Generative Adversarial Imitation Learning for End-to-End Autonomous Driving on Urban Environments

Gustavo Claudio Karl Couto
Automation and Systems Engineering Department
Federal University of Santa Catarina
Florianopolis, Brazil
gustavo.karl.couto@posgrad.ufsc.br

Eric Aislan Antonelo
Automation and Systems Engineering Department
Federal University of Santa Catarina
Florianopolis, Brazil
eric.antonelo@ufsc.br

***Abstract*—Autonomous driving is a complex task, which has been tackled since the first self-driving car ALVINN in 1989, with a supervised learning approach, or behavioral cloning (BC). In BC, a neural network is trained with state-action pairs that constitute the training set made by an expert, i.e., a human driver. However, this type of imitation learning does not take into account the temporal dependencies that might exist between actions taken in different moments of a navigation trajectory. These type of tasks are better handled by reinforcement learning (RL) algorithms, which need to define a reward function. On the other hand, more recent approaches to imitation learning, such as Generative Adversarial Imitation Learning (GAIL), can train policies without explicitly requiring to define a reward function, allowing an agent to learn by trial and error directly on a training set of expert trajectories. In this work, we propose two variations of GAIL for autonomous navigation of a vehicle in the realistic CARLA simulation environment for urban scenarios. Both of them use the same network architecture, which process high-dimensional image input from three frontal cameras, and other nine continuous inputs representing the velocity, the next point from the sparse trajectory and a high-level driving command. We show that both of them are capable of imitating the expert trajectory from start to end after training ends, but the GAIL loss function that is augmented with BC outperforms the former in terms of convergence time and training stability.**

***Index Terms*—autonomous driving, generative adversarial imitation learning, CARLA simulator, behavior cloning**

I. Introduction

Imitation learning is an approach whereby a model is created to imitate an expert by training on a fixed set of observation-action samples (or trajectories) obtained from that expert. This happens without the possiblity of querying the expert while training. Behavioral cloning (BC) [1]–[3] is one approach for imitation learning that relies on supervised learning to learn a mapping between observations and actions. It has been used for autonomous navigation since 1989, starting with the self-driving car ALVINN [1] that relied on camera images as input to a neural network (NN) to drive the car. BC has the issue of sample complexity since it requires a lot of training data (observation-action samples) generated by experts (e.g. human drivers) to work well in practice. However, BC will always suffer from cascading errors and covariate shift [4] since their models are trained only on a subset of the necessary samples (observation-action pairs) for safe, robust driving: as soon as the self-driving car encounters a new road and starts shifting slightly towards the left or right side of the lane, it will feedback its mistake through new observations fed to the NN, which in turn will shift even more the car until no valid action can be taken anymore.

On the other hand, policies learned by Reinforcement Learning (RL) solve the issue of cascading error since they learn from information of whole sample trajectories and not just isolated observation-action samples as in BC, but require a reward (cost) function to be defined for finding the optimal policy. In RL, training is an evolutionary method where an agent learns by trial and error, i.e., interacting with the environment, and receiving a reward signal indicating the quality of the solution found.

In the context of imitation learning, RL can be used to learn to imitate expert driving trajectories in a process called inverse reinforcement learning (IRL) [5], [6]. IRL can be used to find driving policies by: first finding a cost function under which the expert, i.e., the set of training trajectories, is uniquely optimal; and then using RL algorithms that optimize the learned cost function. IRL is usually expensive to run since it requires RL in an inner loop, and thus, has difficulties in scaling to large environments. Recent work in IRL seeks to deal with these issues [7], [8]. Still, learning a cost function in IRL makes the problem more computationally expensive than just learning a policy directly from the training trajectories.

One of the recently developed sample refiefficient approaches for imitation learning comes from Generative Adversarial Imitation Learning (GAIL) framework [9], which can actually generate policies directly from the expert trajectories without having to learn any cost function as in IRL, and is scalable to relatively large environments, such as the one in autonomous driving. However, training in GAIL can be unstable and difficult to achieve a satisfactory result depending on the task. While it is sample efficient in terms of the required number of expert trajectories, it is not so efficient in the number of environment interactions needed for convergence.

