Autonomous Driving via Deep Reinforcement Learning

Roy Amante Salvador, Maria Isabel Saludares
roy.salvador@gmail.com,mis.saludares@gmail.com
Department of Computer Science
University of the Philippines - Diliman
Quezon City, Philippines

ABSTRACT

In this paper, a streamlined working pipeline for an end-to-end deep reinforcement learning framework for autonomous driving was introduced. It integrates the usage of a choice combination of Algorithm-Policy for training the simulator by streamlining the integration of Microsoft AirSim, OpenGym, and Stable-Baselines. The pipeline is tested for goal reaching (represented by waypoints) with penalized collision. Several algorithm-policy combinations were tested and performance was measured using success rate (reaching the goal), number of collisions, and goal distance. Several of the models reached 30% to 100% success rates over 100 test episodes demonstrating the learning of autonomous goal reaching and collision avoidance.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

autonomous driving, deep reinforcement learning, actor critic method

ACM Reference Format:
Roy Amante Salvador, Maria Isabel Saludares. 2019. Autonomous Driving via Deep Reinforcement Learning. In *PCNaval '19: CS 295 Reinforcement Learning*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A car that drives autonomously is a long-standing goal of artificial intelligence with millions of kilometers being driven as of today to test its capability and feasibility. Google's AVs (Waymo) have reached the mark of more than 5 million kilometers driven on American avenues, streets, and roads [11], while Uber's AVs have also reached over 1.5 million kilometers driving through Pittsburgh, Phoenix, and Toronto [4].

Driving requires high-level skill, attention, and experience by a human driver [10]. Computers are capable of long and sustained attention (given a condition), however it requires the intelligence and skill of a driver which is acquired by experience. Sallab, et. al [10] summarized that the tasks involved in creating an autonomous driving agent is subdivided into three categories: recognition, prediction, and planning.

In their framework, recognition is composed of the aggregation of the sensor information in order to sense the surrounding environment. Recognition is a relatively easy task with the advances in deep learning algorithms particularly in object detection, classification, and segmentation problems [1, 3, 7].

. Prediction on the other hand

Figure [] is a view of the driving agent from the simulator where the car tries to navigate to a way point in front of it, represented by the second car. The framework used for training uses several combinations of algorithm and policy were tested in an open source 3D simulator AirSim. The simulation results demonstrate learning of autonomous maneuvering in a scenario of a road with possible obstacles, with an intent to reach a designated way point.

Our contributions are:

- create a wrapper for driving simulator (AirSim) with OpenAI Gym
- explore policy composition problem with PPO, A2C, SAC with MLP, MLP LSTM, MLP with layer normalization and Meta-Learner in a driving simulator
- explore the effects of changing the ranges of movement of the actions in training policies for autonomous driving
- explore reward composition in achieving to a more stable policy
- experimental results suggest that the

2 AUTONOMOUS DRIVING

Driving is a multi-agent interaction problem as it consist of agents and their environment. As a human driver, it is easier to keep driving within the lane without any other car changing lanes in a traffic, and the latter is more difficult because of the inherent uncertainty in the behaviour of other drivers. Similarly, trying to navigate and reach a destination in an unknown city is harder than navigating through your usual routes. The number of vehicles, their size and nature, road infrastructure, and the behavior of the drivers largely contribute to the variability of the condition for which you have to drive and it is difficult to design a supervised system which can potentially cover all possible scenarios. Humans on the other hand employ an online reinforcement learning to understand the behavior of other drivers and behave accordingly. And this is reapplicable in cases which needs negotiation, for example entering a roundabout, moving past certain blockages along the way, adjusting speed in extreme weather conditions, navigating junctions without traffic lights, non-existent road lanes, etc.

PCNaval '19, 17 May 2019, Quezon City, Metro Manila
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

According to a review by Rosenzweig et.al, the three biggest obstacles to reach the mass adoption of driverless cars are [9]:

- legal liability,
- policymakers and customer acceptance, while the following three;
- cost, infrastructure and technology are seen as less of a problem.

Near to 91% of the research as of 2015 focused on the technology development and the existing research gap on the user acceptance of the technology while only 1.3% is on user acceptance of technology, to which it is also important to note that all 5 publications found were released in the last 5 years [9].

Majority of the works and studies in bringing this technology to mainstream has mostly focused on advancing the technology such as improving sensors to meet the demands of the expanded autonomous vehicle operations (radar, lidar, camera) [cite], enabling real-time route optimization through communications between cards and traffic management infrastructure [cite], reduction of energy consumption [cite], and increase safety for passengers and bystanders by reducing the risk of accidents through safe driving [cite]. All of which are geared towards enabling the vehicles to operate at five increasing autonomous levels set by SAEJ [cite].

