2021 Latin American Robotics Symposium (LARS), 2021 Brazilian Symposium on Robotics (SBR), and 2021 Workshop on Robotics in Education (WRE) | 978-1-6654-0761-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/LARS/SBR/WRE54079.2021.9605436

Addressing Lane Keeping and Intersections using Deep Conditional Reinforcement Learning

Vítor A. S. Silva[1] and Valdir Grassi Jr[1]

***Abstract*— End-to-end deep reinforcement learning (DRL) methods have been largely used to solve self-driving tasks. However, they usually deal only with simple problems, such as lane keeping. When it is needed to solve more complex tasks, traditional DRL algorithms fails. In this work we combine Conditional Learning and Proximal Policy Optimization (PPO) to solve the problem of turning at intersection and lane keeping in an end-to-end fashion. In our approach we trained three PPO sub-policies to perform right and left turns and follow lane. The sub-policies are activated once at a time according to a command sent by the local planner. We also used three different image transformation to verify their impact on learning speed and generalization. Experiments were conducted in an urban scenario composed by several intersections of type T on CARLA Simulator. Results show that our approach is feasible and presents good performance on accomplishing the goal. Moreover, we could confirm that properly choosing an image transformation can improve sample efficiency and generalization capability.**

I. Introduction

Autonomous vehicles, when they are well developed and used on a large scale, will have the potential to provide greater traffic safety, solving the main causes of accidents [1]. In addition, this technology would afford greater mobility for elderly and people with physical and visual disability, greater efficiency on energy consumption, and increase productivity of time spent on traffic [1], [2].

Researches on autonomous driving rely on two main approaches to solve self-driving tasks. The first, referred as mediated approach, uses separated and specialized modules that communicate in a pipeline, from sensory to the actuators control [1]. The common modules used in this approach are: perception, localization and mapping, planning and decision making, and control [1]–[4]. The second approach is named end-to-end, in which the control commands of actuators are mapped directly from the observations, commonly relying on machine learning techniques [2], [5], being deep reinforcement learning (DRL) and imitation learning (IL) the two most employed [3].

With success in solving games with super-human results, reinforcement learning proved to be a viable approach to address self-driving tasks. Kendall *et al.* [6] were the first to use RL in this context, making a vehicle to drive following a lane only with aid of a monocular camera. Posteriorly, many other works using a RL method in an end-to-end fashion were done, such as [3], [7], [8], and others.

Among the reasons that one can choose an end-to-end approach over a mediated to solve navigation tasks are that the latter: (i) lacks on generalization for new scenarios and tasks, (ii) is prone to error propagation through pipeline, (iii) is more complex to design, and (iv) can produce conservative driving policies due to human heuristics, since it is hand-engineered [3].

However, end-to-end frameworks also have some drawbacks. Due to the fact that this approach relies mainly on the use of deep neural networks, it is not possible to explain how the policy learned works (lack of interpretability) [3], [9]. Another shortcoming is that it presents good performance only for simple tasks, such as lane keeping. More complex tasks, such as deciding where to go at an intersection, are not possible to solve with just the information from the camera (ambiguity). Without driver's intent information, even though the agent could handle the intersection, its decision could not be the best one, or, in the worse case, the car could take the same decision every time it encounters an intersection [3], [5], [9].

To solve the ambiguity problem, Codevilla *et al.* [5] introduced the idea of Conditional Imitation Learning (CIL). In their work, they trained three distinct neural networks, each one being responsible for one of the possible maneuvers: turn left, turn right and go straight. These networks are activated once at a time according to a high-level command from the route planner. This approach is feasible and effective because, once the agent has the information of driver's intention, the planner command select the correct specialized sub-policy to act at a given moment. Müller *et al.* [10] used this concept to produce the next waypoints instead of steering and throttle controls. Given the waypoints indicating the desired path, a PID controller generates the low-level controls. This idea can be also used combined with DRL, as done in [9] and [11].