On the other hand, BC converges in a few epochs, but assumes that its dataset is composed of independent and identically distributed samples. A recently developed approach [10] combines both BC and GAIL losses into an integrated loss function for stable and sample efficient imitation learning. They have evaluated it on low-dimensional control tasks, and also on the high-dimensional image-based task of CarRacing from OpenAI Gym.

In our work, we propose a GAIL-based architecture for end-to-end autonomous urban navigation, which is evaluated on fixed trajectories in the realistic autonomous driving CARLA simulator [11]. The agent receives a high-dimensional image input from three frontal cameras, as well as other continuous inputs such as velocity and next point of a sparse GPS trajectory in the local vehicle's frame of reference. As far as the authors know, this is the first proposal of architectures based on conventional GAIL and GAIL augmented with BC [10] for end-to-end imitation learning in the CARLA simulator, also considering a much higher observation space (in relation to the simpler CarRacing environment, for instance). Our experiments have shown that although the GAIL architecture can learn to imitate the expert well, GAIL augmented with BC has much faster convergence to the desired navigation trajectory.

In Section II, we give a brief overview on some related works. Next, we present the main methods such as GAIL and GAIL augmented with BC as well as the agent architecture in Section III. Section IV describes the experiments, datasets and settings, while the results are presented in Section V. Conclusions and future work are drawn in Section VI.

II. Related Work

A. Behavior cloning

One of the most important recent works on imitation learning for end-to-end autonomous driving on urban environments corresponds to a conditional imitation learning algorithm implemented in [12]. Based on behavior cloning, the learning is conditioned on a high-level command signal that indicates the way through the trajectory to be followed by the agent. The observation space consists of: images from three frontal cameras installed on the vehicle; the vehicle speed if available; and a high-level driving command. The action space consists of the vehicle's steering angle and acceleration. The experiments are conducted both on CARLA and on a real setting with an off-the-shelf 1/5 scale truck, which was adapted with three frontal cameras and a single board computer for autonomous driving.

Both experiments were successful and also generalized well, making it a milestone for enabling an agent to learn to follow generic trajectories, whereas most of previous works have concentrated on fixed paths. The work has shown that CARLA serves as an important platform to analyse agents and learning approaches before deploying them to the real world.

B. Reinforcement learning

The Controllable Imitative Reinforcement Learning [13] was proposed as a two-phase algorithm that pre-trains a policy using behavior cloning and then refines it during interaction with the environment using an engineered reward function to enforce the best behavior of the agent. This reward signal is composed by negative rewards for abnormal steer angles, damage from collisions, going over the sidewalk or the opposite lane, and by a positive reward for reaching a desired speed.

The observation space of the algorithm consists of an image from a front-facing camera, the vehicle speed and four options of high-level commands ("follow-lane", "turn left", "turn right", "go straight"). The image from the camera feeds the convolution layers of the actor-critic network while the speed is fed directly to the fully connected layers of the network. The high-level command is used to select the final branch of the network, that outputs the signals for the steering wheel, throttle, and brake.

C. Apprenticeship Learning

In [14], a multi-stage learning approach is employed that succeeded in the CARLA benchmark [15]. Their method is based on the training of a teacher in a first stage using behavior cloning, which has privileged information about the landscape and other agents on its observation space in the CARLA simulator. In the second stage, a vision-based agent is trained without access to privileged information using apprenticeship learning [16]. .

The observation space of the agent consists of a 384x184 RGB image from a front-facing camera and the vehicle velocity. The action space corresponds to K waypoints representing the agent's future locations on the next K states in the agent reference. The waypoints are generated by network's four heads representing the following high-level commands "follow-lane", "turn left", "turn right", "go straight".

A low-level rule-based controller uses those waypoints predicted by the network and the high-level command given to the vehicle to generate the car attitude control (steer, throttle, brake).

