Solving Learn-to-Race Autonomous Racing Challenge by Planning in Latent Space

Shivansh Beohar [1] **Fabian Heinrich** [2] **Rahul Kala** [1] **Helge Ritter** [2] **Andrew Melnik** [2]

Abstract

Learn-to-Race Autonomous Racing Virtual Challenge hosted on *www.aicrowd.com* platform consisted of two tracks: Single and Multi Camera. Our *UniTeam* team was among the final winners in the Single Camera track. The agent is required to pass the previously unknown F1-style track in the minimum time with the least amount of off-road driving violations. In our approach, we used the *U-Net* architecture for road segmentation, variational autocoder for encoding a road binary mask, and a nearest-neighbor search strategy that selects the best action for a given state. Our agent achieved an average speed of 105 km/h on stage 1 (known track) and 73 km/h on stage 2 (unknown track) without any off-road driving violations. Here we present our solution and results. The code implementation is available here: *https://gitlab.aicrowd.com/shivansh_beohar/l2r*

1. Introduction

The Learn-to-Race Autonomous Racing Virtual Challenge (Herman et al., 2021)(AICrowd, 2022a)(Beohar & Melnik, 2022) provides an F1-style racing environment for a single player to evaluate the performance of self-learning algorithms. An *Open-AI* gym (Brockman et al., 2016) compliant environment and a *Unreal Engine* based simulator software is provided to develop and test AI agents. In this paper we present our solution and results (AICrowd, 2022b)(Beohar, 2022).

2. Challenge and Environment details

The goal of the challenge is to create autonomous agents which can drive around the tracks in minimum time and with least amount of safety infractions. Safety infractions are described as events in which the agent moves out of the drive-able area, collides with objects like walls or gets stuck somewhere in the track. The primary objective is to get a high success rate, defined as completion % and the secondary objective is to minimize time taken to complete the track.

The challenge presents a single player racing environment, with several tracks (e.g. *Thruxton, Anglesey National, Las Vegas*) with high fidelity graphics and physics. The observations are multi-modal consisting of 384x512 RGB FPVs of front (Single Camera track), and additional left, right areas (Multi Camera track) of the car plus velocity information. The input action space is steering (right - left) and acceleration (braking - acceleration) both scaled to $[-1, 1]$. In our approach we designed the agent to control the speed rather than the acceleration. The required acceleration was calculated using the current speed and desired speed.

In the training phase, apart from the RGB views, the simulator provides segmentation views of the cameras with masks for road, car's hood, ground surrounding the road and sky. However the environment did not provide segmentation masks during the evaluation phase. The competition was split into two stages,

- Stage 1 : Participants have access to the evaluation track (*Thruxton*) and can submit pre-trained models for evaluation.
- Stage 2 : Participants have only access to two tracks (*Thruxton* and *Anglesey National*) for developing the local solution, but the evaluation took place on an unknown track (*Las Vegas*). The participants can upload pre-trained models which can be fine-tuned on the unknown track for 1 hour on an *NVIDIA-T4* based compute VM. The fine-tuned model was then evaluated on the unknown *Las Vegas* track and the final metrics was reported on the leaderboard. Participants cannot access the fine-tuned model weights.

[1]Department of IT, IIIT Allahabad, India [2]Bielefeld University, Germany. Correspondence to: Shivansh Beohar <shivansh.bhr@gmail.com>.

Proceedings of the 39th *International Conference on Machine Learning*, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

