing. In *IEEE Conference on Computational Intelligence and Games (CIG)*, 2015.

[17] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015.

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

[19] T. M. Jochem, D. A. Pomerleau, and C. E. Thorpe. Maniac: A next generation neurally based autonomous road follower. In *Proceedings of the International Conference on Intelligent Autonomous Systems*, pages 15–18. IOS Publishers, Amsterdam., Pittsburgh, PA, 1993.

[20] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60:416–442, 2015.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[22] Y.-P. Lau. Using keras and deep deterministic policy gradient to play torcs. `yanpanlau.github.io/2016/10/11/Torcs-Keras.html`, 2016.

[23] 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. In *International Conference on Learning Representations (ICLR)*, 2016.

[24] D. Loiacono, A. Prete, P. L. Lanzi, and L. Cardamone. Learning to overtake in torcs using simple reinforcement learning. In *IEEE Congress on Evolutionary Computation (CEC)*, 2010.

[25] D. Loiacono, L. Cardamone, and P. L. Lanzi. Simulated car racing championship: Competition software manual. *arXiv preprint arXiv:1304.1672*, 2013.

[26] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[27] B. Mashadi and M. Majidi. Global optimal path planning of an autonomous vehicle for overtaking a moving obstacle. *Latin American journal of solids and structures*, 11(14):2555–2572, 2014.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[30] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. L. Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746, 2006.

[31] S. Narvekar. Curriculum learning in reinforcement learning:(doctoral consortium). In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1528–1529. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[32] D. C. Ngai and N. H. Yung. Automated vehicle overtaking based on a multiple-goal reinforcement learning framework. In *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2007.

[33] M. Obayashi, K. Uto, and G. Takano. Appropriate overtaking motion generating method using predictive control with suitable car dynamics. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 4992–4997. IEEE, 2016.

[34] P. Ondruska and I. Posner. Deep tracking: Seeing beyond seeing using recurrent neural networks. *arXiv preprint arXiv:1602.00991*, 2016.

[35] J. L. Pereira and R. J. Rossetti. An integrated architecture for autonomous vehicles simulation. In *Proceedings of the 27th annual ACM symposium on applied computing*, pages 286–292. ACM, 2012.

[36] S. Phaniteja, P. Dewangan, P. Guhan, A. Sarkar, and K. M. Krishna. A deep reinforcement learning approach for dynamically stable inverse kinematics of humanoid robots. *arXiv preprint arXiv:1801.10425*, 2018.

[37] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[38] A. Provodin, L. Torabi, B. Flepp, Y. LeCun, M. Sergio, L. D. Jackel, U. Muller, and J. Zbontar. Fast incremental learning for off-road robot navigation. *arXiv preprint arXiv:1606.08057*, 2016.

[39] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[40] M. Salem, A. M. Mora, J. J. Merelo, and P. García-Sánchez. Driving in torcs using modular fuzzy controllers. In *European Conference on the Applications of Evolutionary Computation*, pages 361–376. Springer, 2017.

[41] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. End-to-end deep reinforcement learning for lane keeping assist. *arXiv preprint arXiv:1612.04340*, 2016.

[42] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017.

[43] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[44] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[45] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[46] S. Shah, D. Dey, C. Lovett, and A. Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017. URL `https://arxiv.org/abs/1705.05065`.

[47] S. Sharifzadeh, I. Chiotellis, R. Triebel, and D. Cremers. Learning to drive using inverse reinforcement learning and deep q-networks. In *NIPS workshop on Deep Learning for Action and Interaction*, 2016.

[48] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning (ICML)*, 2014.

[49] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[50] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, 2010.

[51] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.

[52] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Physical review*, 1930.

[53] C. J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 1992.

[54] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[55] B. Wymann. Torcs, the open racing car simulator. URL `http://www.torcs.org`.

[56] W. Xia, H. Li, and B. Li. A control strategy of autonomous vehicles based on deep reinforcement learning. In *International Symposium on Computational Intelligence and Design (ISCID)*, 2016.

[57] N. Yoshida. Gym-torcs. `github.com/ugo-nama-kun/gym_torcs`, 2016.

[58] J. Zhang and K. Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016.

[59] S. Zhang and O. R. Zaiane. Comparing deep reinforcement learning and evolutionary methods in continuous control. *arXiv preprint arXiv:1712.00006*, 2017.

[60] Z. Zhang, S. Fidler, and R. Urtasun. Instance-level segmentation for autonomous driving with deep densely connected mrfs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 669–677, 2016.