Reinforcement and Imitation Learning for Multi-Agent 3D Racing: A Comparison

Ryan Camilleri
Department of Artificial Intelligence
University of Malta
Msida, Malta
ryan.camilleri.18@um.edu.mt

Daniel Attard
Department of Artificial Intelligence
University of Malta
Msida, Malta
daniel.attard.18@um.edu.mt

Andrew Magri
Department of Artificial Intelligence
University of Malta
Msida, Malta
andrew.magri.18@um.edu.mt

Francesco Borg Bonello
Department of Artificial Intelligence
University of Malta
Msida, Malta
francesco.borg.18@um.edu.mt

Daniel Cherrett
Department of Artificial Intelligence
University of Malta
Msida, Malta
daniel.cherrett.18@um.edu.mt

***Abstract*—Vision-based approaches have dominated the scene of Reinforcement Learning (RL) when applied to autonomous driving agents due to being a cheaper alternative. Addressing a lack of research on other techniques, this paper introduces a novel sensor-based racing environment for multiple agents. The Soft-Actor Critic (SAC) and Proximal Policy Optimization (PPO) RL algorithms are applied to this environment to determine whether on-policy algorithms are better suited towards autonomous sensor-based driving, or if off-policy techniques are superior. Hybrid approaches through Imitation Learning are also explored which apply Generative Adversarial Imitation Learning (GAIL) and Behavioural Cloning (BC). Results indicate that the off-policy SAC models are better suited towards sensor-based racing by adopting a competitive approach. Additionally, these are also shown to be better at avoiding undesired actions such as collisions with other cars. On the other hand, PPO agents have favoured a cooperative approach which led to safer track navigation at the expense of velocity. Such behaviours could all be fully appreciated within the paper's video (see https://vimeo.com/513575085).**

***Index Terms*—Reinforcement Learning, Imitation Learning, Continuous Actions, 3D Racing Environments, Multi-Agent**

I. Introduction

Recent advancements in Machine Learning (ML) have made intelligent agents much more accessible and efficient when compared to traditional approaches. Several applications have used ML to simplify and/or automate certain processes which would otherwise require a human. Games [1], engineering applications [2], healthcare applications [3] and many other domains have researched and determined the usefulness of artificial intelligence (AI) when applied to the problem at hand. Autonomous driving is one such problem which has drawn great attention from both industrial and academic communities due to its potential of revolutionising transportation systems.

Reinforcement Learning (RL), an area in ML, is one such way in which agents could be made autonomous through training. These would be required to perform specific actions within a simulated environment which would lead to a certain goal [4]. The behavior that is learnt is called a policy, which is an optimal mapping of observations to actions achieved using a reward function as a supervision signal. Repeating this for a number of episodes, would train the agent into maximising their reward, thereby learning how to complete the specified task more efficiently. In complex environments however, this training may take large amounts of time to converge and agents may also get stuck on local minima [5].

A possible solution to this, is to use Imitation Learning (IL), which provides agents with knowledge from expert demonstrations [6]. Through these, agents are given a head start in the training phase. However, if policies were to be learnt solely from these examples, the agent may not adapt to unseen situations as well as when trained solely on RL. As stated by Hussein et al. [5], RL ensures that agents are more adaptable to unseen environments, which negates the need to specifically train models for every environment from scratch. Therefore, by combining the two techniques together, a model could be trained quicker to resemble human driving, and parameters could be fine-tuned through RL to alleviate errors within such demonstrations.

A. Problem Definition

Environment perception of autonomous driving systems favours vision-based approaches as opposed to the sensor-based LIDAR, as these are a cheaper alternative [7]. Therefore most studies have focused their work on applying image-based techniques using convolutional neural networks (CNNs) and other techniques to solve this problem [8], [9]. As a consequence, few studies have applied RL to sensor-based driving simulations. Moreover, no studies were found which investigate which RL techniques are better suited towards 3D racing environments with multiple agents. It is therefore paramount that research is made on using this sensor-based approach so that real-world domains could benefit from safer, more optimal systems.

B. Aims and Objectives

This paper aims to determine whether on-policy RL algorithms are better suited to autonomous sensor-based driving or if off-policy techniques are more optimal. These techniques will be applied in a custom 3D racing environment, constructed specifically to handle continuous actions with multiple agents. Another aim of this paper is to determine how IL could reinforce training and if result could be improved through it. These will be accomplished by fulfilling the following objectives:

- **Objective (O1)**: Train agents using Proximal Policy Optimisation (PPO) for evaluating the effectiveness of on-policy techniques on the environment.
- **Objective (O2)**: Train agents using Soft-Actor-Critic (SAC) for evaluating the effectiveness of off-policy techniques on the environment.
- **Objective (O3)**: Apply Imitation Learning to the techniques to determine the efficacy of its use.

II. Literature Review

This section aims to deliver an overview of RL techniques when applied to racing and car simulations. IL will also be reviewed to determine the current state-of-the-art in this domain.

A. Reinforcement Learning in Racing Environments

Reinforcement learning algorithms can be spilt into two categories; on-policy and off-policy techniques. On-policy techniques collect and improve their policy based on samples obtained from the same policy. These techniques make use of an evaluation function such as value estimate or Q-function to estimate the rewards that will be obtained. On the other hand, off-policy techniques use all the samples gained from different policies in a bid to learn the global function. They update their policy based on hypothetical actions [10] and therefore, they "learn one policy while they follow another" [11]. Both on-policy and off-policy RL algorithms have been applied to the field of autonomous driving.

All RL algorithms face the exploitation and exploration dilemma. The algorithm has to find a balance between exploiting its current knowledge, choosing actions which are known to give high rewards, while also exploring new actions and their resultant reward. This may or may not provide higher rewards further down the line [10]. Holubar and Wiering [12], compare PPO with Sampled Policy Gradient (SPG) in a continuous action and state space racing environment. The authors also interpret the difference between on-policy and off-policy techniques, whereby the former eliminates bad actions (which may not always push the technique towards more optimal policies) while the latter moves towards the best possible actions [12]. After exploring the use of experience replay (ER), SPG improved its results significantly in a short computational time, exceeding PPO's results which was superior without ER. The authors also propose two issues as to why PPO does not make efficient use of ER in continuous action environments [12].

Balaji et al. [13], use DeepRacer to train a PPO algorithm which perceives its environment through a grayscale camera and outputs discrete throttle and steering actions. Robust results were observed from the evaluation of the algorithm on a real-life track. Folkers, Rick and Buskens [14], also successfully control an autonomous vehicle having a continuous and bounded state and action spaces, using a Markov Decision process with PPO.

Loiacono et al. [15], focus on overtaking behaviour in car racing using a Q-Learning algorithm. State observations were discretized since Q-Learning cannot be implemented using continuous values. The authors compared their Q-Learning algorithm to the best agent provided in the The Open Racing Car Simulator (TORCS) environment. The Q-Learning algorithm outperformed the best agent in all of their testing. Q-Learning was also proposed with success for navigating around roundabouts [16]. Liu et al. [17] propose of deep Q-network (DQN) and double deep Q-network (DDQN) to determine the trajectory and velocity of an autonomous vehicle. The authors note that overestimation of action values is a key issue of Q-learning algorithms. DDQN makes use of two Q functions, one for estimating the Q-value and the other for obtaining the superior action. They also note that the DDQN algorithm produces ideal results at the same complexity of DQN whose complexity is more optimal than that of Q-Learning. From their findings, the authors conclude that the DDQN produces more stable results when compared to DQN due to the overestimation issue. A deep Q-Learning algorithm with filtered experience replay (DQFE) was proposed by Xia, Li and Li [18] to control a self driving car. The authors compared their algorithm to a neural fitted Q-iteration (NFQ) algorithm and found that the proposed DQFE algorithm increased the stability and time consumption by 32% and 71.2% respectively. On a similar note, comparing Deep Q Networks (DQN) and Deep Deterministic Actor Critic (DDAC) for end-to-end autonomous driving, Sallab et al. [19] conclude that DDAC offers smooth actions due to the continuous policy. DQN resulted in better performance and faster convergence but suffered from abrupt steering actions due to the tile coding of actions.