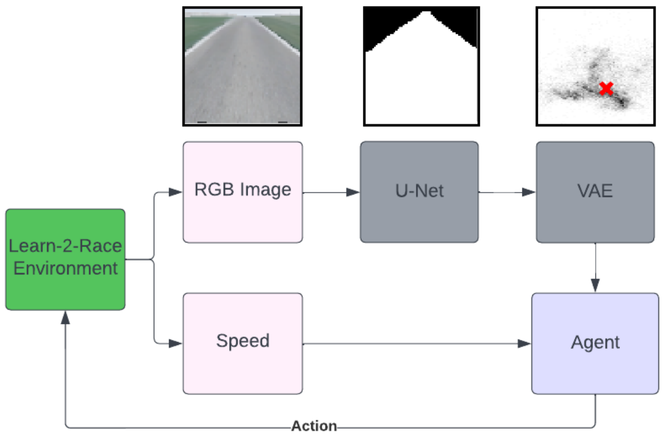

Figure 1. The data-flow pipeline. The pre-processed and resized RGB image arrives at the *U-Net*. The *U-Net* output is a binary mask of the road, reduced to 28x28 and passed to *VAE*. *VAE* performs a two-dimensional latent encoding, red cross represents the encoding of the present binary mask. This encoding, along with speed information from the environment, is used by the agent as a state representation to produce an action.

3. Scene Semantics and State Encoding

Use of semantics as a part of input is a widely adopted approach in autonomous control tasks. Semantics are the important entities in the scene represented often as segmentation masks or coordinates. Using semantic masks instead of (or in combination with) RGB images helps to reduce state complexity while keeping the necessary information in the state representation. Using VAE (Kingma & Welling, 2013) to encode the semantics further help in creating a smooth latent space, which enables better interpolation between the state representations, leading to better noise agnostic learning. (Azizpour et al., 2020) demonstrate a similar End-to-End architecture to train DDPG (Lillicrap et al., 2015) on CARLA town-I task. However, deep neural network based learning algorithms including (Azizpour et al., 2020) require massive amount of data and time to learn good behaviors. They result in robust and high performance, but at the cost of training time and poor interpretability. Fast learning algorithms were a prerequisite for the *L2R* challenge. We achieved fast learning by using semantically sufficient state encoding and searching for safe behaviors in the replay buffer. See section 6 for limitations of our approach.

4. Methods

4.1. U-Net

We trained the *U-Net* model (Ronneberger et al., 2015) to convert RGB images to road masks during the training phase on segmentation masks of car hood, sky, grass, road provided by the environment. To make the *U-Net* model generalize on unknown tracks, we used color-based image augmentation to augment the collected images using the *Albumentations* library (Buslaev et al., 2020). Specifically we used *ColorJitter (p=0.2)*, *Hue(p=0.2)*, *Saturation(p=0.2)*, *Contrast(p=0.2)*, *RGB shift(p=0.5)*, *Channel Shuffle(p=0.5)*, *CLAHE (p=0.2)*, and *Sepia(p=0.2)* for color transformations along with random horizontal flipping, scaling, shifting and rotating (p=0.5) for view transformations, where p indicates the probability of applying the transformation.

4.2. VAE

We trained a variational auto-encoder (*VAE*) (Kingma & Welling, 2013) to encode the road-mask images into two-dimentional latent encodings. The dataset for training of *VAE* was collected from runs in the environment in which the agent drives the car at constant speed in a straight line.

4.3. State representation pipeline

The data-flow pipeline is depicted in Fig. 1. The 384x512 RGB images provided by the *L2R* environment are cropped (lower and upper stripes) to remove the hood and the sky, producing a 100x512 image which contains the essential information for steering. This is resized to 64x64 and fed into the pre-trained *U-Net* model. The output of the *U-Net* model is a binary mask 64x64 of the road. The mask is resized to 28x28 and fed to the *VAE* to produce a 2-dimensional latent encoding. This encoding, along with the speed information from the environment, is used by the agent as the state representation for producing the action output.

4.4. Environment's reward function

The environment returns positive reward value for every time step proportional to the speed of the agent and negative reward value for driving off the road:
min(-1*25, -1*5*velocity)

4.5. Base behavioural policy and value function discriminator

The most frequently used action from the demonstration data set, namely the *drive straight* action, served as the agent's base behavioral policy. We collected a data set of training trajectories using this base behavioral policy (*drive straight*) for 20 episodes, obtaining of about 1200 states altogether. Depending on the initial heading angle and the distance to the edge at the beginning of each episode (fig. 2), this results in the agent driving along the road for some time steps and driving off the road at the end of each episode, thus completing the episode. Using the collected reward, the agent computed the discounted accumulated reward estimate for each state in the trajectories.

