Learning Driving Behaviors for Automated Cars in Unstructured Environments

Meha Kaushik, K. Madhava Krishna

Robotics Research Center, International Institute of Information Technology, Hyderabad

Abstract. The core of Reinforcement learning lies in learning from experiences. The performance of the agent is hugely impacted by the training conditions, reward functions and exploration policies. Deep Deterministic Policy Gradient(DDPG) is a well known approach to solve continuous control problems in RL. We use DDPG with intelligent choice of reward function and exploration policy to learn various driving behaviors(Lanekeeping, Overtaking, Blocking, Defensive, Opportunistic) for a simulated car in unstructured environments. In cluttered scenes, where the opponent agents are not following any driving pattern, it is difficult to anticipate their behavior and henceforth decide our agent's actions. DDPG enables us to propose a solution which requires only the sensor information at current time step to predict the action to be taken. Our main contribution is generating a behavior based motion model for simulated cars, which plans for every instant.

Keywords: Reinforcement Learning, DDPG, overtaking, blocking, driving in traffic, unstructured environments

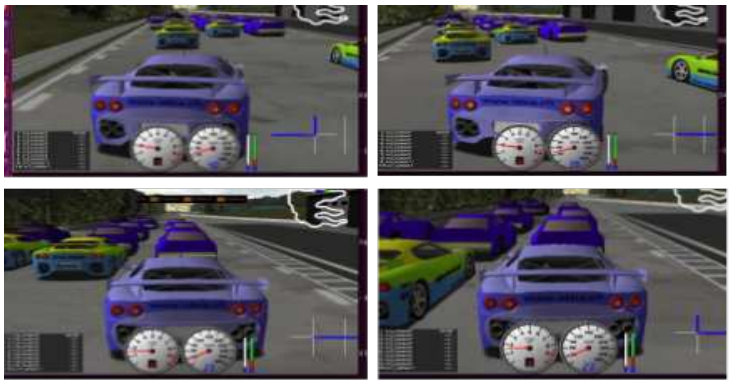

Fig. 1: A view of the environment and traffic settings for our experiments. The scene consists of cars on three lanes moving with random velocities. The light blue car towards the end is our agent, rest all cars are the traffic components. They can steer in any direction with any speed. Our car is navigating from left most to rightmost lane.

1 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 driver's 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. The scenes we have used to evaluate our results consist of three congested lanes, where the cars are driving at random velocities. They can change their lanes anytime and create chaos in the environment. We have learned different behaviors, with two(Opportunistic and Defensive) of them focused only on how to tackle the congested unstructured dynamically changing environments.

2 Related Work

The problem of autonomous driving control has been targeted by perception based methods. Two broad classifications for them are Mediated Perception(The complete scene is segmented and components are recognized and the estimations are used for calculating the control commands of the vehicle) and Behavior Reflex(Information from sensors, range finders, GPS, LiDAR, Radars etc are directly used to calculate control commands). [1] and [2] are based on Mediated Perception approach while [3] and [4] are Behavior Reflex techniques. A third technique called Direct Perception was introduced by DeepDriving [5]. It falls in between the other two paradigms. It learns several meaningful indicators of the road situations which can be used with any controller to make driving decisions.

We have used TORCS [6] an Open Source Simulator for research on autonomous cars. Controllers for driving in TORCS have been developed using various techniques: [7] uses Modular Fuzzy Controllers, [8] uses evolutionary strategy for the controller. Methods using Artificial Neural Networks have also been in developed [9]. End to End driving in TORCS has also been achieved using Imitation Learning [10]. Motivated by these and the recent success of RL algorithms, developing RL based controllers seems a reasonable step.

Reinforcement Learning and driving have been targeted together previously as well. In [11], authors have learned lanekeeping in TORCS using DQN(for discrete action space)[12] and DDPG(for continuous action space)[13]. [14] also learns to drive on lane using Deep Q-Network. Another interesting work is [15], they have used Deep Q-Networks but they have also learned the reward function using Inverse Reinforcement Learning.

Automated Vehicle Overtaking is a standard problem in autonomous learning, it has also been targeted using Reinforcement Learning using multiple approaches. Authors in [16] have used RL along with destination seeking approach and collision avoidance constraints. Collision avoidance is taken care by Double-action Q-Learning while Q-Learning is responsible for destination seeking. Blocking and Overtaking, both were taken up by authors in [17]. They have used simple Q-Learning for the same. [18] also uses Q-Learning to learn overtaking behaviors.