Wang, Jia and Weng [20] propose using a Deep Deterministic Policy Gradient (DDPG) algorithm for autonomous driving in TORCS which obtains good performance along with functional safety. DDPG is composed from parts of; deterministic policy gradient, actor-critic algorithms and deep Q-Learning, and can be applied to continuous state and action spaces. Huang et al. [21] also propose an end-to-end autonomous driving algorithm using the DDPG algorithm which is trained and evaluated using TORCS. The authors visualise which input has the greatest effect on the agents' actions, and deduce that steering is mainly inferred from the distance distribution. Furthermore, acceleration is deduced from the distance in front of the agent, while braking is mainly concluded from speed in

the x direction. Ke, Yanxin and Chenkun [22], compare DDPG and SAC for autonomous driving around maps with various levels of difficulty. The authors propose a variable experience pool allowing the SAC model to replace old data with new interactive data. This improves training speed by increasing the "proportion of successful experience and failure experience" [22]. Apart from this, their NEW-SAC algorithm also produces a lower error rate when compared to the DDPG algorithm.

An Asynchronous Advantage Actor-Critic (A3C) RL algorithm was used by Pan et al. [23] to train an autonomous vehicle around the TORCS simulator. A3C is an on-policy algorithm combining different RL techniques and uses asynchronous parallel threads where each thread has a separate copy of the environment [24]. Realistic images are generated from virtual images and are inputted into the A3C algorithm. The A3C algorithm is outperformed by a supervised model, which has a similar architecture to the RL model, but requires a large amount of data to train.

B. Imitation Learning with Reinforcement Learning

Whereas RL starts off with a random policy, IL could help in reducing the complexity of the search space by eliminating bad solutions and starting off at a local optima [25]. However, one of the earliest studies by Schaal contradicts this statement when using IL as a pre-training step [26]. The author shows that not every learning approach benefits from using prior knowledge, as only when applying learning to linear quadratic regulator problems, was a significant benefit observed. Additionally, policies could also collapse due to over-fitting to demonstrations instead of learning a more generalised solution [6]. This could be mitigated by using large amounts of varied demonstrations as was done by Silver et al. [27]. The time and effort required to gather expert knowledge however, would be significant and is thus, not ideal. Hussein et al. determine an alternative to manual data collection in their review, which uses pre-trained agents on RL to generate demonstrations automatically [5].

Different IL methodologies have been applied which combine expert data with RL, in search for more optimal training procedures. Ross et al. propose DAGGER [28], an algorithm which iteratively trains a stationary deterministic policy by polling experts. Even though DAGGER is shown to outperform previous approaches, the technique is still impractical for long training runs as it is dependent on additional expert feedback throughout the training phase. The same cannot be said to solutions adopting Inverse Reinforcement Learning (IRL), as these are shown to eliminate expert interaction during training [29]–[31]. Inspired by this, Generative Adversarial Imitation Learning (GAIL) was proposed, which improves upon IRL by managing to directly extract a policy from the data without the need of expert support [32]. Despite Ho and Ermon's achievements, the authors still acknowledge the limitation that their solution requires increased environment interaction which increases training time. As an improvement, Behavioral Cloning (BC) is suggested [33] as it requires no environmental interaction and initialises the policy with sub-optimal parameters.

A recent study argues however, that supervised learning methods based on BC suffer from a distribution shift since they greedily imitate the demonstrated actions [34]. As agents drift away from demonstrated states, these encounter out-of-distribution states from which no knowledge is known on how to go back to the demonstrated states. As an alternative to both GAIL and BC, the authors propose their Soft Q Imitation Learning (SQIL) which provides agents with an incentive to return to demonstrated states, on encountering new states [34]. This is shown to achieve superior results over BC and is comparable to GAIL. Li et al. improve on hybrid approaches like GAIL with their proposed Observational Imitation Learning (OIL), which converges to a high performance policy without too much exploration [35]. Additionally, the technique is also capable of supporting online training and automatic selection of behaviours through observing multiple teachers. Being robust, OIL is capable of determining the advantages and disadvantages of a teacher's behaviour, and only update the policy with behaviours that are deemed to be optimal. This IL variant has proven to be among the state-of-the-art by adopting a novel approach to IL and also achieving superior results when compared to BC, DAGGER and even DDPG.

III. Methodology

In the previous section, a review of Reinforcement Learning techniques was carried to determine the state-of-the-art in racing domains. An overview of different approaches to Imitation Learning was also given. This section will describe the methodology adopted for constructing and optimising the racing environment. Prior to this, the RL techniques that will be evaluated in this study will be mentioned and reasons for these choices will be clarified. Following these, the reward system applied to agents will be described in detail and reasons for applying positive/negative rewards in some areas instead of others will be given. The section ends with an explanation of what IL techniques are used and how demonstrations are recorded.

A. Reinforcement Learning Techniques

1) On-Policy Technique - Proximal Policy Optimization: The PPO algorithm was developed by Schulman et al. [36], enabling multiple epochs of minibatch updates while also being simpler to implement with better sample complexity. It improves on trust region policy optimization (TRPO) by combining TRPO's data efficiency and dependable performance by only making use of first-order optimization. There are two versions of PPO, PPO-Penalty and PPO-Clip, where the latter provides superior performance and is the implementation used in ML-Agents [36]. PPO uses two neural networks, the policy and value network. The policy network is responsible for choosing the next action to take while the value network calculates the expected cumulative reward [13]. The authors also note that PPO outperformed most other methods on the continuous control environments that they tested [36].

As described in Section II, Schulman et al. [36] conclude that PPO outperforms other methods in continuous action environments. Furthermore, Holubar and Wiering [12], and Tran and Bae [37], successfully applied PPO to continuous action and state space driving environments. Despite these, limited research is available in such driving applications and therefore PPO was chosen as the on-policy technique to be researched in this paper.

2) Off-Policy Technique - Soft Actor Critic: SAC developed by Haarnoja et al. [38], is an actor-critic algorithm designed to achieve sample-efficient learning and stability by reusing collected data and maximising entropy on continuous state and action spaces. The algorithm combines off-policy actor-critic training with a stochastic actor and two Q-functions which results in performance and efficiency levels that exceed those of DDPG [38]. The authors also note that this algorithm manages to converge to an optimal policy irrespective of the policy parameterisation. Due to its stability without precise tuning of hyperparameters, SAC performs superior on more complex problems were other methods require careful hyperparameter tuning. The reward scale greatly effects the performance of the algorithm with the right scale, finding a balance between exploration and exploitation.

Despite DDPG being unstable and difficult to extend to more complex tasks [38], such technique is still widely used successfully for the task of autonomous driving. SAC is presented as an improvement to DDPG as proved by Ke, Yanxin and Chenkun [22]. These two factors and the limited research available on the application of SAC to the driving domain, lead to the choice of using the SAC algorithm as the paper's off-policy technique.