The value function is implemented as a five-nearest-neighbors classifier. If the average value of five-nearest-neighbors is positive, then the current state is *safe* and the agent continues to follow the base behavioural policy, *drive straight*. If the average value is negative, then the current state is *unsafe*, and the agent follows the correction policy (see section 4.6).

We also experimented with training a convolution neural network for predicting the value estimation of RGB-image input, assuming the base behavioural policy the agent drives straight. This neural network model produced similar results.

4.6. Correction policy: action discriminator

Training of the correction policy occur through execution of random discrete actions whenever a state has become *unsafe*. A randomly selected action is executed until the state become *safe* or until the episode ends. If the selected action executed for some number of time steps lead to a *safe* state, these state-action pairs are labeled as *good* and saved in the correction policy buffer, if lead to the end of episode, then these state-action pairs are discarded. The correction policy buffer was trained for another 20 episodes.

Exploitation of the correction policy works though the five-nearest-neighbors search in the correction policy replay buffer of *good* transitions. The discrete action selected by the majority of five-nearest-neighbors in the current *unsafe* state used by the agent to execute in the environment.

The correction policy is selected to control the agent until the value function discriminator classifies the current state as *unsafe*. After the current state becomes *safe* again, the agent selects the base behavioural policy to control the agent.

4.7. Evaluation

During the evaluation phase, the agent selects the base behavioural policy to control the agent in *safe* states and the correction policy in *unsafe* states. The value function determines whether the current state is *safe* or *unsafe*.

4.8. Speed adaptation

In the one hour given to the agent to adapt to the competition track, the agent gradually increased its speed in episodes to find the maximum speed level at which it would not go off the road. It searched for the optimal speed in *safe* and *unsafe* states. The agent had an internal model of the distance from the beginning of the episode to the current time step, which can be approximated by the number of time steps and the speed at each time step. In this internal model, the speed for positive and negative episodes were accumulated and then an upper bound on the *safe* speed for a given distance segment from the starting position of the track was estimated. In training runs, the agent increases speed for each distance segment it survived in the previous run and decreases it for distance segments where it did not survive in previous runs. We found 15 runs to be enough for the purpose of speed adaptation.

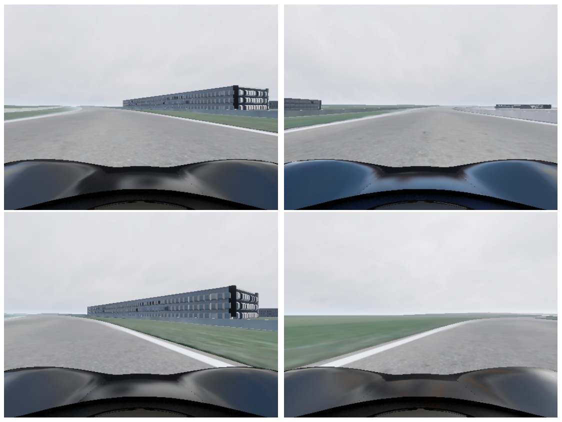
Figure 2. The first row - *safe* state examples. The second row - *unsafe* state examples.

Track	Avg. speed (km/h)
Thruxton (known)	**112.43 ± 8.8**
Anglesey National (known)	**90.34 ± 8.3**
Las Vegas (unknown)	**73.19**

Table 1. Performance comparison on different tracks
Only one data point is available for *Las Vegas* track since we do not have access to the local version of the track. All other readings are an average of 5 evaluations, with 3 laps each.

5. Results

While testing, the agent maintains the maximum *safe* speed for a given distance segment according to the learned internal model. Thus the agent achieved the average speed of 73 km/h on the unknown track and to 112 km/h on the known track.

The average speed achieved shown in Table 1. Due to limited access to *Las Vegas* track, only leaderboard scores (AICrowd, 2022a) are reported. All the experiments achieved 100% success rate in all the tracks.