In this work, we propose a data-efficient end-to-end self-driving method based on RL algorithm Proximal Policy Optimization (PPO) that aims to address the problem of ambiguity when facing intersection of type T in a simulated urban environment by introducing concepts of Conditional Learning and Augmentation Data. The aspects that differ our work from the ones cited above are:

- Lateral and longitudinal controls generated directly

*This work was financed in part by the Brazilian National Council for Scientific and Technological Development - CNPq (grant 465755/2014-3), by the Coordination of Improvement of Higher Education Personnel – Brazil - CAPES (Finance Code 001 and 88887.136349/2017-00), and the São Paulo Research Foundation - FAPESP, Brazil (grant 2014/50851-0).

[1]The authors are with the Department of Electrical and Computer Engineering, São Carlos School of Engineering, University of São Paulo, São Carlos, Brazil. vitor.augustoss@usp.br; vgrassi@usp.br.

from PPO sub-policies;
- Online learning process (no expert demonstrations or prior training needed); and
- Sample efficiency and generalization through image augmentation usage.

This paper is organized as follows: Section II presents the foundation of PPO and augmentation data applied to it; Section III presents how PPO, Conditional Learning and augmentation data were combined in the proposed method; Section IV brings the results and discussion about them; and in Section V final considerations and thoughts for future works are exposed.

II. Proximal Policy Optimization

PPO is a model-free policy gradient (PG) DRL algorithm proposed in [12]. Algorithms based on PG theory optimizes a policy π_θ updating the deep neural networks (DNNs) parameters θ through stochastic gradient ascent algorithm so the policy produces maximum expected cumulative reward. In this kind of approach, gradient estimator is often computed by

$$\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_\theta \, log \, \pi_\theta(a_t|s_t)\hat{A}_t\right], \tag{1}$$

where $\hat{\mathbb{E}}_t[...]$ represents the empiric mean over a finite set of samples, and $\hat{A}_t$ represents the advantage value of an action over others. Advantage usually has the form

$$\hat{A}_t(s_t, a_t) = Q(s_t, a_t) - \hat{V}(s_t), \tag{2}$$

where $\hat{V}(s_t)$ is a function that measures how advantageous it is for the agent to be in state s_t and follow the policy π to reach the goal, and $Q(s_t, a_t)$ is a function that calculates the expected reward that the agent will accumulate being in state s_t, performing action a_t and then follow policy π.

The estimator $\hat{g}$ is obtained through differentiating the objective function

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[log \, \pi_\theta(a_t|s_t)\hat{A}_t\right]. \tag{3}$$

However, perform multiple optimization steps for the loss (3) for the same trajectory can lead to large variations on policy parameters π_θ, resulting in instability during training and low efficiency [12].

As a strategy to overcome this limitation, PPO uses another objective function, called in the literature clipped surrogate objective, in the form

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]. \tag{4}$$

The first term of the function to be minimized,

$$r_t(\theta) = \frac{\pi_\theta(a_t, s_t)}{\pi_{\theta_{old}}(a_t, s_t)}, \tag{5}$$

represents the probability ratio between the actual and old policy. With respect to the second term, $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$, is charged with limiting $r_t(\theta)$ on interval $[1-\epsilon, 1+\epsilon]$, in which ϵ is a hyperparameter used to adjust the length of this interval. Thus, for positive values of $\hat{A}_t$, r_t will not reach values greater than $1+\epsilon$, otherwise, it will not reach values lesser than $1-\epsilon$. Finally, final result is the minimum between these two terms. Adopting this strategy, policy π_θ is updated without distancing itself too much from the old policy, resulting in greater training stability [12].

A. Reinforcement Learning with Augmentation Data

According to [9] and [13], although recent advances in using convolution neural networks to process raw images combined with reinforcement learning, it has been empirically observed that current methods are still lacking on data-efficiency and generalization. Therefore, if it is possible to provide better state representations to the RL method, the agent will probably improve its quality and performance.

Augmentation image data is a strategy that can be incorporated on inputs observations for RL pipelines to mitigate these problems. It consists in making transformations on input images (such as random crop, flip, rotate, color-jitter, cutout, translate, etc), forcing the CNN's responsible to encode images to learn consistencies in their internal representations, resulting in better representations that will be posteriorly fed into policy and value networks [13]–[15].