The main challenge in autonomous driving is dealing with *edge* cases which are unexpected even for most human drivers, like dealing with flooding or navigating an area without a GPS. The RL paradigm then learns from its own experience by taking actions in unchartered scenarios.

The current standard approach for autonomous driving is to decouple tasks into isolated sub-problems, typically supervised learning-like object detection, visual odometry, etc, and then having a post processing layer to combine all the results of the previous step. However with this approach, the sub-problems which may need to be solved is more difficult than autonomous driving. And the isolated sub-problems may not combine coherently to achieve the goal of driving. In reinforcement learning, this is dealt by a reward signal corresponding to good driving which can model the interplay between driving (taking action) and planning (where to drive). As the reward is based on stable driving, not crashing into anything, and reaching a specific place, it is challenging to train an RL System with a real car with the risks involved. Therefore most of the current researches is done using simulation engines to mimic conditions in real cars, with the flexibility to create and iterate scenarios which is relevant for learning.

3 AUTONOMOUS DRIVING USING REINFORCEMENT LEARNING

Sallab et.al introduced a pipeline framework for an end-to-end training for autonomous driving where the input are the states of the environment and their aggregations over time, and the output are the driving actions [10]. In the spatial aggregation, sensor information is fused and in this representation, importance is emphasized into parts of the data that has more relevance to the task such as the usage of attention and glimpse networks [cite]. The combination of the attention glimpse network and convolutional neural network in building the spatial features leverages on deploying the gaze of the kernel to certain parts of the data, thereby reducing computations dramatically in inference time as well as learn which areas are more relevant to sense to perform the task. This is consistent with the addressing the first issue of decoupling sub-problems where the segmentation of the full image of the scene may not be necessarily to adequately perform the driving task. After which, the pipeline continues into the reinforcement learning planning plan where the network follows the same training procedure of the DQN (discrete actions) and DDAC (continuous actions).

Kyushik Min [8], an autonomous car engineer created a vehicle simulator with Unity-ML Agents that features the use of different range sensors as part of the vehicle's Advanced Driver Assistant Systems (ADAS). A medium ranged sensor serves as a forward warning device which controls the speed of the host vehicle. It rapidly drops the velocity whenever there's a car in front with small distance. A short range sensor acts as side warning which prevents cars to change lanes if there are close by. Long range sensor (LIDAR), helps the vehicle stay at the center of the lane and avoiding obstacles. The best performing setup have these sensor information fed to an LSTM combined with forward camera image encoded by a convolutional neural network as observation vector. The vector becomes input to dueling networks representing separate approximation of the state value function and the state-dependent action advantage function. This then is aggregated to get the Q-value for action selection. Action space includes doing nothing, acceleration, slowing down, changing left and right lane.