D. GAIL for autonomous driving

GAIL was first applied to autonomous driving in [17]. In that work, a Wasserstein Gail is designed to control a vehicle on TORCS [18], an open-source racing car simulator. The observation space consists of images taken from the front of the car, and some auxiliary information (the car velocity, the last two actions, and the damage to the car). The action space corresponds to a three-dimensional vector with the steering command, acceleration, and brake.

Their method, called InfoGAIL, still augments the standard GAIL with a replay buffer and a reward signal with constant reward to encourage the agent to stay alive. The focus of InfoGAIL is to demonstrate a capability to learn a policy that can switch between driving behaviors by disentangling in an unsupervised way the different modes of behavior present in

the expert's demonstrations. Other related work can be found in [19]–[21].

To our knowledge, this is the first time that GAIL or GAIL augmented with BC are evaluated for an end-to-end self-driving task in a highly realistic urban vehicle simulator as CARLA.

III. Methods

A. Generative Adversarial Imitation Learning

In Generative Adversarial Imitation Learning (GAIL) [9], basically, there are two components that are trained iteratively in a min-max game: a discriminative classifier D is trained to distinguish between samples generated by the learning policy π and samples generated by the expert policy π_E (i.e., the labelled training set); and the learning policy π is optimized to imitate the expert policy π_E. Thus, in this game, both D and π have opposite interests: D feeds on state-action pair (s, a) and its output seeks to detect whether (s, a) comes from learning policy π or expert policy π_E; and π maps state s to a probability distribution over actions a, learning this mapping by relying on D's judgements on state-action samples (i.e., D informs how close π is from π_E). Mathematically, GAIL finds a saddle point (π, D) of the expression:

$$\mathop{\mathbb{E}}_{\pi}[\log(D(s,a))] + \mathop{\mathbb{E}}_{\pi_E}[\log(1 - D(s,a))] - \lambda H(\pi) \quad (1)$$

where $D : S \times A \rightarrow (0, 1)$, S is the state space, A is the action space; π_E is the expert policy; $H(\pi)$ is a policy regularizer controlled by $\lambda >= 0$ [22]. GAIL works similarly to generative adversarial nets (GANs) [23], which was first used to learn generators of natural images. Both D and π can be represented by deep neural networks. In practice, a training iteration for D uses Adam gradient-based optimization [24] to increase (1), and in the next iteration, π is trained with any on-policy gradient method such as Proximal Policy Optimization (PPO) [25] to decrease (1).

B. GAIL and BC augmentation

1) Wasserstein loss: Instead of the original loss function of GAIL, as in (1), in this work, we employ its improved version using the Wasserstein distance between the policy distribution $P\tau_\pi$ and expert distribution $P\tau_E$ as loss function for training the discriminator, as in [17], [26].

The Wasserstein distance measures the minimum effort to move one distribution to the place of the other and gives a better feedback signal than the Jensen-Shannon divergence.

The new loss function for the improved GAIL is:

$$\mathop{\mathbb{E}}_{\pi_E}[D(s,a)] - \mathop{\mathbb{E}}_{\pi}[D(s,a)] - \lambda H(\pi) - \lambda_2 L_{gp} \quad (2)$$

where the discriminator will try to increase (2), while π seeks to minimize it; and L_{gp} is a loss that penalizes the gradient constraining the discriminator network to the 1-Lipschitz function space, according to [27].

2) BC augmentation: The behavior Cloning loss function can be defined as:

$$- \mathop{\mathbb{E}}_{\pi_E}[\log(\pi(a|s|))] \quad (3)$$

which is the negative expectation of the log probability for the non deterministic policy generator to output the same actions as the expert on the same states from the expert dataset.

The BC augmentation is constructed taking a point from a line between the behavior cloning loss and the GAIL loss, as defined on the following equation:

$$\alpha L_{bc} + (1 - \alpha) L_{GAIL} \quad (4)$$

On equation (4), L_{bc} is the behavior cloning loss function defined on (3) and L_{gail} is the GAIL loss function defined defined on (2).