Most of the previous work is based out of Deep Q-Networks and Q-Learning. A major drawback of these algorithms is the discrete action space. Fortunately, continuous control using Deep RL is also solved using DDPG[13]. Deep Deterministic Policy Gradient (DDPG) has given impressive results in various domains: Manipulators [19], Humanoids [20], Automated Vehicle Driving [21,22,11].

We use DDPG to create various driving behaviors (namely, Lanekeeping, Overtaking, Opportunistic, Defensive and Blocking).

3 Background

3.1 Deep Deterministic Policy Gradients

DDPG is a deep RL algorithm that aims at solving problems where the action and state space are continuous. It implements Deterministic Policy Gradients using Neural Networks. The main components of the algorithm are:

1. **Replay buffer:** The training samples are samples of experiences from a sequence of time steps. The consecutive steps of the sequence are highly correlated. If the correlated experiences are fed sequentially then the training may result in unstable learned weights. To avoid this, 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.
2. **Batch Normalization**: Different components of the state vector inputted 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.

3. **Actor Critic Networks:** Actor Critic Algorithms [23,24,25] are a class of RL Algorithms that exploit the strengths of actor-only and critic-only algorithms. The Actor determines the action to be taken according to a policy. say $\pi(\theta)$. The Critic learns the parameters of the actor policy i.e θ. The Critic network uses a Bellman update to learn a value function based on this policy and using that value function as shown in 1.

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

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 updates its policy parameters in the direction of the ascending gradient of the value function. Its update is as given below:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_{i} \nabla_a Q(s, a)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s)|_{s=s_i} \tag{2}$$

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.

4. **Target Networks:** The stability of the weights learned is improved by using Target Actor and Critic Networks. They are not updated directly by copying weights but by using soft update:

$$\begin{aligned} \theta^{Q_T} &\leftarrow \tau\theta^Q + (1-\tau)\theta^{Q_T} \\ \theta^{\mu_T} &\leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu_T} \end{aligned} \tag{3}$$

Here actor and critic are denoted by $Q_T(s, a)$ and $\mu_T(s)$ respectively. θ^{μ_T} & θ^{Q_T} are their corresponding target network parameters and $\tau << 1$, is the learning rate.

5. **Exploration:** DDPG is an off-policy algorithm, hence exploration need not come from the learned policy. We add OU Noise [26] in the actions produced by Actor Network, as proposed in original paper [13].

1 shows the complete DDPG algorithm for behavior learning.

3.2 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 [27]. Results of[28] prove the effectiveness of Curriculum learning in Reinforcement Learning techniques as well.

3.3 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 [29] 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.
Recent works [30] and [31] have used Intrinsic Motivation to increase the performance of RL algorithms. Intrinsic motivation can be of three types: Empowerment(agent enjoys the level of control it has over future), Surprise(agent is exploring i.e. it gets excited to the outcomes that run contrary to it's understanding of the world) and Novelty(excited to see the new states). In [30] authors formulate Surprise based Intrinsic Motivation for Deep RL. [31] formulates a method to increase exploration using a pseudo-count from arbitrary density model. These pseudo counts is used for improved exploration. Our motivation for using Intrinsic motivation comes from the success of [30] and [31]. Following a similar approach using surprised based intrinsic motivation, we show our agent approximately good trajectories so that it learns the expected behaviour faster.

4 Simulator details

We have used TORCS [6] for all our experiments and development. A modified version called Gym-TORCS [32] is available freely, which enabled us to use RL algorithms at ease with traditional TORCS.
Our agent car is of type scr_server, 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 are also of type scr_server [33] and their actions are decided as in the SnakeOil Agent [34].

5 Driving behaviors

Our work is based on [21]. Authors in [21] have shown how to use DDPG and curriculum learning to learn overtaking in simulated highway scenarios. The

authors handcraft a reward function to learn the overtaking maneuvers. Given below are the details of the work.