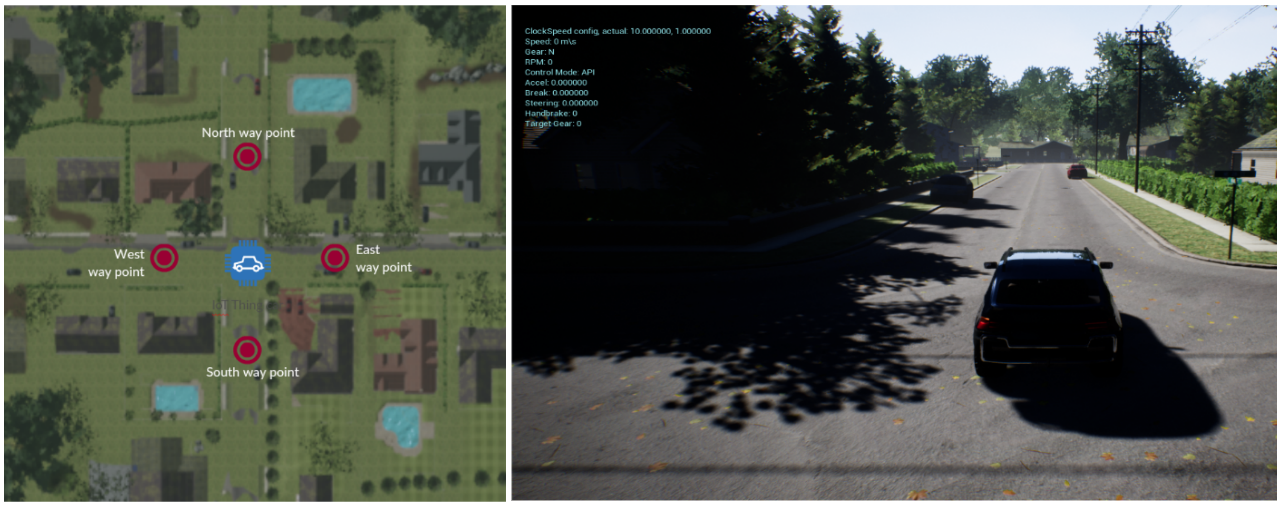
4 ENVIRONMENT SIMULATORS

One of the challenges identified earlier is the training of an RL system with real car. In order to facilitate the training, most researches is done using simulation engines in order to mimic the environment and car conditions that are expected (with the help of the inherent physics engine) and create endless cases and scenarios for which the car can train on.

Advances in computer vision and deep learning methods that integrate learning-based vision and control require massive amounts of domain-specific visual data which can be easily solved by the usage of the simulated learning environments. And to bridge the gap in the transfer policies that are learned in simulators to real scenes and enable domain-independent policy learning without real images [cite], [NAME] introduced a work that optimizes the augmentation technique for sim2real transfer. The method applies an efficient search for depth image augmentations and the resulting random transformations sequence is then used to augment synthetic depth images during policy learning. Another work adapted deep visuomotor representations from simulated to real environments [**?**].

4.1 AirSim

For car driving task, AirSim is an open source simulator for both drones and cars developed by Microsoft AI and Research Group [12]. To enable learning, the simulator has capabilities that support the creation of scenarios designed by the user, be it variation in the scene (city, mountains, indoor, etc.), weather effects, time of day, traffic adherence (city environment can track traffic violations), additional sensor attachment, and other. It also has a ROS wrapper for multirotors which allows the mirroring of the actual robot (car,



```
ClockSpeed config, actual: 10.000000, 1.000000
Speed: 0 m\s
Gear: N
RPM: 0
Control Mode: API
Accel: 0.000000
Break: 0.000000
Steering: 0.000000
Handbrake: 0
Target Gear: 0
```

Figure 1: (left) Top view of the AirSim environment with the corresponding waypoint positions, and (right) view of the simulator behind the car facing the North way point.

drone, underwater vehicle) into the simulator and deployment of the developed algorithm directly back to the robot. Its APIs allow interaction with the vehicles in the simulation programmatically, which is used for research. The full list of capabilities of the simulator is provided in [link].

5 METHODOLOGY

The implementation is divided into three (3) parts: (a) creation of OpenAI Gym Wrapper for AirSim, (b) building environments, and (c) training and testing. The following tools were used:

- Microsoft AirSim simulator for the environment [12]
- OpenAI Gym [2]
- Stable-baselines [6]

Microsoft AirSim [12] simulator was used as the environment for training. The goal is to reach preset waypoints as indicated in Figure 1. The car agent is initially positioned at the center.

Action and Observation Spaces

The action state (driving agent) obtained are:

- steering: angle of steering
- throttle: accelerator
- break: soft break (>0) to hard break (1)
- gear: distribution function from -1 to 1

Steering refers to the angle of steering and controls the direction of the movement of the car. Throttle refers to the usage of the accelerator and controls the acceleration or deceleration of the car. Break also contributes to the deceleration and stopping of the car. The gear is assigned base on using the distribution function in Equation1.

$$A_{\text{gear}} = \begin{cases} \text{drive,} & \text{if } x \geq 0.3 \\ \text{neutral,} & \text{if } -0.3 \geq x \leq 0.3 \\ \text{reverse,} & \text{if } x \leq 0.3 \end{cases} \tag{1}$$

The observation space is composed of the following:

- linear and angular velocity of the car
- Light Detection And Ranging (LIDAR) reading per 1Âř
- orientation (yaw)

The LIDAR is set to read per 1deg, with a 10m range, 10 rotations per second, with 10,000 points per second.

Reward and Penalties

The rewards set for the reinforcement algorithm systems is given by Equation 2.

$$r_{\text{total}} = r_{\text{dist}} + r_{\text{orientation}} + r_{\text{collision}} + r_{\text{reverse}} + r_{\text{movement}} \tag{2}$$

The positive rewards are euclidean distance from the waypoint and orientation of the car with respect to waypoint as shown in Equations 3 and 5, where $d_{\text{car, target}}$, $d_{\text{goal, max}}$, and $d_{\text{goal, th}}$ are defined in Equations 4.