Prior works used augmentation data with reinforcement learning with no large modifications on underlying model-free RL algorithms. Augmented images are used in two main ways: in Reinforcement Learning with Augmented Data (RAD) [13] they are straightly used to predict the next actions within the policy network; on the other hand, in Data-regularized Actor-Critic (DrAC) [14] and Data-regularized Q (DrQ) [15], they are used to compute the value loss and update value network.

III. Method

A. Problem Description

In this work we aim to train a vehicle so that it can navigate from an arbitrary position A to an arbitrary position B autonomously in a simulation urban scenario composed by T-intersections. The agent is expected to drive at speeds near to 20km/h, stay lane centered, and turn or go straight at intersections when it is needed. Our focus in this work is only regarding on how the agent deals with lane keeping and intersections, so, crosswalks, traffic lights, gear shifting, other vehicles and pedestrians will not be considered.

B. Observation space

To perceive the environment, our vehicle is equipped with a front facing semantic segmentation camera that provides RGB images of size 84x168. In order to get temporal information, the visual observation is composed of the last four environment images stacked. Besides the images from camera, the agent also observes the throttle (t), steering angle (s), vehicle speed (v) values and the angular difference (a) between vehicle heading and the vector of current waypoint.

C. Action space

Since we want to control the lateral and longitudinal movement of the vehicle so that it correctly performs the three maneuvers described above, the action space considered is continuous and two-dimensional, comprising the throttle t

in the range [0,1], where 0 represents no throttle and 1 represents full throttle, and steering s angle in the range of [-1,1], where -1 represents maximum steering to the left and 1 represents maximum steering to the right.

D. State representation

To extract features from the 4-stacked RGB frames collected by the semantic segmentation camera, an encoder inspired in IMPALA [16] deep architecture is used. Fig. 1 shows the encoder architecture, with the number of convolutional and fully-connected layers, number of units, filters, size of kernels and strides, and the activation functions used. It also shows that measurements v, t, s and a are fed into a fully-connected layer with 64 units, and then, its output is concatenated with the output from encoder, resulting in our latent state vector of size 128 that will be posteriorly passed to the sub-policies modules.

E. PPO with sub-policies

Regarding to the ambiguity shortcoming, Codevilla *et al.* [5] proposed an end-to-end conditional imitation learning architecture that turns an intelligent vehicle able to solve the problem of decision making at intersections. Instead of using just one DNN that learns a general policy that tries to solve the intersection problem, they used a branched model in which, for each branch, a DNN learns a sub-policy. Just one sub-policy is activated at a time and they are selected by a set of discrete commands $C = \{c^0, ..., c^k\}$, where k represents the number of possible commands and sub-policies. Fig. 2 shows the architecture of the model proposed. Under this structure, each sub-policy, A^k, is optimized individually depending on which command is active. Thereby, each sub-module is able to specialize in a specific kind of action.

In this study, we combine conditional learning with RL, then, our model is composed by three PPO trainers, each one responsible for one of the three possible vehicle maneuvers:

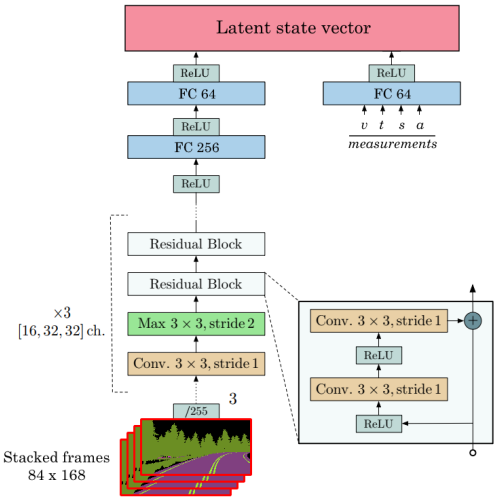

Fig. 1. Composition of the state representation vector. Adapted from [16]

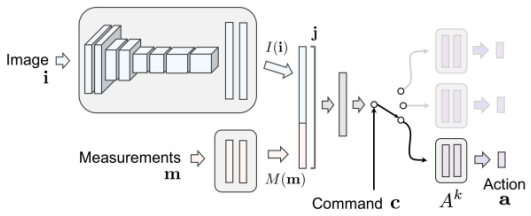

Fig. 2. Branched conditional learning: a specialized sub-module is chosen depending on a specific condition [5].

follow lane, turn right and turn left. The active sub-module in a given step will be defined by a high level command from the local planner, which tells the car which maneuver to perform, just like GPS applications for driving directions. The policy and value networks of each sub-module is composed by three fully connected layers, with 500 and 300 units in the hidden layers, and 2 units in the output layer of policy network and 1 unit in the output layer of value network. To compute the policy output we used tanh and to compute value we used linear as activation function. On the hidden layers the activation function used is leaky relu.