1. Lanekeeping behaviour i.e. to drive on lane smoothly without collisions or abrupt velocity and acceleration changes was trained using

$$R_{Lanekeeping} = v_x(cos\theta - sin\theta) - v_x abs(t) \tag{4}$$

   where v_x denotes the longitudinal velocity of the car, θ denotes the angle between the car and the track axis. t is the fraction by which the car has moved away from the track axis, it lies in between [-1,1].
2. The weights of neural network learned in step 1 were loaded for second phase of training. The environment now consists of (n-1) other cars. Reward for this step is

$$R_{overtaking} = R_{Lanekeeping} + 100 * (n - racePos) \tag{5}$$

   where n is the total number of cars and $racePos$ indicates how many cars are ahead of the agent.
3. To handle collision and off-track drifting, negative rewards were given as per table 1

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

Table 1: Extra Rewarding Conditions

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.

Further details about each of these sensor readings can be found in [33].

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 to 1, where -1 indicates steer completely to right and +1 indicates to steer completely to left.
2. **Brake:** This indicates the strength of braking and ranges from 0 to 1, where 0 indicates no brake and 1 indicates brake with complete strength.
3. **Acceleration:** This is like the opposite of brake in the sense that it ranges from 0 to 1, 0 indicates no acceleration and 1 means complete strength.

Algorithm 1 Behavior Learning using DDPG

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

The following section shows the different driving behaviors we attempted to learn for an agent.

5.1 Overtaking on Highways

Our approach for overtaking on highways is derived from the method used by authors in [21]. We have 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 [21] 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.
While training we have kept 4 other cars in front of the agent.
Extra reward conditions are same as in table 1.

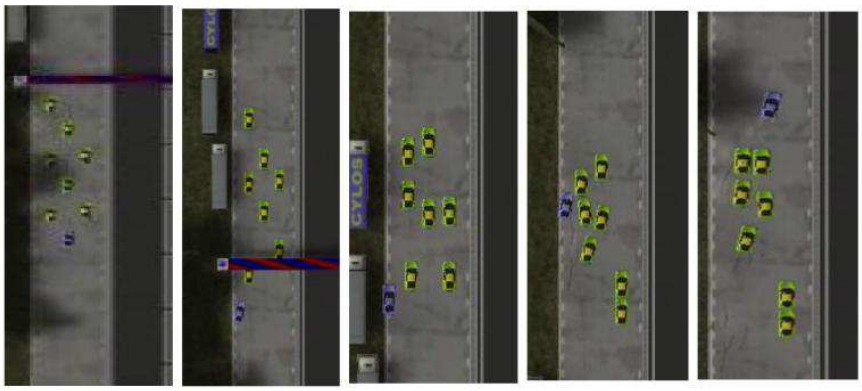

Fig. 2: Results for highway overtaking using our proposed method. Our agent(blue) starts from the last and overtakes all other cars(yellow), till the last frame.

Results Our results indicate smooth overtaking trajectories, with collisions hugely decreased than [21]. Table 2 shows a comparison between our method and the method used in [21]. As it can be inferred from the 2, collisions decrease hugely in our 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 2 is the quality of overtaking trajectories. The average number of cars overtaken is lesser in our case. This clearly shows that the agent was not trained enough. Although it was trained for 1500 episodes which is 500 more than the training episodes of [21]. 1500 training episodes is not sufficient because of the increased state space. The state space increases by more than double, from 65 to 173.

5.2 Lane-keeping with restricted maximum speed

Approach This behavior is derived out of the work done in [22]. In [22] the agent assumes no velocity constraints, hence acquires velocities in the range 120-170km/hr after stable learning. In real world scenarios, such high velocities do not classify as safe behaviors, hence we train an agent with a constraint of maximum possible velocity. We achieved the velocity restrictions using two methods:

1. **Manually limiting the acceleration applied:** We kept the reward function, state vector and action vector same as in [22] and added one extra constraint, for all time steps, i.e. if velocity exceeds the maximum allowed velocity, acceleration is manually set to zero.
2. **Modifying the reward function:** Here as well, the state vector and action vector remained same, but reward was modified as in eq. 6:

$$Reward = \begin{cases} R_{Lanekeeping}, & \text{if } velocity < maxVelocity \\ -900, & \text{otherwise} \end{cases} \tag{6}$$