3) Training: Unity's ML-Agents was used for training due to the extended control provided along with it's ease of compatibility with Unity, which was required to build our custom environment [39]. This package also boasts the required PPO and SAC techniques by this paper with additional extensions of IL techniques such as GAIL, BC and Curiosity.

Being recently released, ML-Agents is still being actively developed and therefore some technical implementation issues still arise. Despite Haarnoja et al. [38] noting that training SAC is significantly faster than PPO due to the large batch sizes required. In our implementation however, it was noted that training with SAC was considerably slower when using ML-Agents. In order to speed up training, 4 training environments were used. Each training environment was composed of 5 agents racing against each other, whose goal is to decrease lap times. During training, a new episode is started whenever an agent; completes a race with 5 laps, topples over or reaches the maximum steps which was defined to be 10k steps. The maximum number of steps was initially set to 50k steps. After observing that a lap could be completed in around 500 steps, the max was reduced to 10k to limit the agent exploring inferior policies and getting stuck on local minima. Initially, the agents were trained for a maximum of 30M steps. However for each technique, these were observed to converge well before this limit. This value was therefore reduced to 15M steps, which decreased the training time required substantially whilst still achieving convergence. The config files used for training the SAC and PPO models can be observed within the Appendix. Additionally, config files of the IL boosted techniques could also be observed in the Appendix.

B. Game Environment and Mechanics

The decision to what environment was to be used closely followed after determination of which RL techniques were to be evaluated in our study. Research was done on open-source car simulators like TORCS as these are common baselines for RL [40]. Despite this, an alternative was still required however, as more control of the environment was needed by our study.

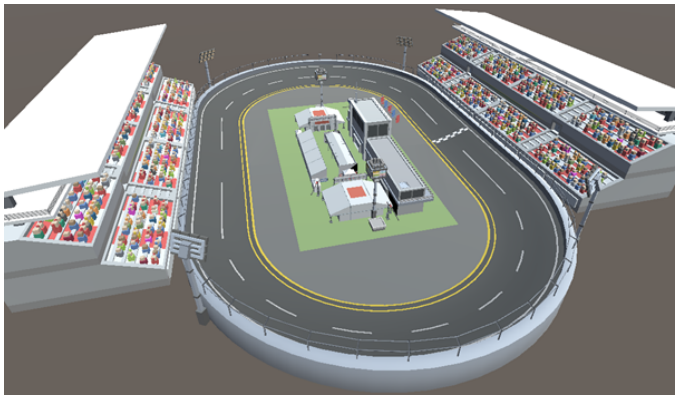


Fig. 1: Oval track racing environment

1) Racing Environment: The Unity3D game engine was used for the custom environment [41]. Using this allowed for a higher degree of testing and control, especially when changing environmental parameters and observing the impact this would have in the training phase. Instead of creating an environment from scratch, the Simple Racer asset by Synty Studios [42] was used as a starting point. Since the asset was mostly aimed towards low-poly 3D environments, Probuilder [43] was used to smoothen the track so that this would have a more streamline bend. Additionally, the use of Probuilder was also beneficial when re-constructing object meshes after smoothing. Having full control of object meshes allowed for individualisation of specific parts of the track mesh itself. Applying this knowledge, the outer border of the track was made separate from the track's mesh, so that when cars collide with it, the cars would register a border collision instead of a collision with the whole track. Further observations when testing with a manually driven car, showed that in areas where banking was highest, the car body was getting stuck in between the flat and curved parts of the track. To resolve this issue, the track was modified vertically to have less banking and was also widened to allow more space for car reversing and orientation

fixes when crashing. The end result can been seen in Figure 1.

Fig. 2: Racing cars used for training.

2) Car Mechanics: To control the cars shown in Figure 2, the horizontal and vertical input axes are used. These serve as the agents' continuous action spaces which determine their motion. Whereas the horizontal input is responsible for steering the wheels in the horizontal direction, the vertical input is responsible for applying an accelerative or decelerative motor torque. These are applied to Unity's wheel colliders, which are wrapped around every car's wheels, as these simulate real world physics perfectly. Using these wheels, parameters such as suspension, forward friction, sideways friction and more, could be controlled. Several tests were done in order to optimise and balance the frictional forces mentioned. Additionally, parameters like the max steering angle and speed applied to the wheel were also defined.

3) Game Modes: By using Unity ML-Agents, agents could be controlled both through inference and through heuristics. This allowed our environment to support both multi-agent racing with the user as the observer and the user as a racer among other agents. The latter was essential when carrying out qualitative evaluation which can be seen in *Section IV*.

C. Environment Optimisation

Following the setup of basic car controls, work was shifted towards modifying the environment so that the agents could obey standard rules. In typical racing scenarios, cars should avoid colliding with other cars and obstacles. Moreover, cars should also prioritise racing on the track itself instead of the inner flat region.

1) Collision Objects and Checkpoints: To accomplish the above requirements, the outer border of the track was separated from the whole track object so that a unique mesh collider could be defined to it. To apply a penalty to cars that enter the flat area, a capsule collider which encapsulates all the flat region was used as a trigger. Cars were also given a mesh collider which wraps around their body so that collision events could be registered on car collision. Lastly, all buildings located in the middle of the track were also given colliders to control any collisions made with these. When using this environment and training agents, it was observed that some went in the opposite direction and some even reversed to complete the whole track. This behaviour was addressed by applying checkpoints throughout the track and on the starting line. Whenever an agent enters such checkpoints, a reward is given to the agent based on its velocity and direction. The way in which rewards are calculated is described in further detail in Section III-D. With all these modifications, the final track with the implemented colliders may be seen in Figure 3.

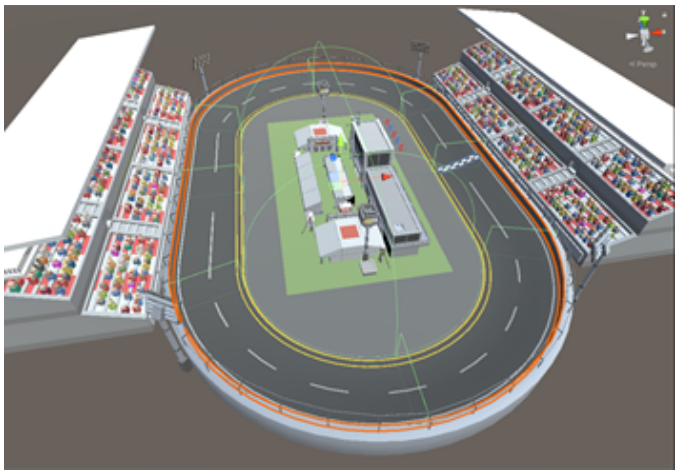
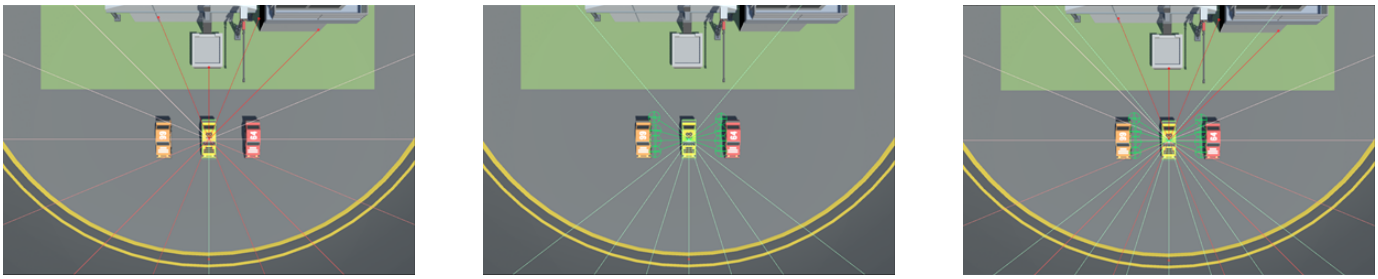


Fig. 3: Race track with checkpoint, inner and outer border colliders.