The α on (4) controls the participation of each term during the training. By the start of the GAIL training, the discriminator is yet not fully trained and the behavior cloning participation should be stronger. For that, α should not be the same during the entire training and its value decreases during the training using a fixed decay factor. This definition and the practical implementation follows [28].

C. Agent and Network architecture

1) Agent: The autonomous car has several sensors, from which we consider: three frontal cameras (Fig. 1), an inertial unit used to compute the vehicle linear speed and angular position, and a GPS unit for global positioning.

Before training begins, the agent has access to the whole trajectory it must perform, defined as a vector of sparse points and high-level driving commands that characterize the trajectory with no ambiguity. These driving commands can be one from the following in this work:

- LANE_FOLLOW: Continue in the current lane.
- LEFT: Turn left at the intersection.
- RIGHT: Turn right at the intersection.

Thus, the agent can use this trajectory to know which route the car should follow. In practice, this is accomplished by a route planner, that monitors the agent's progress and sends him the next target position in the car's frame of reference as well as the high-level driving command. These two data, totalling 8 dimensions, are given as input to the agent. Notice that the command is input as an one-hot encoded vector.

2) Networks architecture: The networks represented in Fig. 2 are composed of a convolutional block of four layers, with kernel size of 4 and stride of 2. Each layer in this block is followed by a leaky ReLU activation function, and the numbers of channels starts in 32 on the first layer and is multiplied by 2 on every new layer, ending with 256 channels.

That convolutional block is followed by a fully-connected network block with two layers, with leaky ReLU activation function for the first hidden layer. The second layer represents the output of the architecture.

Fig. 1. Images from the three frontal cameras located at the left, central, and right part of the vehicle, respectively. They were taken after the first few interactions of the agent in the CARLA simulation environment considering our defined trajectory. Each camera produces a RGB image with 144 pixels of height and 256 pixels of width. These images are fed to the networks as they are. .

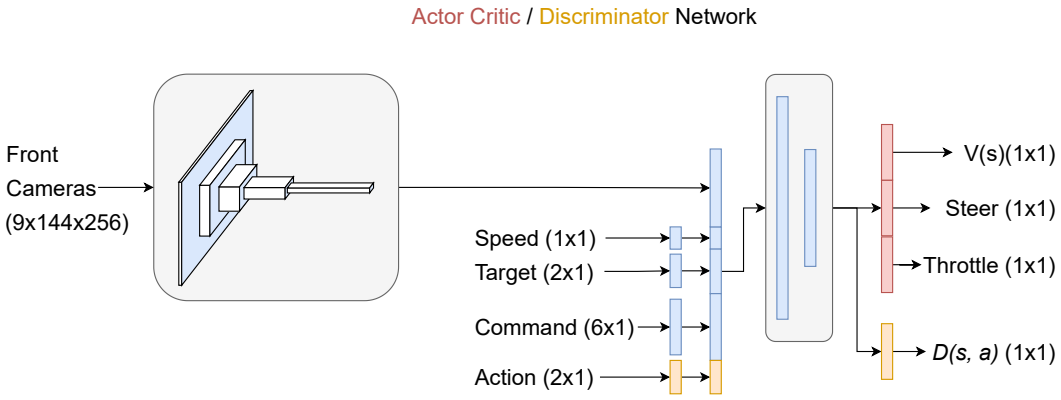

Fig. 2. Architecture of the actor-critic network and discriminator - each of them has its own separate network, with the latter having an additional input for the action, in orange color, and a sigmoidal output $D(s, a)$ instead of the output layer of the actor-critic network which consists of the steering direction, throttle as actions for the actor (policy) and value of the current state $V(s)$ for the critic. The common, though not shared architecture (in blue) is composed of a convolutional block that process the images of the three frontal cameras, whose output features are concatenated with other nine continuous inputs for speed, next target point in the sparse GPS trajectory, and a high-level driving command. The resulting feature vector is input to a block of two fully-connected (FC) layers.