Track Name	Avg No of cars overtaken				% of colliding timesteps				% of episodes where agent overtook all other cars			
	Case A		Case B		Case A		Case B		Case A		Case B	
	4cars	9cars	4cars	9 cars	4 cars	9cars	4cars	9 cars	4 cars	9 cars	4 cars	9 cars
Wheel2	3.95	7.55	3.3	7.7619	0.2291	0.2328	0	0	100	50	80	85.7143
Forza	4	7.8	3.6	2.75	25.8348	9.64	0	0	100	40	95	0
CG2	4	8.45	2.8	6.85	7.0111	8.135	0.2117	0.494	95	65	65	65
CG3	3.05	6.15	0.45	0	25.6376	39.7	0	0	30	35	5	0
Etrack1	4	8.35	3.45	8.4	7.0789	1.05	0	0	100	80	95	90
Etrack2	3.55	7.8	3.3	8.15	26.4079	0	0	0	65	60	90	100
Etrack3	4	6.35	3.1	7.25	7.987	2.36	0	0	100	40	60	85
Etrack4	4	8.5	3.65	8.2	0	7.23	0	0	100	70	95	100
Etrack6	3.65	7.8	3.55	6.8	26.1345	10.5	0	0.1537	90	60	90	55
ERoad	4	8.05	3.75	5.4	3.2502	6.9	0	1.3537	100	75	95	40
Alpine1	4	8.55	3.4	8.1	17.4464	0.67	1.0067	36.5944	100	80	75	95
Alpine2	3.9	7.95	1.9	5.4	7.5701	0.7064	0.9969	1.3537	85	50	20	40
Olethros	4	7.1	0.25	2	7.2993	18.84	0	0	100	30	5	10
Spring	3.8	7.8	3.65	8.6	3.8753	8.05	0.2283	0	95	45	90	100
Ruudskogen	3.95	7.65	3.5	8.25	2.2113	12.2895	0	0	100	40	85	90
Street1	3.95	8.55	2.7	6.9	4.0665	6.1907	0	0	100	80	65	75
wheel1	4	8.5	3.8	7.35	0	10.3769	0	0	100	50	90	75
CG-Speedway1	3.85	7.95	3.25	1.7	5.621	0.2328	0	0	95	50	75	5

Table 2: Comparison of Results in [21] and our approach. Case A refers to [21]'s approach and Case B refers to our approach

Here, $R_{Lanekeeping}$ is same as in eq. 4, *velocity* refers to the velocity of the agent at current time step and $maxVelocity$ refers to the maximum allowed velocity, this can be set to any reasonable positive value of choice. 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 $maxVelocity$ as 30km/hr, hence -900 was a huge negative value compared to $R_{Lanekeeping}$, which would be less than or equal to 30.

In both of the cases the extra reward conditions are same as in table 1.

Results and Observations Learning was very stable, the car made smooth turns because of the controlled velocity. Both of the methods work equally good. We also trained the same conditions with state space consisting of opponent information, the results were not affected by the presence of this redundant information.

5.3 Driving in traffic/Opportunistic Behavior

Approach In [21], 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. We train the agent in a manner similar to [21] i.e using Curriculum Learning based training and preloading the weights of Lanekeeping agent(here, with restricted velocity)

$$Reward = \begin{cases} R_{Lanekeeping} + R_{overtaking}, & \text{if } velocity < maxVelocity \\ -900, & \text{otherwise} \end{cases} \tag{7}$$

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.
During training, there exist 4 other opponent cars which move with velocities ranging from 5km/hr to $maxVelocity$.
To facilitate faster learning we 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. This method of exploration can be considered an example of surprise based intrinsic motivation. Again, the extra reward conditions are same as in table 1.

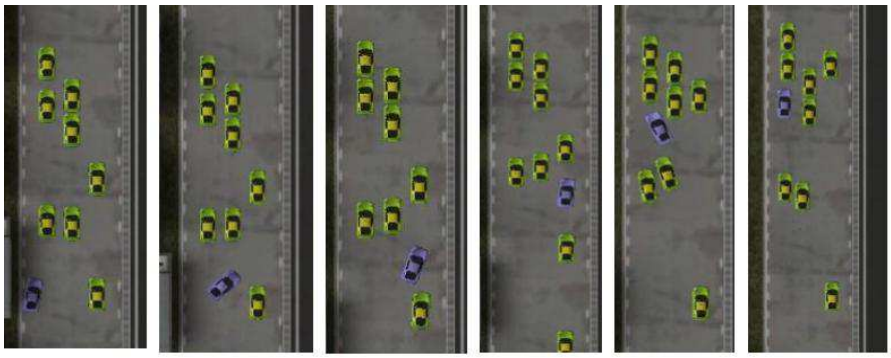

Fig. 3: Opportunistic behavior shown by our agent(blue) in presence of dynamically changing traffic. Our agent detects the free spaces and navigates in between the other cars and takes up the free spaces. This behavior is typical in scenes like Indian traffic.