2) Agent Sensors and Observations: In order for agents to detect and observe the above colliders, 3D ray perception sensors were utilised. Initially, only one sensor object was used to detect every obstacle surrounding the agent. Testing revealed however, that this method was not optimal as agents were unaware of cars behind detected colliders. This was specifically a problem when adding checkpoints throughout the track, since agents often sped through the checkpoints' plane, without any regard for what was behind. Therefore, an additional sensor masked to strictly detect cars, was added to each agent. This allowed for the obstacle sensor to see incoming obstacles through vehicles, and the car sensor to see incoming vehicles through obstacles. For each sensor, 8 rays per direction were used and each was given a vertical offset to account for track banking. Whereas the obstacle sensor was set to cover the whole 360 degree range of the car, the car sensor was set to a max of 140 degrees per direction. This was done to remove boosting, as cars often accelerated to avoid incoming cars, which often led to unwanted crashing. The obstacle sensor, car sensor and both combined can be



(a) Obstacle sensor. (b) Car sensor. (c) Combined sensors.

Fig. 4: 3D Ray Sensors

seen in Figures 4a, 4b and 4c respectively. Apart from such sensor observations, each agent was given 10 additional vector observations so that the agent would continuously get information about its velocity, rotation and relative position to the starting line.

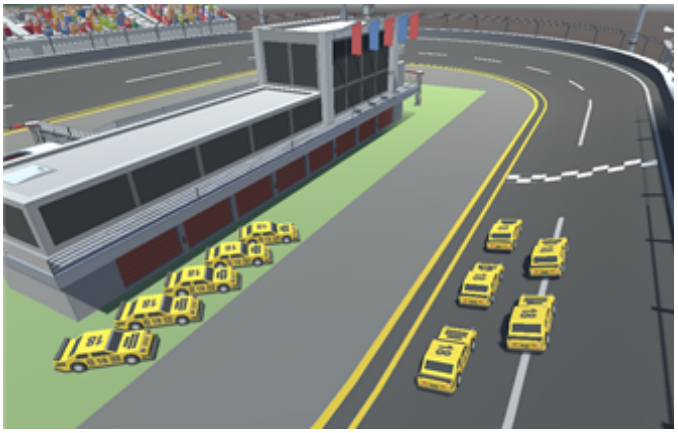


Fig. 5: Agent spawn positions.

3) Managing Episode Resets and Spawn Positions: Originally, agents were set to respawn at their starting positions. This would either occur when the agent's episode is forcefully reset or when the agent finishes the race. It was observed however, that when agents were being reset to these positions, the cars would spawn over those that hadn't yet been reset, or collide with incoming cars that were still racing. This affected training results greatly and so, a pit stop was introduced as a solution. On launch, the cars are given random starting positions from the set of positions available, including a pit stop slot. Should an agent be reset before it finishes the race, it is reset to its pit stop instead of its starting position. Since previous lap information is reset and a startling line observation is given to the agent, the behaviour is as expected and is shown to resemble closely real world behaviour. To avoid spawning cars that finished their race back to their pit stop, a spawn manager was defined to disable each car that has finished its race and when all cars are finished, these are reset to their starting line positions. Such positions along with pit stop positions can be seen in Figure 5.

D. Agent Reward System

Rewards play an important part in RL to entice a certain behaviour more than others. Restricting unwanted behaviour, such rewards must be given negatively and positively otherwise.

1) Lap Reward: To motivate agents into performing laps, a reward is given each time the agent passes the starting line. The race is set to start for each car whenever it goes over the initial starting line. At this point a reward is given to the agent and a timer is started to find the time per lap. When the agent completes a lap, the reward given takes into account the number of checkpoints passed and the time taken. The smaller the lap time and the higher the number of checkpoints passed, the higher the reward. With every lap completed, these values are reset so that the agent tries to achieve better results in the following lap. No rewards based on race positions were given, as it is believed that by using lap time as a reward signal, the agents would still explore the possibility of over-taking to achieve shorter times.

2) Directional Handling: As previously described, checkpoints were defined throughout the track to reward agents which drive in the right direction. The reward given when passing these, is also set to be based on the lap time, so as to encourage shorter times in between checkpoints. To determine if the agent is driving in the right direction, the angle between the checkpoint's forward axis and that of the car is determined. If it is larger than or equal to 90 degrees, the car is given a negative reward, otherwise, a positive reward is given. To determine whether agents are reversing through checkpoints, the local velocity of cars is found and if the velocity is smaller than 0 (reversing) the agents are penalised. Even though this was implemented, the agents were still observed to repeatedly reverse back and forth through checkpoints to maximise their reward. The reason for this was that an exploit was being found by entering checkpoints, and reversing half way through. In such cases, when exiting the checkpoint, and driving back through,

the agents were only being rewarded and never penalised. To control this, exit event listeners were defined to penalise agents by removing the initial reward given, when entering the specific checkpoint. Since lap time is used as a reward signal, when the agent goes back through the checkpoint, less reward is given from when it initially passed through it to ensure that the behaviour isn't repeated in the future.

3) Collision Handling: To discourage collisions, a negative reward is given whenever an agent collides with another car, building or the outer border. When entering the inner border trigger, a negative reward is also given which is removed when exiting to encourage agents to go back to continue the race. This wasn't enough however, as agents were observed to spend large amounts of time within the flat region which was not ideal. When training the agents it was also observed that these had developed a cooperative solution whereby the agents sacrificed the initial negative reward on collision with other cars to maximise there reward in the long run. By sticking together, the agents seemingly created a larger mass which was less prone to sliding and crashes thereby maximising reward. To fix this, a small negative reward was applied per frame to agents that stayed in collision with other objects. However, since the negative reward applied to car collision was the same as when hitting a building or outer border, the cars were afraid to overtake leading cars due to the possibility of collision. By reducing the negative reward given substantially, the agents were observed to take more risks compared to before.

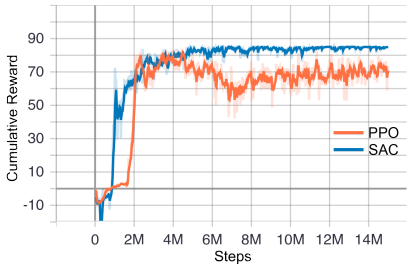
4) Car Topple Handling: In cases where agents accumulated large velocities and took sharp turns, these were observed to topple over and get stuck since their wheels wouldn't be touching the ground. By checking the angle between the local up axis of the agent's body with the global up axis this event was detected and handled whenever the angle is greater than 75 degrees. When toppled, the agent is reset to its pit stop and a negative reward is applied.