$$r_{\text{dist}} = w_{r_{\text{dist}}} * \frac{\mathrm{d}_{\text{goal, max}} - \text{dist}}{\mathrm{d}_{\text{goal, max}} - \mathrm{d}_{\text{goal, th}}} \tag{3}$$

$$d_{\text{car, target}} = \sqrt{\sum_{i=1}^{n} (S_{\text{car}, i} - S_{\text{target}, i})^2} \tag{4a}$$

$$d_{\text{goal, max}} \in \{30, \dots, 40\} \tag{4b}$$

$$d_{\text{goal, th}} \in \{2, \dots, 5\} \tag{4c}$$

$$r_{\text{orientation}} = \cos(S_{\text{car}_{yaw}} - S_{\text{target}_{yaw}}) \tag{5}$$

The negative rewards (penalties) are collisions detected, reverse, and lack of movement and are computed based on Equations 6 - 8.

$$r_{\text{collision}} = -w_{\text{collision}} * count_{\text{collision}} \tag{6}$$

$$r_{\text{reverse}} = -w_{\text{reverse}} * 1 \tag{7}$$

$$r_{\text{movement}} = \begin{cases} -w_{\text{car}_v} & \text{, if } S_{\text{car}_v} < S_{\text{car}_{v,th}} \\ 0 & \text{, otherwise} \end{cases} \tag{8}$$

	Algorithm-Policy	total reward (mean)	collisions (mean)
1	***A2C-MLP_Full***	Steering, throttle, break, reverse	
2	***A2C-MLP_Full_1***	Steering, throttle, break, reverse (vary reward weight)	Euclidean Distance
3	***A2C-MLP_Partial***	Steering, throttle, break	Collision
4	***A2C-MLP_Partial_1***	Steering, throttle, break (vary reward weight)	Reverse
5	***A2C-MLPLSTM_Full***	Steering, throttle, break, reverse	Lack of Movement Speed
6	***A2C-MLPLSTM_Partial***	Steering, throttle, break	
7	***PPO2-MLP***	Steering, movement (Throttle, Brake, Reverse)	Orientation
8	***PPO2-MLPLSTM***	Steering, movement (Throttle, Brake, Reverse)	Euclidean Distance
9	***PPO2-MetaRL***	Steering, movement (Throttle, Brake, Reverse)	Collision
			Reverse
10	***SAC-LnMLP***	Steering, movement (Throttle, Brake, Reverse)	Lack of Movement Speed

Table 1: Summary of experiments: algorithm-policy, environment, and rewards.

The goal for the driving agent is to reach set waypoints, selected randomly, in sequence. The initial condition is the car is spawned at the middle of the map and the four (4) waypoints are indicated by the red dots labeled North, South, East and West waypoints. The episode ends when all waypoints are reached or time runs out (which is set at 60 timesteps).

Reinforcement learning agents

The algorithms used are advantage actor critic (A2C), proximal policy optimization (PPO2), and soft actor critic (SAC). In A2C, the actor controls the behavior of the agent while the critic measured the value of the action taken and learns the advantage function instead of the Q value functions. Proximal policy optimization on the other hand is also an actor critic method by limiting how far the policy can change for each iteration and adds a soft constraint to the objective function. SAC is an off-policy maximum entropy deep reinforcement learning algorithm that has a stochastic actor to introduce robustness. The policies and their descriptions are:

- MLP: 2 layers of 64 neurons
- MLP-LSTM: 2 layers of 64 neurons followed by LSTM layer with 256 cells
- Deep Meta-RL: 2 layers of 64 neurons followed by LSTM layer with 256 cells, reward and action from previous timestep as input [13]
- Custom MLP: 3 normalized layer of 512 neurons

These are available and were implemented via Stable Baselines [6].

5.1 Training and Testing

In terms of simulation setup, the input to the network is the aggregation of some or all components of the observation space: car state, LIDAR (10k points), and yaw. The output are the steering, gear, throttle, and break. Algorithm-Policy combinations were trained end-to-end following the objective of reaching a waypoint within the timeframe. The different experiments performed are summarized in Table 1. The training was set to end after one million (1M) timesteps with 60 timesteps for each episode. And the testing is composed of 100 episodes with 120 time steps for each episode.

The computers used to run the simulations and their estimated runtimes for the episodes were:

- i7, 16GB RAM, NVIDIA GTX 970, 17 hours / 300k timesteps
- NVIDIA GTX 1060, 12 hours / 400k timesteps
- Google Compute n1-standard-4 (4 vCPUs, 15 GB memory) P4 workstation, 5 hours / 100k timesteps

It is interesting to note that after the agent selects the action we allow the environment to react and perform the necessary simulation for a time interval inverse to the set clock speed of the simulator before deciding on the next action. The time interval adds to the wall clock duration of a time step and ultimately adds to the length of training time. Setting the clock speed to a higher number degrades the quality of simulation by missing detection of collisions for example.