Results and Observations 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

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

training we did not see any significant results. The logical explanation behind the difficulty in learning is the huge state space.

5.4 Blocking Behavior

Approach By blocking we mean the agent tries to block the car behind it from overtaking. This is a very hostile behavior which is not appreciated nor expected in common life.
A very important feature of RL is the fact that training conditions alter the results drastically. The agents learns by exploration and whatever conditions it is exposed to result in the final behavior. For blocking we had the same reward as overtaking but now during training our agent starts infront the other car. Eventually after 2k episodes it learns how to make sure that car behind never overtakes.
Since the car need not run ahead here, we do not use curriculum based learning. The agent is trained with one single car in the environment.
All the extra reward conditions are same as in table 1, apart from the overhauling condition, which has been removed here.

Results and Observations 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 itself. Table 4 shows that AI cars BT, Damned and Olethros could not be blocked by our Blocking Agent, on the other hand Berniw, Inferno, Illiaw and Tita could be blocked with 65% chances and InfHist, BerniwHist are easily blocked most of the times.

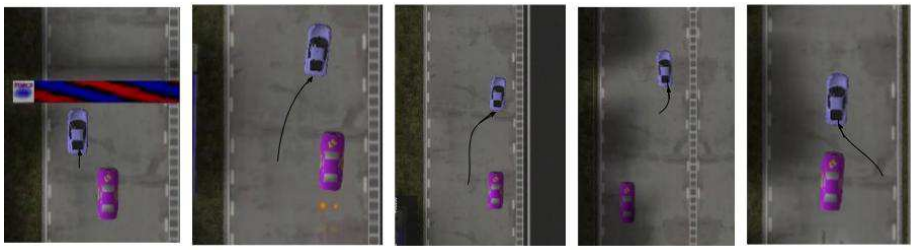

Fig. 4: Results of blocking behavior, in first three images, the purple car moves towards right to overtake our(blue) agent, our agent also moves towards the right to prevent the overtaking. In last two frames, the purple car is translating towards left and back to center and our car, also translates in center to block it from overtaking.

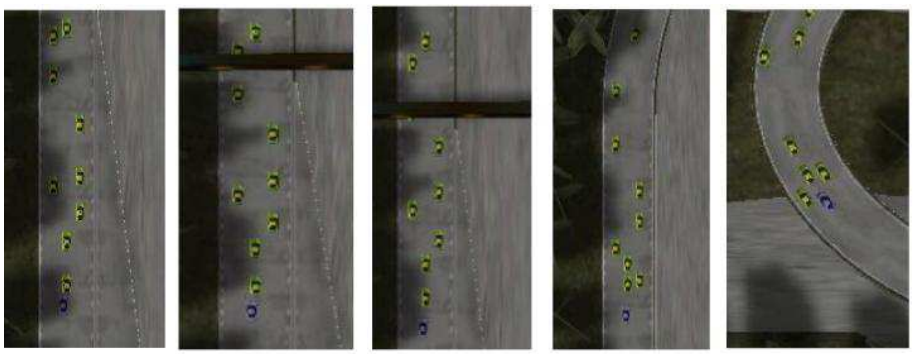

Fig. 5: This is an example of defensive behavior, whenever our agent(blue) is at a risk of colliding with any other car(green), it slows down. It takes care to not collide with cars in same lane as well as in adjacent lanes. For safety reasons, defensive behavior can be considered better than opportunistic.

Comparison with existing approach Blocking behavior has been targeted using Reinforcement Learning in [17]. Their approach is derived from work shown in [18]. 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.

Differences between the two approaches:

- 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.

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	35
Olethros	0.9202	100
Tita	2.0032	35

Table 4: 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.

5.5 Defensive Behavior

Approach 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 [34] agent's steer calculation:

$$steer = (10/PI) \times trackAngle - (0.10) \times trackPos \tag{8}$$

here $trackAngle$ is angle between car's heading angle and track axis, $trackPos$ is the relative position of car on the track. This agent is called defensive since it would never try to overtake anyone, it will avoid collisions by decreasing its own speed. 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. The extra reward conditions are same as in table 1.

Results and Observations 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. 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.