F. Augmentation Data

In this work it will be evaluated the effects of three different types of image augmentation in our learning method for our autonomous driving environment. We choose the transformations that leverage the best results in works from [13], [15], and [14]: random crop, translation, and cutout color.

G. Reward

The reward function that we used takes into account three aspects of vehicle movement: vehicle speed v; center lane deviance d; and the angle a between vehicles heading and current waypoint. The immediate reward is computed by multiplying auxiliary functions that evaluate these three vehicle variables, as follow,

$$r(v, d, a_r) = \begin{cases} -10 & \text{on infraction} \\ v_r(v) * d_r(d) * a_r & \text{otherwise} \end{cases} \quad (6)$$

where,

$$d_r(d) = 1 - \frac{d}{d_{max}}, \quad (7)$$

$$a_r = \begin{cases} 1 - \left|\frac{a_{diff}}{a_{max}}\right| & |a_{diff}| < a_{max} \\ 0 & \text{otherwise} \end{cases}, \quad (8)$$

and

$$v_r(v) = \begin{cases} \frac{v}{v_{min}} & v < v_{min} \\ 1 & v_{min} \leq v < v_{target} \\ \frac{v - v_{target}}{v_{max} - v_{target}} & v \geq v_{target} \end{cases} \quad (9)$$

For the calculation of immediate reward, we consider infractions when the vehicle deviates more than d_{max} from the center of the lane, when the vehicle speed reaches less than 1 km/h after 5 seconds of the episode beginning or when it surpasses the maximum speed v_{max}.

IV. Simulation and Results

In this work, the simulation were performed on CARLA Simulator [17]. CARLA is a simulation environment for training and validating machine learning and control techniques applied to autonomous driving tasks. This simulator presents different scenarios with high level of realism, offering the possibility to change parameters such as weather, lighting, etc. Moreover, it features a range of vehicles and sensors, such as cameras, LIDAR and radar, to equip the autonomous system.

Training experiments were carried out on Town 2, illustrated in Fig. 3-a, in three stages: in first stage, the agent was set to learn to complete two laps in a simple circuit, following the sequence 1-2-3-4-5-6-1; in the second stage, using the model with best performance of previous stage, the agent was set complete two laps in a more complex circuit, following the sequence 1-2-3-6-5-4-3-6-1, being forced to turn at intersections; and finally, in third stage, using the best agent from second stage, agent was set to learn to navigate from an arbitrary location A to another arbitrary location B. For all stages, we run an evaluation of the policy learnt every 10 training episodes. After finishing the training process, we use the best third-stage model so the vehicle drives random routes on Town 1, a map with similar road structure of Town 2, but larger and with different surroundings. Thus, we could verify and validate the agent capability of generalization.

To evaluate our proposed method, we ran and compared VAE-PPO [9], a version of PPO with no augmentation applied, and three versions of PPO with augmentation, that will be referred as PPO cutout, PPO translate and PPO crop. In the original work from Vergara, he pre-trained a Variational Autoencoder (VAE) with 10,000 images captured on Town 7 from CARLA, which is a rural area. Therefore, we also pre-trained a VAE, but with 10,000 images captured on Town 2, with the default hyperparameters found on his code. To train the VAE-PPO agent we also used the default hyperparameters except learning rate, which we used the same for all models.

The hyperparameters of PPO algorithm and other training details that are common to all stages are: discount of 0.99; GAE λ of 0.95; horizon of 512 steps; 3 epochs of training; 32 minibatches per epoch; Adam optimizer; d_{max} = 3m; a_{max} = 180º; v_{max} = 25 km/h; v_{min} = 15km/h; and v_{target} = 20 km/h. Hyperparameters that are unique to a specific stage will be commented in its respective subsection.