E. Imitation Learning

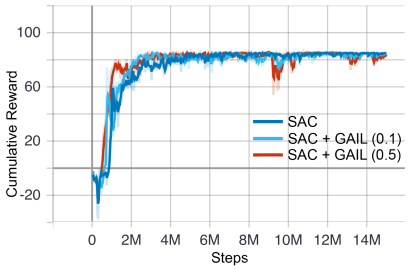
1) Applied IL techniques: This study applies Ho and Ermon's GAIL [32] which uses an adversarial approach to reward agents for behaving similar to the given demonstrations. A discriminator is used to distinguish whether an action made by the agent is from a demonstration or whether it was produced by the agent. By examining the new action, a reward is provided based on how close the discriminator believes it to be to the recorded actions. Such discriminator is trained to better distinguish between demonstrations and agents states, as training progresses. In this way, agents improve by trying harder to fool the discriminator which keeps getting stricter. By mixing the GAIL reward signal with an extrinsic reward, which was used for RL training, the learning process could be guided to find a more optimal behaviour that resembles that of the demonstrations. As suggested by the authors [32], BC could bolster training results using GAIL, by initialising the policy with parameters to avoid the increased interaction required by GAIL. To evaluate this, BC is also used in this study for determining its effectiveness. Both techniques are applied using the ML-Agents toolkit which allowed for ease of integration with the RL models mentioned previously.

2) Recording Demonstrations: To set up the recording environment, all agents were applied the SAC trained model, as this was observed to achieve more human-like behaviour and shorter lap times when compared to PPO. Initially, recording of demonstrations was done using a heuristic approach whereby an expert manually drove a car and competed against the other agents. Not only was this inefficient due to the long amount of time needed to gather a substantial dataset [27], but the demonstrator was also prone to more slips and collisions than the agents were. As an alternative, the process was automated by recording the agent's actions through inference as suggested in [5]. To ensure variation in demonstrations, the environment was also modified to reset pit stop and starting line positions after every race. It was also found, that by disabling the recording agent and re-enabling it when a new race is started, an infinite import loop was being set which stopped the recording process. As a fix, the environment was reset whenever the agent finished its race irrespective of whether the other agents finished their race or not; Thereby avoiding the disabling and re-enabling of the demonstrator.

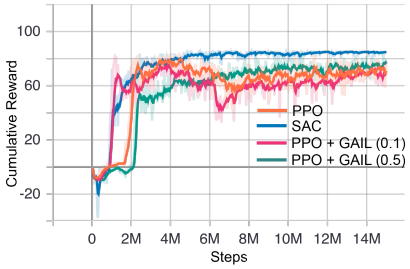
3) Hybrid Training: Since IL is by definition off-policy, it is typically limited to off-policy algorithms [44]. This means that hardly any significant benefit will be observed when applying IL to on-policy algorithms like PPO. Despite this, by using observations from SAC, this paper applies IL to both PPO and SAC to validate the above statement and determine improvements if any. Carrying this forward, two further training sessions were done on each algorithm. The first session involved the inclusion of GAIL among the standard extrinsic reward. Two training instances were performed on each algorithm to evaluate whether giving more weight to GAIL was more effective when compared to having less weight. In each case, GAIL's strength is considerable less than that of the extrinsic reward. This was ensured so that resemblance to observations isn't given priority and exploration would still be encouraged. The first instance applied 1.0 and 0.5 strengths to the extrinsic reward and GAIL respectively, whilst the second modified GAIL's weight to 0.1. The second session extends the first by applying BC which was also given the same strangths as GAIL in this individual training instances. However, instead of using BC throughout the whole training phase, this is only used as a pre-training step as suggested in [32], for the first 5M steps from the total 15M. Unfortunately, due to ML-Agents restrictions, BC could not be applied to SAC and so, this was only trained on PPO.



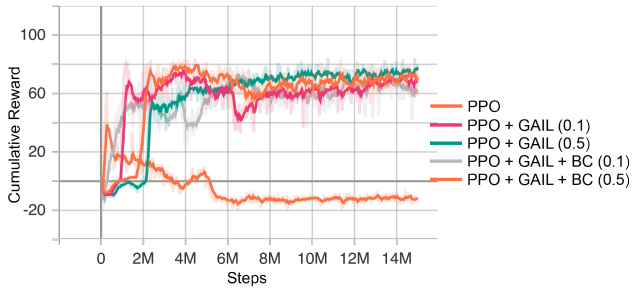
(a) Rewards of standalone PPO and SAC models.



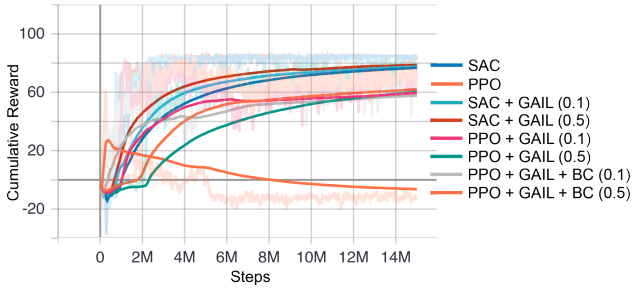
(b) Rewards of all SAC models.



(c) Rewards of PPO with GAIL compared to SAC and PPO.



(d) Rewards of all PPO models.



(e) Rewards of all SAC and PPO models combined.

Fig. 6: Model Cumulative Rewards.

IV. Results and Evaluation

Prior RL research has applied quantitative and qualitative evaluation as both are important in determining if natural and optimal behaviours are achieved by agents [5]. This paper applies such knowledge by performing a quantitative evaluation from results obtained after applying the same model to five agents racing against each other. For obtaining a fair sample of results, the models were evaluated on 20 races with 5 laps each, 100 laps in total. With inspiration from [45], the model's ability to avoid undesired actions are also measured by counting the number of collisions made with obstacles, such as car collisions. Moreover, the time taken to complete a race is also compared to determine the fastest model among all the trained models. Qualitative evaluation is also applied by observing the driving behaviour of agents and comparing it with that of humans. This part will also highlight specific patterns observed during inference for every RL algorithm.

A. Result Discussion - Models

As shown in Figure 6a, SAC was able to converge faster and generate a better policy than PPO, resulting in a higher cumulative reward in the end. After 15 million steps of training, a 24% increase in cumulative reward and a 19% decrease in episode length is obtained by SAC when compared to PPO. It should be noted that while SAC took approximately 6.5 hours to train, PPO was able to train in half that time but this can be attributed to an issue with ML-Agents as described in Section III-A3.

For applying IL to each model, SAC was used to produce 3 hours worth of demonstrations automatically, due to it's superior performance. When applying GAIL to the SAC algorithm, results show that double the training time is required to reach the maximum 15M steps. Additionally, no significant improvements were observed, both when using 0.5 and 0.1 IL weighting. As illustrated in Figure 6b, only a 3% improvement was obtained in cumulative reward and episode length between the standalone SAC model and the SAC+GAIL(0.5) model. Apart from obtaining higher rewards and also converging earlier, no significant improvement is obtained at the end of training. This behaviour confirms IL's ability to aid algorithms in converging to specific observations, but shows that this is mostly useful in earlier stages of training since hardly any new knowledge is obtained in later stages. IL's ability to explore the search space further is also shown within Figure 6b, when observing the reward spikes. Proportionally, since the SAC+GAIL(0.1) model has a lower GAIL weighting, no significant variations were observed from the original SAC model. This shows that in order to achieve some improvements over the standard, the weighting must be significant since otherwise, the extra time required would be impractical.