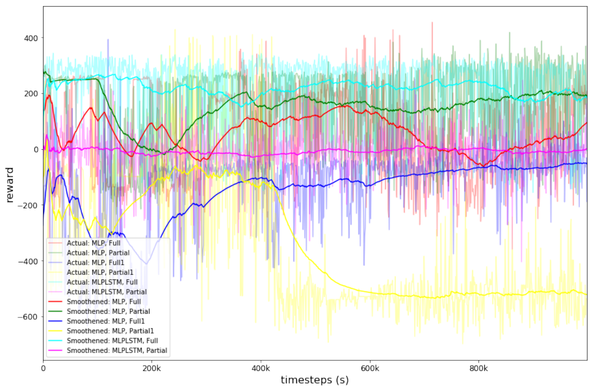
5.2 Performance Metrics

To evaluate the performance of the models, success rate, average timestep, number of collisions, and trajectory distances were measured. Success is defined as reaching a goal distance of 10m or below and the success rate is measured by the proportion of episodes that succeeded over all the episodes run. Average timesteps measures how long it takes to complete an episode and lower values indicate that the car reaches its goal faster. Trajectory distance measure how far the car has traveled, and lower values may mean that the agent is not moving much or is always in collision.

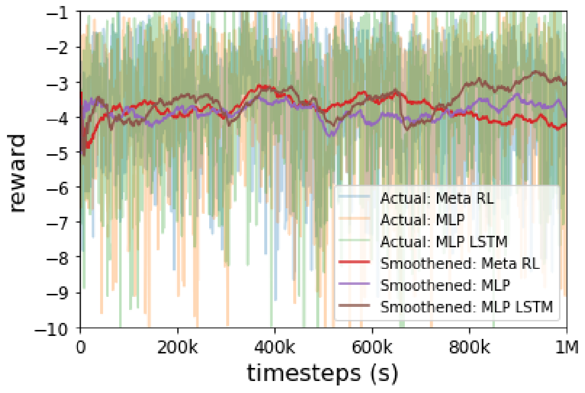
5.3 Results and Discussion

Rewards and entropy losses on all the experiments are shown in Figure 2 and Figure 3 respectively. It is observed that for A2C, the MLP and MLP+LSTM policies that uses the full set of actions has positive values reward values compared to the rest of the experiments. The observed increasing trends in rewards are in A2C-MLP and SAC. For SAC trained only at 500k timesteps, there is a discernable increase of reward which can be attributed to it being more sample efficient than the other algorithms. The presence of a stochastic actor forces more exploration which has contributed to a faster convergence to a policy.

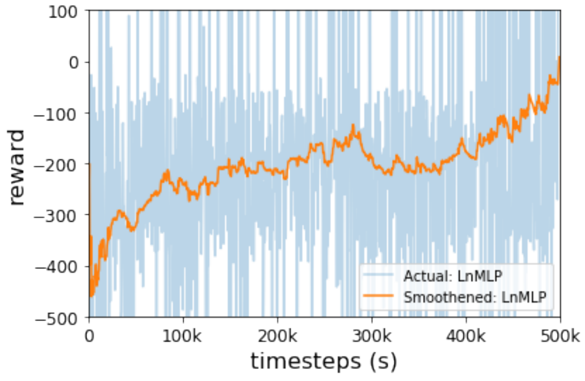
Entropy loss on the other hand is a measure on how random a set of actions is chosen. The higher the number, the more random the