A. Software and Hardware settings

The proposed method implementation was written in Python 3.7 with TensorFlow 2.3.0. The simulations and training were run on CARLA Simulator version 0.9.5, on a system with a single Nvidia GTX 970M with 3 GB of video memory, an Intel i7-4710HQ 4-core CPU 2.50GHz, 16GB of RAM, and Windows 10 (64 bits). Code was strongly inspired by OpenAI Baselines PPO implementation [18], VAE-PPO [9] and RAD [13]. Complete details about the implementation can be found at https://github.com/vitoraugustoss/carla.

B. Stage 1: simple circuit

At this stage, we set the vehicle to drive twice around the city in a clockwise direction, going straight through any intersections it encounters. Thus, only one sub-policy (follow lane) will be trained. The purpose of this is to focus on learning to drive within the limits of the lane at speeds close to the target. Learning rate used at this stage is 1e-4, with exponential decay rate of 0.9 every 4,000 updates, clipping parameter of 0.2, initial σ of 0.15 and entropy bonus of 0.01 for follow lane sub-policy, and 1.0 and 0.001 for the other sub-policies.

Fig. 4 presents the trend of cumulative reward of each agent best run under training and evaluation. For better visualisation, we used the smoothing feature from TensorBoard with smooth factor of 0.93.

Fig. 4-a shows that PPO agents, with and without augmentation, presented ascending learning curve over training time. The agents with best performance during training were PPO cutout and PPO without augmentation, presenting similar learning curves. Regarding to PPO crop and PPO translate, former learning curve rises earlier but the later converges to a higher value at the end. On the opposite way, the VAE-PPO agent have not presented good performance, once its learning curve does not tend to rise in any moment. It can be explained by the fact that this method needs to pre-train a VAE with well representative samples. Perhaps with better samples, the results would be better as well. Fig. 4-b shows that behaviour of learning curves during evaluation are similar to training. The only difference that one can notice

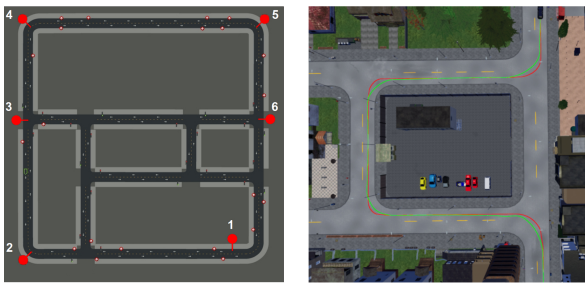

Fig. 3. Map of Town 2 of Carla Simulator (a) and region of the map showing a path driven by the agent (b).

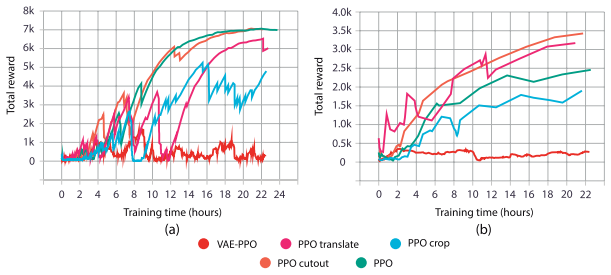

Fig. 4. Trend of learning curve during training (a) and evaluation (b) on stage 1.

is that PPO translate trends to higher values compared to PPO without augmentation. Table I presents metrics of the best models of each agent during evaluation: average speed in km/h; average center lane deviation in meters; and route completion (%). One can notice that PPO agents with some type of augmentation presented results closer to the ideal (speed target = 20 km/h and minimum center deviance as possible).

C. Stage 2: addressing intersections

At this stage, we set the previous stage best agent to drive two laps of a more complex circuit, being necessary to turn left and right at some intersections. Therefore, all sub-policies were optimized on this stage. Learning rate used is 3e-5, with exponential decay rate of 0.95 every 10,000 updates, clipping parameter of 0.15. In this experiment, we only considered the agents PPO and PPO cutout, which presented the best results on previous stage, to verify the influence of augmentation. VAE-PPO was discarded because of poor results on the first experiment.

