Formula-RL: Competitive Self-driving Cars

Zongnan Bao
University of California, Los Angeles
zb3@g.ucla.edu

Ruoyu He
University of California, Los Angeles
rhe9527@g.ucla.edu

Yiming Shi
University of California, Los Angeles
syiming@g.ucla.edu

Tingfeng Xia
University of California, Los Angeles
tingfengx@cs.ucla.edu

Xinyu Zhao
University of California, Los Angeles
xinyuz9611@g.ucla.edu

***Abstract*—With the development of representation learning, the area of reinforcement learning has become a powerful learning framework now capable of learning complex policies in high dimensional environments. We aim to create an artificial kart racing experience where contestants will train their own reinforcement learning-based kart driving models and compete with each other. The 3D racing track is designed and implemented by Unity3D platform with ML-Agents Toolkits. We have proved and compared the effectiveness of reward function and different components, including direction, speed, crash avoidance in kart racing games. Moreover, we have designed a new reward function that mitigate the bias of between human intuition and kart's actual behaviors.**

***Keywords*—Reinforcement Learning, Computer Graphics**

I. Introduction

A. Background and Motivation

Competitive coding games, such as CodeCombat and Lia [1], [2] has attracted a lot of attention in the past years. In these games, users generally write a segment of code to control one or more agents inside an environment, interacting with other agents alike. These games are widely accepted among people who wish to practice coding and abstract problem solving skills as they make the practice much less dull. In the recent years, the Machine Learning (ML) community has bloomed thanks to the advances in computation power and Kaggle has raised as a popular platform for ML competitions. In these competitions, the performance of the models are generally measured using one or more quantitative metrics. However, the usually overlooked qualitative evaluations also provide insights into the robustness as well as reasonableness of the model's performance, particularly with Reinforcement Learning (RL) methods.

B. Problem Statement

We focus on the niche RL-based self driving car competition. We create an artificial kart racing experience where contestants will train their own reinforcement learning-based kart driving models and compete with each other. In the competitive programming regard, instead of letting the contestants write code to control the agents (karts in our case) directly, we ask them to supply Reinforcement Learning (RL) based control models and let different models compete with each other.

Upon this built platform, we compare different RL formulations for self-driving cars. Firstly, we have tested the effectiveness of reward functions. We set the reward function with checkpoints passing rewards and hit collision rewards as our baseline and compare this baseline with no reward condition. Secondly, we separately tried different combinations of signals, including driving speed only condition, driving direction only condition and driving speed and driving direction together condition to test the different effect of signal on our RL process. Thirdly, in order to mitigate the bias of human intuition for kart behavior and its actual behavior, we have redesigned the reward function for driving speed signals and implemented it with our baseline signals.

II. Implementation

A. Platform

We use Unity 3D with the Karting Micro-game Learning template [4] to develop our RL-based karting competition. This template provides a simulation environment for tracks and cars. On top of the karting environment, we use the Unity ML-Agents Toolkit [5] for the training of autonomous driving models. ML-Agents Toolkit also provides a backbone for environment-agnostic training methods [3], which is a feature that we intend to build on.

We redesigned the game flow to include a new model selection screen at the start of the game, where the contestants will enter the game with their own model (an onnx file). The selected models will be used to run the karts in the simulated race. Upon the first kart completing all the objectives, a result screen will be pushed to indicate whether you (player one) lost or won the game.

B. Reinforcement Learning

Reinforcement Learning (RL) is a method that leads agents to learn behavior from train-and-error interactions with a dynamic environment. There are two main strategies for solving reinforcement-learning problems: searching in the space of behaviors to find an environment suitable solution and using statistical techniques and dynamic programming methods to estimate the utility of taking actions in states of the world.

[9] The second strategy is properly adaptive to our scenario or kart racing competitions which have various states of environments and models need to instantaneously decide the direction of next move according to different conditions and past experiences.

In order to learn from past experience, we need to set up a reward function that provides a numerical score based on the state of the environment which specifies the intrinsic desirability of that state. The training purpose of agents is to discover a policy that maximizes such numerical scores which are called reward. In our case, we aim to train the kart agents to learn to drive in complicated racing tracks and make appropriate decisions facing various environments such as obstacles, forks and different slopes in the road. In order to achieve this goal, we provide kart agencies with different reward signals, including driving speed, driving direction, collision with obstacles and checkpoint passing conditions, to help them learn the behavior of completing a tap in the shortest time during training. The total reward that the agent will learn to maximize can be a mix of extrinsic and intrinsic reward signals.

Formally, our training procedure is as follows: we start by simulating the environment for many trials where the car, over time, learns what is the optimal action to take for every observation it measures by maximizing its future reward. Such learned behavior is called a policy, which is essentially a mapping from observations to actions. ML-Agents in Unity provides an implementation of two reinforcement learning algorithms:

- Proximal Policy Optimization (PPO)
- Soft Actor-Critic (SAC)

We use the general PPO since it has proven itself to be much more stable at convergence when compared to many other RL algorithm counterparts. Specifically, PPO strikes a balance between ease of implementation, sample complexity, and ease of tuning, via computing update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small.

III. Experiment Design and Result

A. Track Design

We designed two tracks for models to compete. The first track is named "Mountain Top", in which we will test model's ability of driving in the narrow lane, turning in sharp circles, etc. We also aim to test the model's predictability for future road situations by dividing the track into two branches, and models with future predictability will be able to choose a shorter and easier path. The second track is named "Cyber Spiral City", in which we will test the model's decision making abilities. In the second track, the model will pass an intersection where it also needs to follow the correct checkpoint order. We believe both tracks are important for testing driving abilities of an autonomous driving model.