Both actor-critic and discriminator networks follow that same architecture, although they do not share parameters. The inputs to both networks correspond to 256x140 RGB images from the three frontal cameras. When stacked, these images yield an input with 9 channels, that is fed to the convolution block (Fig. 2). The other continuous input is the car's linear velocity, which is concatenated with the 8-dimensional input from the trajectory as well as to the flattened feature vector from the last convolutional layer. The discriminator has an additional continuous input for the action.

For the actor-critic network, three outputs compose the last layer of the network: a linear unit for the value $V(s)$, a $\tanh$ unit for the steering wheel action, and a sigmoid unit for the throttle action, restricting the outputs to the valid domain of these commands [17].

3) Non deterministic policy: The agent learning process is based on the use of a stochastic policy to calculate action probabilities. This is achieved by using the Gaussian distribution, whose mean is predicted by the policy network, and the standard deviation is fixed to a predefined value [17]. This was necessary because a variable entropy was shown to be not suitable: the agent with a high entropy is easily disturbed on sensitive moments like a turn, whereas there is not enough exploration during turns if the entropy is too low.

IV. Experiments

The learning navigation experiments are inspired on the CARLA Leaderboard evaluation platform and consists of navigating autonomously on two setups: a short route of 100 meters and one turn (setup 1); and a long route of $2,500$ meters and four turns (setup 2). The long route was chosen from the ones available in the CARLA Leaderboard [15]. The short one corresponds to the first 100 meters of the long route.

A top down image from the simulator presenting each turn from the trajectories is displayed on (Fig. 3).

A. Dataset

The expert dataset is built using a deterministic agent that navigates using a dense point trajectory and a classic PID controller [14]. While a dense point trajectory provide many points at a finer resolution, a sparse point trajectory is made of considerably less points to follow, providing just a sense of the right direction to the agent. Thus, the former is used to generate training data by the expert, while the latter is used by the agent for more high-level directions. For instance, the first setup (the short route) considers 80 and 4 points for the dense and sparse trajectories, respectively. The second setup (the long route) uses 760 points in the dense trajectory, and 20 points in the sparse one.

Fig. 3. The top-down view of the simulation with the car in the center and making a turn for the long route. Each picture shows one of the four possible turns, from left to right: left, left, right, and right turns.

For both setups, 10 complete trajectories were recorded at 10 hertz, i.e., 10 observation-action pairs per second were generated. For the short route, those trajectories correspond to 5 minutes of driving as if in a real scenario, totalling $3,000$ training samples (4GB of uncompressed data). For the long route, those trajectories correspond to half an hour of driving, totalling $18,000$ training samples (30GB of uncompressed data).

B. Training

The training was performed using ten parallel actors in a synchronous way, each one running its own CARLA simulator. An eleventh CARLA simulator was also run for evaluation purposes.

In a simulation, every episode starts with the vehicle at zero speed on a particular initial point. The episode ends at every infraction, collision or lane invasion and a new episode starts with the vehicle initially located where the infraction occurred with 90% chance. With 10% chance, the location in the trajectory is randomly chosen, in order to diversify the experience for each policy update.

For the short (long) route, 240 (720) environment interactions or timesteps are recorded for every actor and then the resulting training set of $2,400$ $(7,200)$ samples is used to train the parametrized policy in a central computer using (4) as loss function. Thus, the episode does not have to end for a policy update to happen. Notice that any one of the ten actors can be interacting with the environment in different parts of the trajectory at a certain moment. Other hyperparameters can be seen in Table I. For instance, the standard deviation of the Gaussian distribution (σ_1 for steer and σ_2 for throttle) for the policy is fixed to a predefined value. Thus, the output of the policy network only affects the mean of the distribution.

A behavior cloning (BC) agent is also trained for comparison, using the same dataset available for the GAIL agents, but 70% of the samples are used for training, while 30% for validation. The dataset is not augmented using random rotations or shifting, so that the techniques are compared on their sample efficiency on the same expert trajectories. The BC agent is trained using the loss function from (3), and an ADAM optimizer with a learning rate of 3.0×10^{-4}. The network with best validation error is then evaluated in the simulation experiments.