Fig. 5 presents the trend of cumulative reward of each agent best model during training and evaluation, respectively. Graph of training learning curve was smoothed by a factor of 0.99. On Fig. 5-a one can see that both learning curves rise over training time. Although PPO curve started climbing latter, it reached higher values at the end in comparison with PPO cutout curve. On the other hand, analysing Fig. 5-b, one can clearly notice that PPO cutout agent had better results than PPO agent during evaluation.

D. Stage 3: random routes on Town 2

At this stage, we set the previous stage PPO cutout agent best model to drive random routes, starting from an arbitrary location A to another arbitrary location B. Once the agent already know how to behave when encountering crossroads at this point, we choose a low learning rate of just 1e-6 for a fine tuning of model parameters. Also for this reason, we reduce the initial gaussian noise of both trainers responsible for turning right and left to 0.15, because too much exploration is not needed anymore. We expected the car to complete routes successively until the total distance reaches 3000 m. Thus, we guarantee that vehicle drives by challenging routes, not finishing an episode just driving short and easy ones.

Table II shows how best agent acted on both training and evaluation mode. Although agent presented higher total

TABLE I

PERFORMANCE OF THE BEST MODEL OF EACH AGENT DURING EVALUATION

Agent	Avg speed (km/h)	Avg center dev (m)	%
VAE-PPO	18.28	0.53	37.04
PPO	17.08	0.24	**100**
PPO cutout	18.24	**0.04**	**100**
PPO crop	**18.43**	0.12	**100**
PPO translate	17.98	0.10	**100**

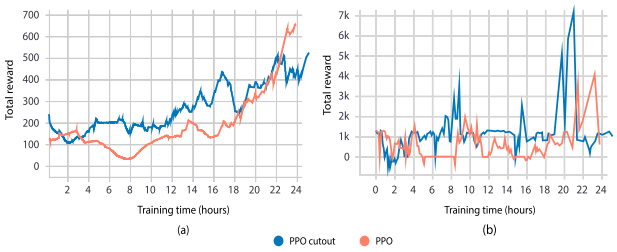

Fig. 5. Trend of learning curve during training (a) and evaluation (b) on stage 2.

TABLE II

PERFORMANCE OF THE BEST MODELS ON SCENARIO 3

Metrics	Training	Evaluation
Avg speed (km/h)	16.92	**19.47**
Avg center dev (m)	**0.16**	0.17
Distance (m)	**3000**	**3000**
Total reward	**13667**	10244

cumulative reward on training, the agent behaviour on evaluation is more appreciated because its average speed is closer to the speed target and its average center lane deviation is almost as low as the one on training mode.

In order to analyze the agent's speed profile and its control inputs while the vehicle is driving, we considered the path indicated by the red line in the Fig. 3-b, in which the vehicle must handle four different intersections. Fig. 6-a presents how the control inputs vary along the path. We can see that steering input (blue) has two valleys and two peaks, representing the moment of left and right turns, respectively. Also, one can note that the throttle input (red) decreases its values when peaks and valleys occur in the steering input (blue). This behaviour indicates that the vehicle decreases its speed to perform turns properly. Before and after steering peaks and valleys, we can see that steering and throttle values have small variance, maintaining steering near to zero and throttle approximately in 0.55. This means that vehicle keeps centered at constant speed on straight stretches of the path. These considerations about vehicle's velocity profile are also shown in Fig. 6-b, in which velocity value decreases at the beginning of turns and increases toward around 20km/h (target speed) when the turns are finished. Moreover, Fig. 3-b also attests the agent's good performance, in which we

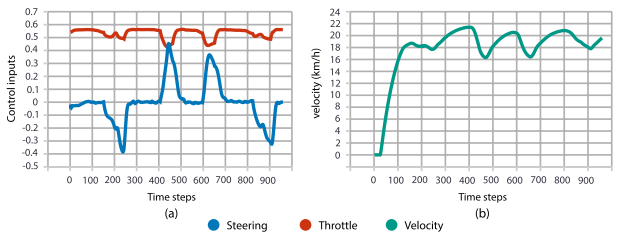

Fig. 6. Control inputs (a) and velocity profile (b) for path with intersections and turns.