Table 5: Comparing various behaviors shown in the paper

Behaviour	**Reward function**	**State Space**	**Action Space**	**Number of episodes trained**	**Training Conditions**
Lanekeeping	$R_{Lanekeeping}$	29	3	1k-2k	Single car on track
Lanekeeping with restricted maximum speed	$R_{Lanekeeping}$, if vel<maxVel -900, otherwise	29	3	1k-2k	Single car on track
Highway Overtaking	$R_{Lanekeeping} + R_{overtaking}$	65	3	1k	4 cars ahead of agent car
	$R_{Lanekeeping} + R_{overtaking}$	173	3	1.5k	4 cars ahead of agent car
Driving in traffic/ Opportunistic behaviour	$R_{Lanekeeping} +$ $R_{overtaking}$, if $vel < maxVel$ -900, otherwise	65	3	4k	4 cars(Velocities approximately equal to the agent car's maximum velocity) ahead of agent car
Blocking	$R_{Lanekeeping} + R_{overtaking}$	65	3	2k	1 car behind the agent car
Defensive Behaviour	$R_{Lanekeeping}$	65	2	700	4 cars ahead of the agent car

6 Conclusion and Future Work

The main contribution of this research are the behavior driven agents. We show how RL can be used to develop agents which are not driven by any goal but by a behavior. We show how reward function is important in affecting the learned behavior. On top of everything, we show how can be speed up the process of learning by intelligently using Curriculum learning and Intrinsic motivation. We show the effectiveness of RL in dense unstructured environments. Our agent is able to navigate in dense, dynamic and diverse situations.
RL when used with the correct choice of reward in an environment which generates enough experiences, can give impressive results. The main driving force for any RL algorithm's behavior is the reward function and the environmental settings, observe how the results varied for Blocking and Overtaking behavior, they had same rewards but the environment setting was different, in overtaking the agent started from the end while in blocking it started from the beginning.
An important contribution using these behaviors would be to learn a meta function which decides which behavior to be followed. Using the meta function and these behaviors we can generate an end to end motion model for navigating safely in unstructured environments.

Since most of the learning takes place in an environment with other cars, we can speed up the learning by using the other car's experiences as well. In Asynchronous Actor-Critic Methods[35] multiple workers work together to update a single network, this way the network learns from all the agents in scene. Another interesting approach to efficiently speed up the training would be Distributed DDPG [36]. The given results can be hugely improved using Distributed Methods.

References

1. Caltagirone, L., Bellone, M., Svensson, L., Wahde, M.: Lidar-based driving path generation using fully convolutional neural networks. arXiv preprint arXiv:1703.08987 (2017)
2. Siam, M., Elkerdawy, S., Jagersand, M., Yogamani, S.: Deep semantic segmentation for automated driving: Taxonomy, roadmap and challenges. In: Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on, IEEE (2017) 1–8
3. Hadsell, R., Sermanet, P., Ben, J., Erkan, A., Scoffier, M., Kavukcuoglu, K., Muller, U., LeCun, Y.: Learning long-range vision for autonomous off-road driving. Journal of Field Robotics **26**(2) (2009) 120–144
4. Pomerleau, D.A.: Alvinn: An autonomous land vehicle in a neural network. In: Advances in Neural Information Processing Systems (NIPS). (1989)
5. Chen, C., Seff, A., Kornhauser, A., Xiao, J.: Deepdriving: Learning affordance for direct perception in autonomous driving. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 2722–2730
6. Wymann, B., Espié, E., Guionneau, C., Dimitrakakis, C., Coulom, R., Sumner, A.: Torcs, the open racing car simulator. Software available at http://torcs. sourceforge. net **4** (2000) 6
7. Salem, M., Mora, A.M., Merelo, J.J., García-Sánchez, P.: Driving in torcs using modular fuzzy controllers. In: European Conference on the Applications of Evolutionary Computation, Springer (2017) 361–376
8. Kim, T.S., Na, J.C., Kim, K.J.: Optimization of an autonomous car controller using a self-adaptive evolutionary strategy. International Journal of Advanced Robotic Systems **9**(3) (2012) 73
9. Kim, K.J., Seo, J.H., Park, J.G., Na, J.C.: Generalization of torcs car racing controllers with artificial neural networks and linear regression analysis. Neurocomputing **88** (2012) 87–99
10. Zhang, J., Cho, K.: Query-efficient imitation learning for end-to-end autonomous driving. In: AAAI Conference on Artificial Intelligence (AAAI). (2017)
11. Sallab, A.E., Abdou, M., Perot, E., Yogamani, S.: End-to-end deep reinforcement learning for lane keeping assist. arXiv preprint arXiv:1612.04340 (2016)
12. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. In: NIPS Deep Learning Workshop. (2013)
13. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: International Conference on Learning Representations (ICLR). (2016)
14. Xia, W., Li, H., Li, B.: A control strategy of autonomous vehicles based on deep reinforcement learning. In: Computational Intelligence and Design (ISCID), 2016 9th International Symposium on. Volume 2., IEEE (2016) 198–201
15. Sharifzadeh, S., Chiotellis, I., Triebel, R., Cremers, D.: Learning to drive using inverse reinforcement learning and deep q-networks. In: NIPS workshop on Deep Learning for Action and Interaction. (2016)
16. Ngai, D.C., Yung, N.H.: Automated vehicle overtaking based on a multiple-goal reinforcement learning framework. In: IEEE Intelligent Transportation Systems Conference (ITSC). (2007)
17. Huang, H.H., Wang, T.: Learning overtaking and blocking skills in simulated car racing. In: IEEE Conference on Computational Intelligence and Games (CIG). (2015)