TABLE I
HYPERPARAMETERS FOR TRAINING

	Short Route	Long Route
Parallel environments (N)	10	10
Adam step size (lr)	1.0×10^{-4}	1.0×10^{-4}
Number of PPO epochs (K)	4	4
Mini-batch size (m)	300	900
Discount (γ)	0.99	0.99
GAE parameter (λ)	0.95	0.95
Clipping parameter (ϵ)	0.1	0.1
Value Function coefficient (c_1)	0.5	0.5
Entropy coefficient (c_2)	0.0	0.0
Timesteps per epoch (T)	2400	7200
Log Standard Deviation Steer (σ_1)	-2.0	-2.0
Log Standard Deviation Throttle (σ_2)	-3.2	-3.2

V. RESULTS

In order to evaluate the performance of agents, the reward or score metrics is defined as the number of crossed points from the dense trajectory, representing how much of the trajectory the agent has completed without any mistake. Thus, a maximum reward is equivalent to total number of points in the dense trajectory of the particular route.

The learning performance of both GAIL and GAIL augmented with BC (BC_GAIL) can be seen on Figures 4 and 5 for the short and long routes, respectively. In the first setup, BC_GAIL is able to converge significantly faster than GAIL, achieving maximum reward of 80, slightly higher than just behavior cloning. On the second more challenging setup which consists of four turns, the learning takes considerably more time. On average, the agent by BC_GAIL was able to complete the route without any mistake much earlier than the GAIL agent, also showing early fast improvement of the policy. This is possible due to the strong influence of the BC term in the loss function in the early part of the training process. Notice that an agent trained only by BC is not able to solve this task (achieved only a reward of 173.8) by training only on the same dataset as GAIL was trained. In addition, after 15×10^5 environment interactions, we can observe that the average reward stabilizes between 500 and 700 for the stochastic policy of both GAIL and BC_GAIL. The spikes seen in Fig. 5 can be

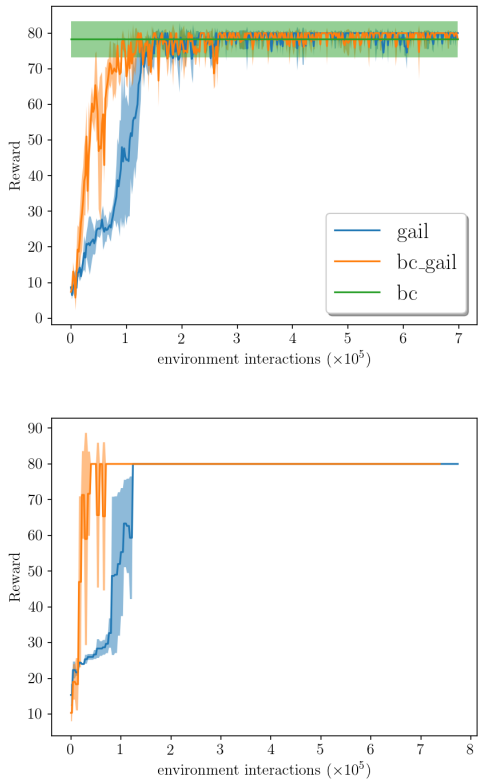

Fig. 4. Average rewards vs environment interactions during training in the short route (setup 1). For each method (GAIL and GAIL with BC), the average performance of three runs (i.e, three agents trained from scratch) is shown with a stochastic policy (top plot) and a deterministic policy (bottom plot). The shaded area represents the standard deviation. The behavior cloning (BC) agent attains an average reward of 78.3 for ten episodes, while the maximum is at 80, achieved by both GAIL and GAIL augmented with BC.

caused by random actions of a stochastic policy which can lead to forgetting of some already acquired skills (such as turning at an intersection) or skills that are not well formed yet. For instance, the agent can learn to make a turn at some point and, after some iterations, fail to repeat that behavior, causing a sudden drop of the reward. This happens because turning is a difficult skill to learn, while the reward is proportional to the traveled distance.