TABLE III
PERFORMANCE OF THE BEST MODELS ON TOWN 1 AND 2

Metrics	Test (Town 1)	Evaluation (Town 2)
Avg speed (km/h)	20.33	**19.47**
Avg center dev (m)	**0.13**	0.17
Max distance (%)	**100**	**100**
reward/m	3.077	**3.415**

observed that the path driven by the car (green) is very similar to the expected (red).

E. Test: random routes on Town 1

After finishing all three stages of training process, we took our resulting best model and tested it on Town 1. The object in this test is the same of stage 3 of training: complete random routes until the total distance travel reaches a target distance. As Town 1 is larger than Town 2, we set the maximum distance to 5000 m. Table III presents the performance or our agent on test environment.

Analysing these results one can perceive that the agent performance is similar for both maps, meaning that it gained good capability of generalization during training process. For better visualization of vehicle's performance, please refer to the videos in `https://youtube.com/playlist?list=PLNGyU4O2fqprOd30Fe5EIU1_MGh9MoyPR`.

V. CONCLUSION

In this work, we presented an end-to-end framework based on multiple PPO trainers to address decision making for self-driving cars at intersections in urban scenarios on CARLA Simulator. Learning sub-policies specialized in solve minor tasks promises to mitigate the ambiguity problem. We also applied different types of image augmentation that have presented good results regarding to sample efficiency in recent works to verify whether its use will also reflect in good performance in our problem.

Results show that deep conditional reinforcement learning is a viable approach to solve the problem of ambiguity on autonomous driving domain. We could also verify that the application of augmentation data can improve the learning process and the agent generalization capability. However, it is important to highlight that not every image transformation will leverage gains on training. Augmentation function has to be chosen according to the task. Naively choosing image transformation can produce worse results than not using any augmentation. In resume, after training process, our vehicle is capable to drive from any start point to any destiny both on the map used on training (Town 2) and on map used to testing (Town 1).

In future works, we aim to improve the agent decision making capability, such that the vehicle also behave properly when encountering crosswalks, traffic lights and roads with multiple lanes. Moreover, we intend to improve the collision avoidance feature, making the vehicle to avoid colliding not just with static objects, such as poles, fences and speed limit boards, but with moving obstacles as well, such as other vehicles and pedestrians.

REFERENCES

[1] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[2] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 187–210, 2018.

[3] J. Chen, S. E. Li, and M. Tomizuka, "Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–11, 2021.

[4] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58443–58469, 2020.

[5] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4693–4700, 2018.

[6] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8248–8254, 2019.

[7] K. Min, H. Kim, and K. Huh, "Deep distributional reinforcement learning based high-level driving policy determination," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 3, pp. 416–424, 2019.

[8] M. Stang, D. Grimm, M. Gaiser, and E. Sax, "Evaluation of deep reinforcement learning algorithms for autonomous driving," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1576–1582, 2020.

[9] M. L. Vergara, "Accelerating training of deep reinforcement learning-based autonomous driving agents through comparative study of agent and environment designs," Master's thesis, Norwegian University of Science and Technology, 2019.

[10] M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun, "Driving policy transfer via modularity and abstraction," *CoRR*, vol. abs/1804.09364, 2018.

[11] B. Osinski, A. Jakubowski, P. Zicina, P. Miłoś, C. Galias, S. Homoceanu, and H. Michalewski, "Simulation-based reinforcement learning for real-world autonomous driving," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6411–6418, 2020.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms.," *CoRR*, vol. abs/1707.06347, 2017.

[13] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," *Advances in Neural Information Processing Systems*, vol. 2020-December, 2020.

[14] R. Raileanu, M. Goldstein, D. Yarats, I. Kostrikov, and R. Fergus, "Automatic data augmentation for generalization in deep reinforcement learning," *CoRR*, vol. abs/2006.12862, 2020.

[15] D. Yarats, I. Kostrikov, and R. Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," in *International Conference on Learning Representations*, 2021.

[16] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* (J. G. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 1406–1415, PMLR, 2018.

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

[18] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Z. Tan, Y. Wu, and P. Zhokhov, "Openai baselines." `https://github.com/openai/baselines`, 2017.