(a) A2C

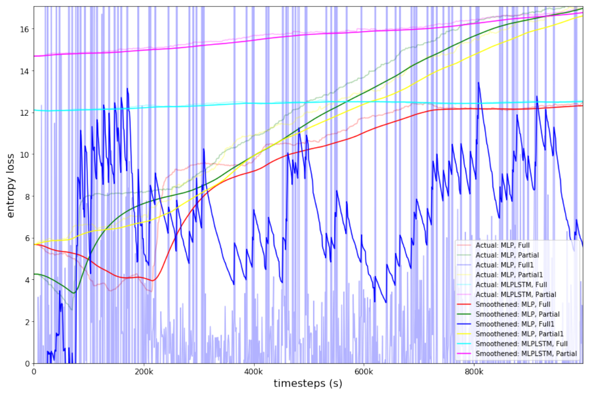


(b) PPO2

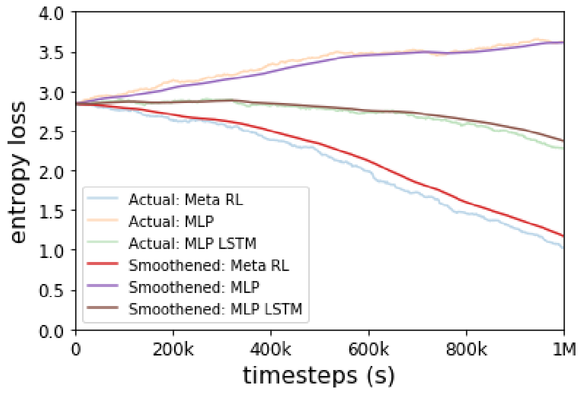


(c) SAC

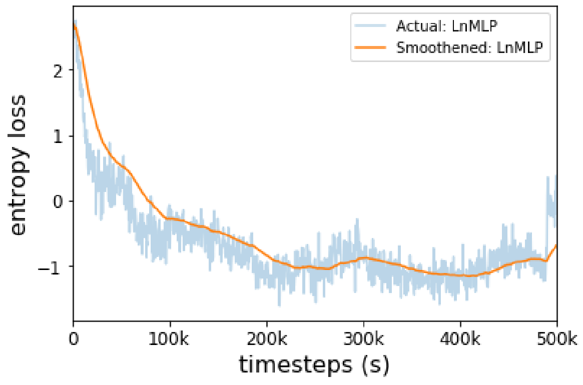
Figure 2: A2C, PPO2, and SAC rewards for 1M timesteps.



(a) A2C



(b) PPO2



(c) SAC

Figure 3: A2C, PPO2, and SAC Entropy Losses for 1M timesteps.

driving agent selects, and lower value indicate more exploitation and exploration. Again, for SAC, a steep decline to negative values in 100k timesteps despite its stochastic nature is observed. The exploration performed at the early stages of training helped the policy to converge faster. It also learns relatively faster compared to A2C and PPO2 which is consistent with a previous work that has demonstrated that SAC beats PPO in robotic tasks [5].

For policy networks, MLP policies for both A2C and PPO has an increasing trend after 1M timesteps while LSTM is relatively flat and slowly declining. In Meta RL which leverages on the previous reward and action to a recurrent policy network, the rate of decrease is higher suggesting faster convergence to some policy.

The trajectories of the car agent during testing are shown in Figure 4. The color of the points indicate at which point in time

	Algorithm-Policy	success rate	timesteps (mean)	total reward (mean)	collisions (mean)	trajectory distance (mean)	goal distance (mean)	goal distance (std)
0	***A2C-MLP_Full***	7%	117.84	408.16	22.64	21.17	19.14	4.95
1	***A2C-MLP_Full_1***	0%	120	-323.08	18.17	10.41	25.11	2.84
2	***A2C-MLPLSTM_Full***	0%	120	432.78	22.90	30.93	32.40	2.82
3	***A2C-MLP_Partial***	15%	112.48	192.32	52.12	21.68	18.92	7.42
4	***A2C-MLP_Partial_1***	0%	120	-1851.52	86.50	33.89	40.93	5.63
5	***A2C-MLPLSTM_Partial***	30%	104.98	239.40	32.36	28.29	13.15	3.67
6	***PPO2-MLP***	0%	120	-2.46	12.42	47.0	32.29	5.56
7	***PPO2-MLPLSTM***	0%	120	-3.12	11.44	34.29	44.69	10.60
8	***PPO2-MetaRL***	32%	105.73	-4.05	17.80	33.16	22.83	12.49
9	***SAC-LnMLP***	100%	28.67	938.42	0.01	23.21	9.46	0.40

Table 2: Performance evaluation during testing at 100 episodes.

during the 120 timestep episode while x and y values refer to the relative position of the car with respect to its initial position. Target car (point) is represented by the blue circle. The magenta and orange circles are a 10m and 12m perimeter around the target and is used as reference to identify if the car has succeeded in reaching the goal (magenta circle, <= 10m away from target). Based on the plots, A2C-MLP (Full), A2C-MLP (Partial), A2C-MLP+LSTM (Partial), PPO2-MLP, and SAC-LnMLP models yielded to several episodes succeeding in reaching the goal. There is also an observed circular movement of the car (in an attempt to move and explore the environment) for some of the models.