The trajectory of the agent for the long route can be viewed in Fig. 6. It shows the early mistakes in red color made by an BC_GAIL agent in the topmost plot. As training proceeds, less and less mistakes are made as it can be noticed in the remaining plots.

VI. Conclusion

In this work, we have proposed a GAIL-based architecture for end-to-end autonomous driving in urban environments. Despite the known difficulties and learning instabilities of generative adversarial networks, both GAIL and GAIL augmented with BC were able to converge and generate agents able to complete the whole trajectory without mistakes, with the latter able to quickly find a suitable policy when compared to the former. Both of them surpassed Behavior Cloning in performance, which was not able to generate an agent even capable of making more than one turn on average in the long route.

Although the trajectories were fixed beforehand, the architecture is general enough to allow for variable routes, i.e., an agent that can change course in real time depending on a dynamic route, which will be tackled as future work. We also plan to investigate risky scenarios with rare events, and which loss functions or models could be of use to handle those important settings. This is where reinforcement learning approaches can make a difference as BC depends on static collected data which might not be representative of all possible real-world scenarios.

References

[1] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991.

[2] E. A. Antonelo and B. Schrauwen, "On learning navigation behaviors for small mobile robots with reservoir computing architectures," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 4, pp. 763–780, 2015.

[3] E. Antonelo and B. Schrauwen, "Supervised learning of internal models for autonomous goal-oriented robot navigation using reservoir computing," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Anchorage, AK, May 2010, pp. 2959–2964.

[4] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 661–668.

[5] A. Y. Ng, S. J. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *ICML*, 2000, pp. 663–670.

[6] S. Russell, "Learning agents for uncertain environments (extended abstract)," in *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, ser. COLT' 98. New York, NY, USA: ACM, 1998, pp. 101–103. [Online]. Available: http://doi.acm.org/10.1145/279943.279964

[7] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *International Conference on Machine Learning*, 2016, pp. 49–58.

[8] S. Levine and V. Koltun, "Continuous inverse optimal control with locally optimal examples," in *ICML*, 2012.

[9] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573.

[10] R. Jena, C. Liu, and K. Sycara, "Augmenting gail with bc for sample efficient imitation learning," 2020.

[11] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," 2017.

[12] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," 2018.

[13] X. Liang, T. Wang, L. Yang, and E. Xing, "Cirl: Controllable imitative reinforcement learning for vision-based self-driving," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, pp. 604–620.

[14] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," 2019.

[15] Feb 2020. [Online]. Available: https://leaderboard.carla.org/

[16] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010. [Online]. Available: https://doi.org/10.1177/0278364910371999

[17] Y. Li, J. Song, and S. Ermon, "Infogail: Interpretable imitation learning from visual demonstrations," 2017.

[18] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "Torcs, the open racing car simulator," *Software available at http://torcs. sourceforge. net*, vol. 4, no. 6, p. 2, 2000.