18. Loiacono, D., Prete, A., Lanzi, P.L., Cardamone, L.: Learning to overtake in torcs using simple reinforcement learning. In: IEEE Congress on Evolutionary Computation (CEC). (2010)
19. Gu, S., Holly, E., Lillicrap, T., Levine, S.: Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: Robotics and Automation (ICRA), 2017 IEEE International Conference on, IEEE (2017) 3389–3396
20. Phaniteja, S., Dewangan, P., Guhan, P., Sarkar, A., Krishna, K.M.: A deep reinforcement learning approach for dynamically stable inverse kinematics of humanoid robots. arXiv preprint arXiv:1801.10425 (2018)
21. Kaushik, M., Prasad, V., Krishna, K.M., Ravindran, B.: Overtaking maneuvers in simulated highway driving using deep reinforcement learning. In: Intelligent Vehicles Symposium (IV), 2018 IEEE, IEEE (2018)
22. Lau, Y.P.: Using keras and deep deterministic policy gradient to play torcs. `yanpanlau.github.io/2016/10/11/Torcs-Keras.html` (2016)
23. Konda, V.R., Tsitsiklis, J.N.: Actor-critic algorithms. In: Advances in Neural Information Processing Systems (NIPS). (2000)
24. Bhatnagar, S., Sutton, R.S., Ghavamzadeh, M., Lee, M.: Natural actor–critic algorithms. Automatica (2009)
25. Ghavamzadeh, M., Engel, Y.: Bayesian actor-critic algorithms. In: International Conference on Machine Learning (ICML). (2007)
26. Uhlenbeck, G.E., Ornstein, L.S.: On the theory of the brownian motion. Physical review (1930)
27. Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: International Conference on Machine Learning (ICML). (2009)
28. Narvekar, S.: Curriculum learning in reinforcement learning:(doctoral consortium). In: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems (2016) 1528–1529
29. Singh, S., Lewis, R.L., Barto, A.G., Sorg, J.: Intrinsically motivated reinforcement learning: An evolutionary perspective. IEEE Transactions on Autonomous Mental Development **2**(2) (2010) 70–82
30. Achiam, J., Sastry, S.: Surprise-based intrinsic motivation for deep reinforcement learning. arXiv preprint arXiv:1703.01732 (2017)
31. Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., Munos, R.: Unifying count-based exploration and intrinsic motivation. In: Advances in Neural Information Processing Systems. (2016) 1471–1479
32. Yoshida, N.: Gym-torcs. `github.com/ugo-nama-kun/gym_torcs` (2016)
33. Loiacono, D., Cardamone, L., Lanzi, P.L.: Simulated car racing championship: Competition software manual. arXiv preprint arXiv:1304.1672 (2013)
34. Edwards, C.X.: 2015 snakeoil competition entry. `http://xed.ch/p/snakeoil/2015/` (2015)
35. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: International conference on machine learning. (2016) 1928–1937
36. Zhang, S., Zaiane, O.R.: Comparing deep reinforcement learning and evolutionary methods in continuous control. arXiv preprint arXiv:1712.00006 (2017)