Applying GAIL to the PPO algorithm, similar training times were achieved to those of the standalone model. Despite very similar cumulative rewards, a 10% decrease in episode length was noted between the standalone PPO model and the PPO+GAIL(0.1). As shown in figure 6c, despite the similar final cumulative rewards, the 0.1 weight version obtained higher rewards quicker which highly resemble SAC's pattern. This

	PPO	SAC	SAC + GAIL (0.1)	SAC + GAIL (0.5)	PPO + GAIL (0.1)	PPO + GAIL (0.5)	PPO + GAIL + BC (0.1)
Border Collisions	1	0	0	0	1	1	0
Car Collisions	50	5	22	5	48	20	28
Race Time	72.0s	65.2s	67.5s	65.3s	69.9s	71.4s	78.3

TABLE I: Average of border collisions, car collisions and race time for each model

illustrates that at first, IL pushed the agent's policy towards the more optimal SAC policy from it's demonstrations, but failed to converge further. On the other hand, the PPO+GAIL(0.5) struggled in converging to higher rewards in earlier stages which may me due to getting stuck in local minima; However, the model maintained a constant incline which led to better convergence at the end of training.

As mentioned in Section III-E3, BC was further applied as a pre-training step to the first 5M steps, only on PPO. After training, an 8% decrease in cumulative reward was observed between the standalone PPO model and the PPO+GAIL+BC(0.1) model. Despite this, the model achieved higher rewards earlier in the training phase, both when compared to the standalone PPO and the PPO+GAIL(0.1), as shown in Figure 6d. This illustrates that BC improved the model at first, being able to reach higher rewards with few training steps, but failed in exploring more optimal policies. Additionally, a 15% increase in training time was required to train PPO with GAIL and BC when compared to the standalone PPO, which is not ideal when considering the model's outcome. This is especially true for the PPO+GAIL+BC(0.5) model which failed to converge optimally, possibly getting stuck on local minima.

1) Model Result Summary: As can be seen in the above results, in our continuous state and action space environment, all SAC outperformed PPO as shown in Figure 6e. Only minor improvements in earlier stages of the training phase, were made to the SAC model when combining this with the GAIL algorithm. In the case of applying IL techniques to PPO, a decrease in episode length was noted when GAIL was combined with PPO, especially with a weighting of 0.1. Similar to SAC, IL was provably useful in earlier stages of training when applied to PPO. This finding confirms the conclusion made by Hester et al. [44], in that no significant improvements can be obtained by applying IL to on-policy algorithms. This paper's findings show however, that earlier stages of training may benefit slightly from this. Regarding the use of BC as an extension to GAIL, worse performance results are obtained which is not ideal considering the increased amount of training time required.

B. Result Discussion - Agent Behaviour

As mentioned above, 5 identical agents were deployed on the same track and completed a total of 100 laps for each model. Table I illustrates; the average number of border collisions, both with the inner and outer border, the number of car collisions and the average race time, i.e. time to complete 5 laps. The model combining PPO, GAIL and BC with a 0.5 weight was not included since the agents were not able to complete the task at hand.

As can be seen in Table I, all the models avoid colliding with the inner and outer border successfully, with only sparse border collisions. When comparing the PPO and SAC agents, it is further confirmed that SAC is superior to PPO with a 10% decrease in race time. A considerable amount of car collisions are also produced by PPO agents. When visually evaluating how PPO agents race, these were observed to adopt a cooperative approach which requires agents to stick closer together thereby increasing the number of car collisions. Not only does this decrease the cumulative reward at the end, it also reduces the speed of the agents due to the frictional forces present. An explanation as to why this approach was learnt, may be related to policy's prioritisation of global reward which is achieved by reducing the risk of slipping in bends through creating a larger mass as seen in Figure 7.

Fig. 7: Cooperative PPO behavior among agents.

On the other hand, the SAC algorithm showcases more human-like behaviour where agents are observed to spread out evenly, also showing an element of overtaking as shown in Figure 8. Further human-like behaviour is showcased within Figure 9, which illustrates the agents mimicking professional racing lines performed in competitive human racing. Agents using SAC+GAIL models showed very similar behaviour to the standalone SAC model as confirmed by the results obtained in Table I. Few car collisions are obtained by these models since agents tend to drive safely after one another (Figure 10). Whereas SAC agents were more far apart to reduce the risk of such collisions, an interesting observation was made with IL boosted SAC models, which had agents stay much closer to

each other as shown in Figure 10. Unfortunately, this approach did not yield better lap times as with increased car collisions, momentum is decreased for the colliding agent.

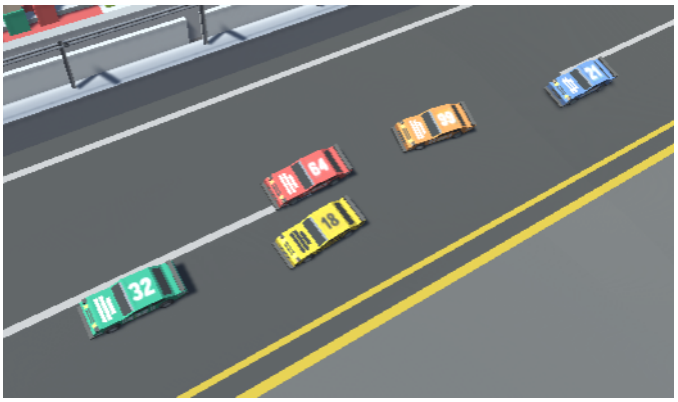


Fig. 8: SAC Human-like Behaviour

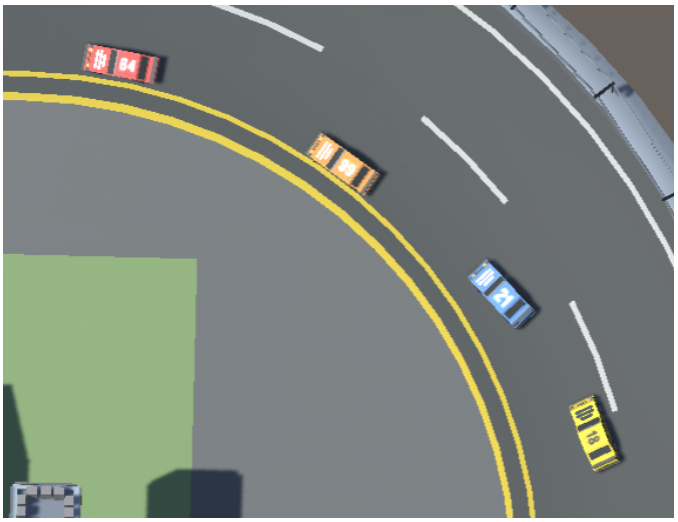


Fig. 9: Optimal path taken by SAC agents

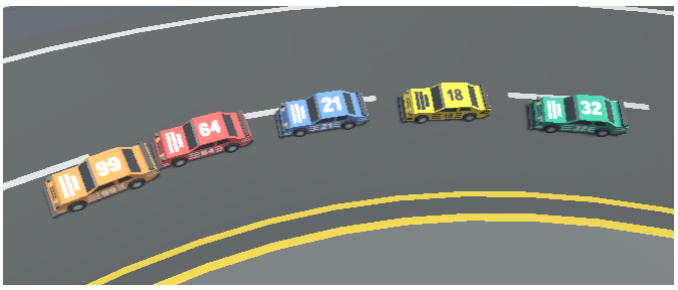


Fig. 10: Agents in close proximity

When evaluating the PPO+GAIL combined models, improvement is observed with respect to race time to the standalone PPO model as can be seen in Table I. The agents in these models are still observed to collaborate with each other to a lower degree. Unlike in PPO where all cars are merged together, in the IL boosted PPO models cars are observed to merge only two at a time. Additionally, these agents present better over taking behaviour similar to that observed in SAC. The worst performing model out of all models within Table I, was the PPO+GAIL+BC(0.1) model. The reason for this was that agents tend to get in each others way causing more crashes which the agents find difficult to recover from. Crashing between agents was also observed in other models but to a much lower degree. It is considered that these issues were further extrapolated to the 0.5 weighted version of this model which failed to converge to any usable policy.