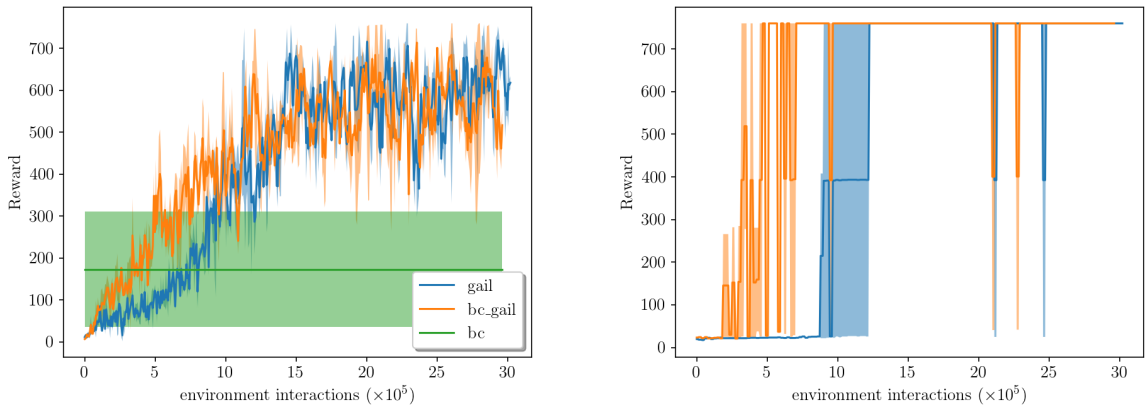

Fig. 5. Average rewards vs environment interactions during training in the long route (setup 2). For each method (GAIL and GAIL with BC), the average performance of two runs (i.e, two agents trained from scratch) is shown with a stochastic policy (left plot) and a deterministic policy (right plot). The shaded area represents the standard deviation. The behavior cloning (BC) attains an average reward of 173.6 for ten episodes, while the maximum is at 760, achieved by both GAIL and GAIL augmented with BC.

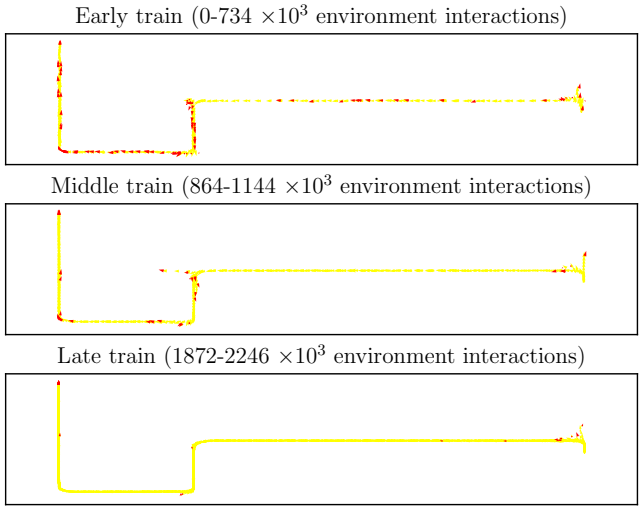

Fig. 6. The vehicle's trajectory, in yellow, for the long route during different moments of the training process. In the early training iterations, errors, marked in red color, are common. As training proceeds, less and less mistakes happen. The trajectory starts at the right side, heading North, and ends at the left side, also heading North.

[19] J. Huang, S. Xie, J. Sun, Q. Ma, C. Liu, J. Shi, D. Lin, and B. Zhou, "Learning a decision module by imitating driver's control behaviors," 2021.

[20] Z. Huang, J. Wu, and C. Lv, "Efficient deep reinforcement learning with imitative expert priors for autonomous driving," 2021.

[21] M. Zhou, J. Luo, J. Villella, Y. Yang, D. Rusu, J. Miao, W. Zhang, M. Alban, I. Fadakar, Z. Chen, A. C. Huang, Y. Wen, K. Hassanzadeh, D. Graves, D. Chen, Z. Zhu, N. Nguyen, M. Elsayed, K. Shao, S. Ahilan, B. Zhang, J. Wu, Z. Fu, K. Rezaee, P. Yadmellat, M. Rohani, N. P. Nieves, Y. Ni, S. Banijamali, A. C. Rivers, Z. Tian, D. Palenicek, H. bou Ammar, H. Zhang, W. Liu, J. Hao, and J. Wang, "Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving," 2020.

[22] M. Bloem and N. Bambos, "Infinite time horizon maximum causal entropy inverse reinforcement learning," in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*. IEEE, 2014, pp. 4911–4916.

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[26] M. Zhang, Y. Wang, X. Ma, L. Xia, J. Yang, Z. Li, and X. Li, "Wasserstein distance guided adversarial imitation learning with reward shape exploration," *2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS)*, Nov 2020. [Online]. Available: http://dx.doi.org/10.1109/DDCLS49620.2020.9275169

[27] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," 2017.

[28] R. Jena, C. Liu, and K. Sycara, "Augmenting gail with bc for sample efficient imitation learning," *arXiv preprint arXiv:2001.07798*, 2020.