Overall performance of these trajectories as evaluated using success rate, average rewards, average timesteps, number of collisions, and trajectory distances during testing are summarized in Table 2. Straighter plots to the goal is observed for SAC which also showed a 100% success rate with minimal collisions and low values of average goal distance. The next best performing models are PPO2-MetaRL, A2C-MLP+LSTM (Partial), A2C-MLP (Partial), A2C-MLP (Full) with corresponding success rates of 32%, 30%, 15%, and 7% respectively. In terms of collisions, SAC-LnMLP, PPO2-MLP+LSTM, PPO2-MLP, PPO2-MetaRL, and A2C-MLP (Full 1) has the lowest average collisions over episodes. However only SAC-LnMLP and PPO2-MetaRL has a positive success rates while the rest is 0%. The policies trained may be focused mainly on avoiding collision, as opposed to reaching its goal which is reflected by their high values of goal distances, one of which reach to as far as 44.69m (PPO2-MLP+LSTM) which is much further from its initial position relative to the target. Based on the results, the best performing model is from using SAC-LnMLP which resulted with a 100% sucess rate, positive average total reward and a 9.46m average goal distance.

6 CONCLUSION AND RECOMMENDATION

In this paper, a framework for an end-to-end Deep Reinforcement Learning pipeline for autonomous driving which easily integrates the usage of several Algorithm-Policy combinations for training the simulator. Integration of Microsoft AirSim, OpenGym, and Stable-Baselines was implemented for streamlined implementation and experimentation. The framework was tested for goal reaching (with penalized collision) and test trajectory paths with successful learning are shown. The best performing model is from using SAC-LnMLP which resulted with a 100% success rate, positive average total reward and a 9.46m average goal distance. Future work includes continuing the training from the 1M timesteps (which is easily supported by the usage of Stable-Baselines), deploying the framework using RGB-D information (as opposed to LIDAR), other reward/penalty functions, additional obstructions. The stream-lined framework can lead to potential extension of the same framework to other robotic tasks and autonomous driving (underwater vehicles, drones).

REFERENCES

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. 2017. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence* 39, 12 (2017), 2481–2495.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[3] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. 2016. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3213–3223.

[4] Rodrigo Marçal Gandia, Fabio Antonialli, Bruna Habib Cavazza, Arthur Miranda Neto, Danilo Alves de Lima, Joel Yutaka Sugano, Isabelle Nicolai, and Andre Luiz Zambalde. 2019. Autonomous vehicles: scientometric and bibliometric review. *Transport reviews* 39, 1 (2019), 9–28.

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *CoRR* abs/1801.01290 (2018). arXiv:1801.01290 http://arxiv.org/abs/1801.01290

[6] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. 2018. Stable Baselines. https://github.com/hill-a/stable-baselines.

[7] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 740–755.

[8] K. Min and H. Kim. 2018. Deep Q Learning Based High Level Driving Policy Determination. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. 226–231. https://doi.org/10.1109/IVS.2018.8500645

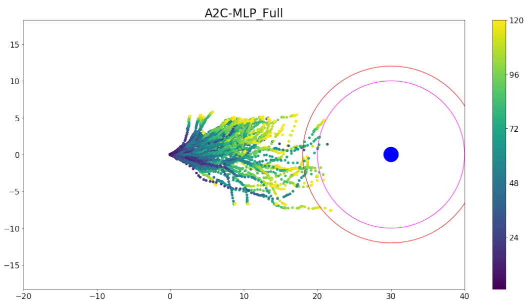
[9] Juan Rosenzweig and Michael Bartl. 2015. A review and analysis of literature on autonomous driving. *E-Journal Making-of Innovation* (2015).

[10] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. 2017. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging* 2017, 19 (2017), 70–76.

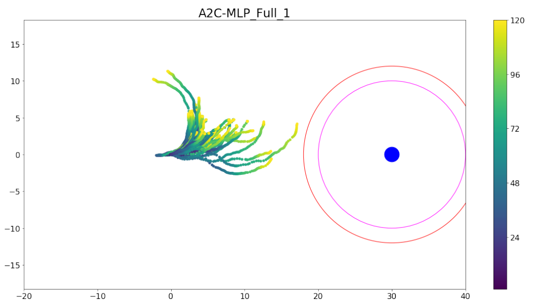
[11] Paul Sawyers. 2018. Waymo clocks 10 million self-driven miles on public roads, double in 8 months. https://venturebeat.com/2018/10/10/waymo-clocks-10-million-self-driven-miles-on-public-roads-double-in-8-months/ Accessed: 2019-05-07.