V. Conclusion

This paper applies the on-policy PPO and off-policy SAC RL algorithms within a custom 3D racing environment to investigate which is superior for sensor-based racing. The effectiveness of using IL with such techniques is also investigated by applying GAIL along with GAIL+BC as a pre-training step. Moreover, the methodology involved in constructing a reward based environment is also detailed to inspire similar research in such domains. Through a quantitative and qualitative evaluation, SAC's superiority is determined which manages to achieve a higher cumulative reward and shorter lap times when compared to PPO. Additionally, more competitive human-like behaviour is observed in SAC agents when compared to the more cooperative approach utilised by PPO. IL yielded no significant improvements for both techniques at the end of the training phase. Instead, only slight benefits were observed in earlier stages of training.

Building from these results, the paper acknowledges the potential for future improvement. Firstly, more observations could be given to agents such as the current position of the agent and the position relative to the next checkpoint. Secondly, instead of using solely lap time and checkpoints as reward signals, the system may be extended by applying rewards based on car positions in the race and overtaking. To better evaluate the models, evaluation could be carried on more complex tracks to determine the generalisability of the techniques. The environment could also be enhanced to replicate real life physics further by including forces such as air resistance. Additionally, since only 3 hours worth of observations were recorded for Imitation Learning, more data could improve the technique's effectiveness when used this as a hybrid approach. Lastly, this paper motivates the application of other RL algorithms such as DDPG, to the domain of sensor-based racing. It may also be interesting to determine whether evolutionary algorithms such as Genetic Algorithms, are better at this task.

References

[1] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, "A Survey of Deep Reinforcement Learning in Video Games," *arXiv:1912.10944 [cs]*, Dec. 2019. arXiv: 1912.10944.

[2] J. Hegde and B. Rokseth, "Applications of machine learning methods for engineering risk assessment – A review," *Safety Science*, vol. 122, p. 104492, Feb. 2020.

[3] J. Wiens and E. S. Shenoy, "Machine Learning for Healthcare: On the Verge of a Major Shift in Healthcare Epidemiology," *Clinical Infectious Diseases*, vol. 66, pp. 149–153, Jan. 2018.

[4] N. Ravishankar and M. Vijayakumar, "Reinforcement Learning Algorithms: Survey and Classification," *Indian Journal of Science and Technology*, vol. 10, Jan. 2017.

[5] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation Learning: A Survey of Learning Methods," *ACM Computing Surveys*, vol. 50, pp. 21:1–21:35, Apr. 2017.

[6] J. Harmer, L. Gisslén, J. d. Val, H. Holst, J. Bergdahl, T. Olsson, K. Sjöö, and M. Nordin, "Imitation Learning with Concurrent Actions in 3D Games," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8, Aug. 2018. ISSN: 2325-4289.

[7] D. Li, D. Zhao, Q. Zhang, and Y. Chen, "Reinforcement Learning and Deep Learning Based Lateral Control for Autonomous Driving [Application Notes]," *IEEE Computational Intelligence Magazine*, vol. 14, pp. 83–98, May 2019. Conference Name: IEEE Computational Intelligence Magazine.

[8] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend, E. Calleja, S. Muralidhara, and D. Karuppasamy, "DeepRacer: Autonomous Racing Platform for Experimentation with Sim2Real Reinforcement Learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2746–2754, May 2020. ISSN: 2577-087X.

[9] Y. Zhu and D. Zhao, "Driving Control with Deep and Reinforcement Learning in The Open Racing Car Simulator," Lecture Notes in Computer Science, (Cham), pp. 326–334, Springer International Publishing, 2018. ISSN: 0302-9743.

[10] T. Jaakkola, M. L. Littman, C. Szepesvari, and S. Singh, "Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms," *Machine Learning*, vol. 39, no. 1998, pp. 287–308, 2000.

[11] C. Szepesvári, *Algorithms for reinforcement learning*, vol. 9. 2010.

[12] M. S. Holubar and M. A. Wiering, "Continuous-action reinforcement learning for playing racing games: Comparing SPG to PPO," *arXiv*, 2020.

[13] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend, E. Calleja, S. Muralidhara, and D. Karuppasamy, "DeepRacer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning," *arXiv*, pp. 2746–2754, 2019.

[14] A. Folkers, M. Rick, and C. Buskens, "Controlling an autonomous vehicle with deep reinforcement learning," *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2019-June, pp. 2025–2031, 2019.

[15] D. Loiacono, A. Prete, P. L. Lanzi, and L. Cardamone, "Learning to overtake in TORCS using simple reinforcement learning," *2010 IEEE World Congress on Computational Intelligence, WCCI 2010 - 2010 IEEE Congress on Evolutionary Computation, CEC 2010*, 2010.

[16] L. G. Cuenca, E. Puertas, J. F. Andrés, and N. Aliane, "Autonomous driving in roundabout maneuvers using reinforcement learning with q-learning," *Electronics (Switzerland)*, vol. 8, no. 12, 2019.

[17] X. Liu, Y. Liu, Y. Chen, and L. Hanzo, "Enhancing the Fuel-Economy of V2I-Assisted Autonomous Driving: A Reinforcement Learning Approach," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 8329–8342, 2020.

[18] W. Xia, H. Li, and B. Li, "A control strategy of autonomous vehicles based on deep reinforcement learning," *Proceedings - 2016 9th International Symposium on Computational Intelligence and Design, ISCID 2016*, vol. 2, pp. 198–201, 2016.

[19] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep Reinforcement Learning framework for Autonomous Driving," *arXiv*, pp. 70–76, apr 2017.

[20] S. Wang, D. Jia, and X. Weng, "Deep Reinforcement Learning for Autonomous Driving," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12109 LNAI, pp. 67–78, nov 2018.

[21] Z. Q. Huang, Z. W. Qu, J. Zhang, Y. X. Zhang, and R. Tian, "End-to-End Autonomous Driving Decision Based on Deep Reinforcement Learning," *Tien Tzu Hsueh Pao/Acta Electronica Sinica*, vol. 48, no. 9, pp. 1711–1719, 2020.

[22] P. Ke, Z. Yanxin, and Y. Chenkun, "A Decision-making Method for Self-driving Based on Deep Reinforcement Learning," *Journal of Physics: Conference Series*, vol. 1576, no. 1, 2020.

[23] X. Pan, Y. You, Z. Wang, and C. Lu, "Virtual to real reinforcement learning for autonomous driving," *arXiv*, 2017.

[24] V. Mnih, A. P. Badia, L. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, pp. 2850–2869, 2016.

[25] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot Programming by Demonstration," in *Handbook of robotics*, pp. 1371–1394, Jan. 2008. Journal Abbreviation: Handbook of robotics.

[26] S. Schaal, "Learning from demonstration," in *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS'96, (Cambridge, MA, USA), pp. 1040–1046, MIT Press, Dec. 1996.

[27] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan. 2016. Number: 7587 Publisher: Nature Publishing Group.

[28] S. Ross, G. J. Gordon, and J. A. Bagnell, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," *arXiv:1011.0686 [cs, stat]*, Mar. 2011. arXiv: 1011.0686.