6. Discussion

In this paper we present our solution which matches latent space representation of states to control the car and optimizes the mean speed. One drawback of our solution is that it is only able to predict on a short horizon, which does not allow human like performance by choosing the optimal curved path on turns. Another drawback of out solution is that it works well for low dimensional noise-free state representations, where KNN lookup is feasible. Alternatively, control tasks with rich sensory information can be solved with deep reinforcement learning techniques (Melnik et al., 2021)(Bach et al., 2020a) by learning a policy (Konen et al., 2019)(Schilling & Melnik, 2018)(Schilling et al., 2021). In the course of the competition, we tried out various other RL techniques (Bach et al., 2020b) like SAC, DQN etc., but they turned out either very expensive to train or resulted in low average speeds due to zig-zag driving. There is a potential to extend our solution by using Monte Carlo Tree Search to evaluate different paths (Harter et al., 2020), and modularization of the task for efficient learning (Melnik et al., 2019), while maintaining safety constraints.

We would like to thank the organizing team (Herman et al., 2021) for the technical support provided throughout the competition.

References

AICrowd. Learn-to-race autonomous racing virtual challenge, 2022a. URL https://learn-to-race.org/2021/12/07/challenge/.

AICrowd. Learn-to-race: Autonomous racing virtual challenge announcement. congratulations to the winners!, 2022b. URL https://discourse.aicrowd.com/t/congratulations-to-the-winners/7658.

Azizpour, M., da Roza, F., and Bajcinca, N. End-to-end autonomous driving controller using semantic segmentation and variational autoencoder. In *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)*, volume 1, pp. 1075–1080, 2020. doi: 10.1109/CoDIT49905.2020.9263921.

Bach, N., Melnik, A., Rosetto, F., and Ritter, H. An error-based addressing architecture for dynamic model learning. In *International Conference on Machine Learning, Optimization, and Data Science*, pp. 617–630. Springer, 2020a.

Bach, N., Melnik, A., Schilling, M., Korthals, T., and Ritter, H. Learn to move through a combination of policy gradient algorithms: Ddpg, d4pg, and td3. In *International Conference on Machine Learning, Optimization, and Data Science*, pp. 631–644. Springer, 2020b.

Beohar, S. Learn-to-race uniteam solution, 2022. URL https://gitlab.aicrowd.com/shivansh_beohar/l2r.

Beohar, S. and Melnik, A. Planning with rl and episodic-memory behavioral priors. *arXiv preprint*, 2022.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., and Kalinin, A. A. Albumentations: Fast and flexible image augmentations. *Information*, 11 (2), 2020. ISSN 2078-2489. doi: 10.3390/info11020125. URL https://www.mdpi.com/2078-2489/11/2/125.

Harter, A., Melnik, A., Kumar, G., Agarwal, D., Garg, A., and Ritter, H. Solving physics puzzles by reasoning about paths. *arXiv preprint arXiv:2011.07357*, 2020.

Herman, J., Francis, J., Ganju, S., Chen, B., Koul, A., Gupta, A., Skabelkin, A., Zhukov, I., Kumskoy, M., and Nyberg, E. Learn-to-race: A multimodal control environment for autonomous racing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9793–9802, 2021.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Konen, K., Korthals, T., Melnik, A., and Schilling, M. Biologically-inspired deep reinforcement learning of modular control for a six-legged robot. In *2019 IEEE international conference on robotics and automation workshop on learning legged locomotion workshop,(ICRA) 2019, Montreal, CA, May 20-25, 2019*, 2019.

Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *CoRR*, 09 2015.

Melnik, A., Fleer, S., Schilling, M., and Ritter, H. Modularization of end-to-end learning: Case study in arcade games. *arXiv preprint arXiv:1901.09895*, 2019.

Melnik, A., Harter, A., Limberg, C., Rana, K., Sünderhauf, N., and Ritter, H. Critic guided segmentation of rewarding objects in first-person views. In *German Conference on Artificial Intelligence (Künstliche Intelligenz)*, pp. 338–348. Springer, 2021.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Schilling, M. and Melnik, A. An approach to hierarchical deep reinforcement learning for a decentralized walking control architecture. In *Biologically Inspired Cognitive Architectures Meeting*, pp. 272–282. Springer, 2018.

Schilling, M., Melnik, A., Ohl, F. W., Ritter, H. J., and Hammer, B. Decentralized control and local information for robust and adaptive decentralized deep reinforcement learning. *Neural Networks*, 144:699–725, 2021.