[12] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2017. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*. arXiv:arXiv:1705.05065 https://arxiv.org/abs/1705.05065

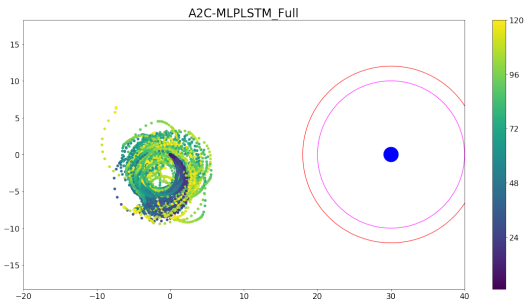
[13] Jane X. Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Rémi Munos, Charles Blundell, Dharshan Kumaran, and Matthew Botvinick. 2016. Learning to reinforcement learn. *CoRR* abs/1611.05763 (2016). arXiv:1611.05763 http://arxiv.org/abs/1611.05763



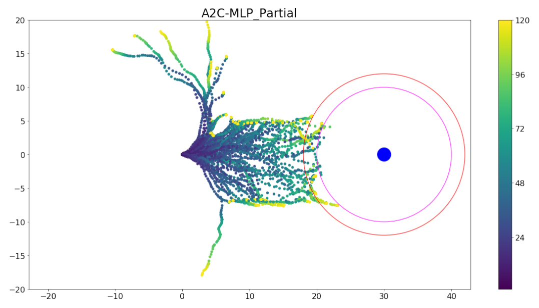
(a) A



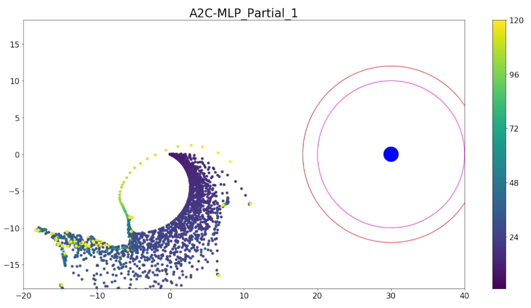
(b) A



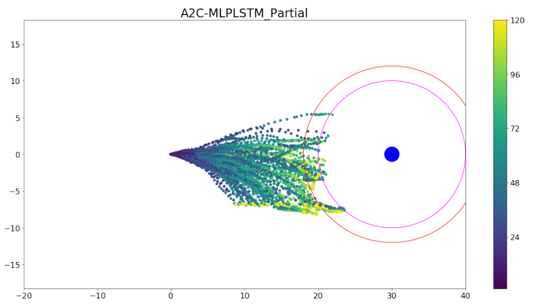
(c) A



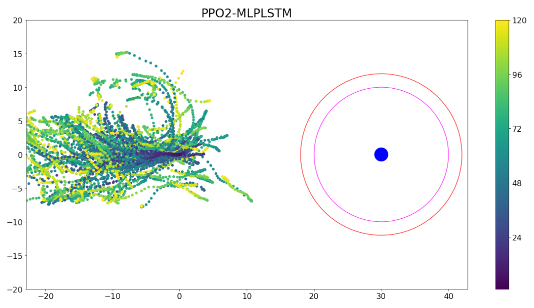
(d)



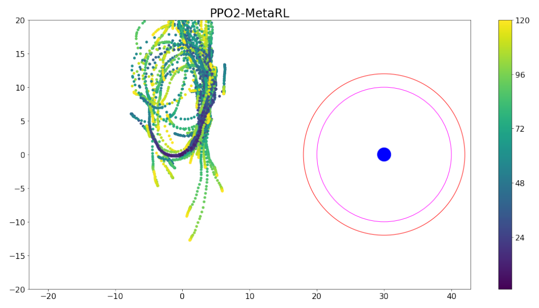
(e)



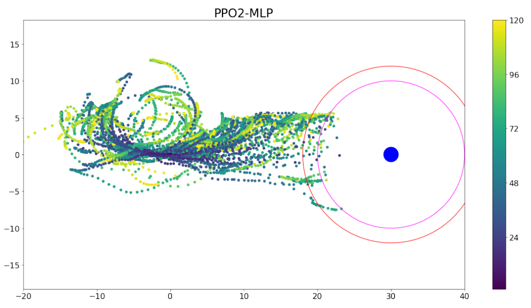
(f)



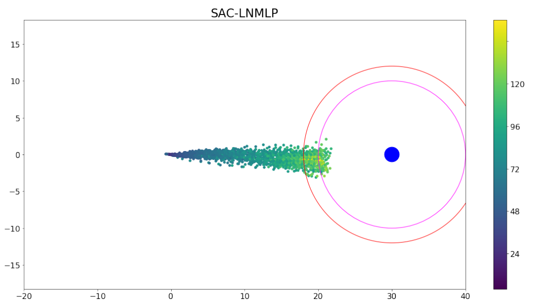
(g)



(h)



(i)



(j)

Figure 4: Test trajectories for each of the experiment where the green circle indicate a 12m radius from target, and orange circle is 10m away.