[29] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, (New York, NY, USA), p. 1, Association for Computing Machinery, July 2004.

[30] B. Ziebart, A. Maas, J. Bagnell, and A. Dey, "Maximum Entropy Inverse Reinforcement Learning.," pp. 1433–1438, Jan. 2008.

[31] A. J. Snoswell, S. P. N. Singh, and N. Ye, "Revisiting Maximum Entropy Inverse Reinforcement Learning: New Perspectives and Algorithms," *arXiv:2012.00889 [cs]*, Dec. 2020. arXiv: 2012.00889.

[32] J. Ho and S. Ermon, "Generative Adversarial Imitation Learning," *arXiv:1606.03476 [cs]*, June 2016. arXiv: 1606.03476.

[33] D. A. Pomerleau, "Efficient Training of Artificial Neural Networks for Autonomous Navigation," *Neural Computation*, vol. 3, pp. 88–97, Mar. 1991. Conference Name: Neural Computation.

[34] S. Reddy, A. D. Dragan, and S. Levine, "SQIL: Imitation Learning via Reinforcement Learning with Sparse Rewards," *arXiv:1905.11108 [cs, stat]*, Sept. 2019. arXiv: 1905.11108.

[35] G. Li, M. Müller, V. Casser, N. Smith, D. L. Michels, and B. Ghanem, "OIL: Observational Imitation Learning," *arXiv:1803.01129 [cs]*, May 2019. arXiv: 1803.01129.

[36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv*, pp. 1–12, 2017.

[37] D. Q. Tran and S. H. Bae, "Proximal policy optimization through a deep reinforcement learning framework formultiple autonomous vehicles at a non-signalized intersection," *Applied Sciences (Switzerland)*, vol. 10, no. 16, 2020.

[38] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *35th International Conference on Machine Learning, ICML 2018*, vol. 5, pp. 2976–2989, 2018.

[39] A. Juliani, V. P. Berges, E. Vckay, Y. Gao, H. Henry, D. Lange, and M. Mattar, "Unity: A general platform for intelligent agents," *arXiv*, pp. 1–28, 2018.

[40] Eric Espié and Christophe Guionneau, "TORCS The Open Racing Car Simulator." Accessed on: Dec. 5, 2020. [Online]. Available: http://torcs.sourceforge.net/.

[41] "Unity Real-Time Development Platform — 3D, 2D VR & AR Engine.." Accessed on: Dec. 5, 2020. [Online]. Available: unity.com.

[42] Synty Studios, "Simple Racer." Accessed on: Dec. 5, 2020. [Online]. Available: https://assetstore.unity.com/packages/3d/vehicles/land/simple-racer-cartoon-assets-37490.

[43] "ProBuilder." Accessed on: Jan. 20, 2021. [Online]. Available: https://unity3d.com/unity/features/worldbuilding/probuilder.

[44] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Leibo, and A. Gruslys, "Learning from Demonstrations for Real World Reinforcement Learning," Apr. 2017.

[45] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Twenty-first international conference on Machine learning - ICML '04*, no. 346, (New York, New York, USA), p. 1, ACM Press, 2004.

APPENDIX

A. PPO Training parameters (Config file)

```
behaviors:
  CarAgent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 512
      num_layers: 3
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.995
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 15000000
    time_horizon: 1000
    summary_freq: 30000
    threaded: true

```

B. SAC Training parameters (Config file)

```
behaviors:
  CarAgent:
    trainer_type: sac
    hyperparameters:
      batch_size: 128
      buffer_size: 4000000
      learning_rate: 0.0003
      learning_rate_schedule: linear
      buffer_init_steps: 0
      tau: 0.005
      steps_per_update: 20.0
      save_replay_buffer: false
      init_entcoef: 0.5
      reward_signal_steps_per_update: 10.0
    network_settings:
      normalize: true
      hidden_units: 512
      num_layers: 3
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.995
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 15000000
    time_horizon: 1000
    summary_freq: 30000
    threaded: true

```

C. SAC+GAIL (0.1) Training parameters (Config file)

```
behaviors:
  CarAgent:
    trainer_type: sac
    hyperparameters:
      batch_size: 128
      buffer_size: 4000000
      learning_rate: 0.0003
      learning_rate_schedule: linear
      buffer_init_steps: 0
      tau: 0.005
      steps_per_update: 20.0
      save_replay_buffer: false
      init_entcoef: 0.5
      reward_signal_steps_per_update: 10.0
    network_settings:
      normalize: true
      hidden_units: 512
      num_layers: 3
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.995
        strength: 1.0
      gail:
        strength: 0.1
        demo_path: ../Imitations/SAC-Demonstrations.demo
    keep_checkpoints: 5
    max_steps: 15000000
    time_horizon: 1000
    summary_freq: 30000
    threaded: true

```

D. SAC+GAIL (0.5) Training parameters (Config file)

```
behaviors:
  CarAgent:
    trainer_type: sac
    hyperparameters:
      batch_size: 128
      buffer_size: 4000000
      learning_rate: 0.0003
      learning_rate_schedule: linear
      buffer_init_steps: 0
      tau: 0.005
      steps_per_update: 20.0
      save_replay_buffer: false
      init_entcoef: 0.5
      reward_signal_steps_per_update: 10.0
    network_settings:
      normalize: true
      hidden_units: 512
      num_layers: 3
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.995
        strength: 1.0
      gail:
        strength: 0.5
        demo_path: ../Imitations/SAC-Demonstrations.demo
    keep_checkpoints: 5
    max_steps: 15000000
    time_horizon: 1000
    summary_freq: 30000
    threaded: true

```

E. PPO+GAIL (0.1) Training parameters (Config file)

```
behaviors:
  CarAgent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 512
      num_layers: 3
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.995
        strength: 1.0
      gail:
        strength: 0.1
        demo_path: ../Imitations/SAC-Demonstrations.demo
    keep_checkpoints: 5
    max_steps: 15000000
    time_horizon: 1000
    summary_freq: 30000
    threaded: true

```

F. PPO+GAIL (0.5) Training parameters (Config file)

```
behaviors:
  CarAgent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 512
      num_layers: 3
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.995
        strength: 1.0
      gail:
        strength: 0.5
        demo_path: ../Imitations/SAC-Demonstrations.demo
    keep_checkpoints: 5
    max_steps: 15000000
    time_horizon: 1000
    summary_freq: 30000
    threaded: true

```

G. PPO+GAIL+BC (0.1) Training parameters (Config file)

```yaml
behaviors:
  CarAgent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 512
      num_layers: 3
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.995
        strength: 1.0
      gail:
        strength: 0.1
        demo_path: ../Imitations/SAC-Demonstrations.demo
    behavioral_cloning:
      steps: 5000000
      strength: 0.1
      demo_path: ../Imitations/SAC-Demonstrations.demo
    keep_checkpoints: 5
    max_steps: 15000000
    time_horizon: 1000
    summary_freq: 30000
    threaded: true


```

H. PPO+GAIL+BC (0.5) Training parameters (Config file)

```yaml
behaviors:
  CarAgent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 512
      num_layers: 3
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.995
        strength: 1.0
      gail:
        strength: 0.5
        demo_path: ../Imitations/SAC-Demonstrations.demo
    behavioral_cloning:
      steps: 5000000
      strength: 0.5
      demo_path: ../Imitations/SAC-Demonstrations.demo
    keep_checkpoints: 5
    max_steps: 15000000
    time_horizon: 1000
    summary_freq: 30000
    threaded: true

```