B. Reward Design & Experiments

Reward design impact hugely on how the model behaves. We incrementally tested how reward functions are impacting the model's behaviors, and did ablation studies to find out the most important factors in our reward design. We first setup objectives that we want the model to achieve: 1) Correctness: pass checkpoints in the correct order, 2) Speed: finish a lap as fast as possible, 3) Crash Avoidance: reduce crashes as much as possible, 4) Direction: should always heading towards next checkpoint.

1) Performance Evaluation: We evaluate the performance of each reward design from 3 perspectives: 1) Time per lap averaged under 30 laps, 2) Crashes per lap averaged under 30 laps, 3) Subjective comments as we observe the model's behavior.

2) Reward Components: We created reward building blocks to achieve each objective as we explained them in Fig. 1 in detail. We also designed a new reward scheme (Improved Speed) from another perspective according to [6], in which it states that it's better to design reward function based on the actual objective explicitly instead of based on what human thinks it should behave. For instance, if we want to achieve the speed objective, we actually want the model to run from one checkpoint to the next in the shortest time possible, thus we should reward the model with that objective directly, i.e.

$$\text{reward} = \frac{\text{distance between checkpoints}}{\text{time elapsed between checkpoints}}$$

instead of what we used to reward based on the local speed. Noting that since this improved speed reward also considers correctness, we merged Correctness and Speed reward into one reward component named Improved Speed.

3) Effects of Direction Component: From the experiment results, we conclude that one important reward component that has large impact on the performance of the model is the Direction. More general speaking, the model needs a guidance when it's driving between two checkpoints. As you can see from experiments 1 and 2 from Fig. 1, without Direction reward, the model performs nearly as bad as receiving no rewards, and we observe that the reward it gets converges from a negative number to 0. We speculate that since poorly-performed model will have low chance passing through checkpoints, the model barely gets reward during training.

4) Effects of Speed Component: The Speed components also plays an important part in model's performance. Experiment 2 and 4 from Fig. 1 shows that adding Speed components improves the average time per lap, but it also increased average number of crashes per lap. This conclusion is reasonable since the model runs faster thus it will have higher probably running into walls.

5) Experiment Setup: We trained 1,000,000 steps for each model in our "Mountain Top" track. During training phase, the model will be reset to random checkpoint after it collides with walls (to prevent model stucking at the wall).

#	Reward Components	Avg Time per Lap	Avg Crash per Lap
0	None	N/A	N/A
1	Correctness + Crash Avoidance	N/A	N/A
2	Correctness + Crash Avoidance + Direction	00:01:57:90	313
3	Correctness + Crash Avoidance + Speed	N/A	N/A
4	Correctness + Crash Avoidance + Speed + Direction	00:01:26:45	600
5	Crash Avoidance + Improved Speed + Direction	00:01:29:47	588

Fig. 1. **Reward Design Experiments**: N/A means the model performs too badly and can never finish one lap.

Reward Components	Explanation
None	No reward or penalty is given to the model.
Correctness	+1 for passing checkpoints in the correct order.
Crash Avoidance	-1 for hitting the wall.
Direction	Dot product of model's velocity and direction towards next checkpoint. Positive if towards checkpoint, negative if backwards checkpoint.
Speed	Local Speed (scalar value)
Improved Speed	(distance between checkpoints)/(time elapsed between checkpoints)

Fig. 2. **Reward Detail**

IV. Future Works

According to experimental results and observation, we demonstrate there are main directions that can implement improvement on reward function. The first problem we need to solve is the "hit and drive back" condition that an agent sometimes turns to the opposite direction after hitting an obstacle. "Hit and drive back" condition affects our training result and model converging time to a large extent. When it happens, the time an agency completes a tap dramatically increases and leads the experiment to an unexpected way.We want to design a reward function that can help solve the "hit and drive back" condition that allows all the agencies to always drive towards the right direction which is obviously true in real scenarios but causes problems for reward policies.

The other possible improvement of reward function is about expected future behavior. Currently, our reward function only receives the real-time sensor's results and considers the effect of instantaneous action. In the future, we want to add rewards for future conditions to help make current decisions. For instance, while the sensor detects there are obstacles or forks in the road later, our reward function can involve this future element to the current decision and help our cart adjusting its directions in advance to avoid collision.

Moreover, we want to incorporate Imitation Learning into our model in the future. Imitation learning techniques aim to mimic human behavior in a given task. An agent (a learning machine) is trained to perform a task from demonstrations by learning a mapping between observations and actions. [8] In our task, imitation learning can help the reinforcement learning rapidly study the driving behavior in the correct direction and reduce the time agent taking to solve the environment such as obstacles and complicated topography. We want to compare the training time and experimental results with and without the imitation learning in order to figure out the effectiveness of imitation learning in reinforcement learning tasks.

V. Conclusion

In summary, we have designed two different kart racing tracks, tested the effectiveness and influence of signals including driving speed, driving direction, checkpoint condition, collision condition and improved the computation method of driving speed to reduce bias. From the observation of the training process and experimental results, we discover that the direction component plays the most important role which decides whether the trained kart behavior is eligible. Moreover, the main influence of driving speed is to improve the average lap time. Our self-designed reward function also helps improve the reasonability of the speed component. In the future, we can still improve the reward function by solving "hit and drive back" conditions, involving future environment detection and implementing imitation learning during the RL process.

References

[1] CodeCombat. Accessed June 18, 2022. URL: https://codecombat.com/

[2] Lia - Competitive Coding Game. Accessed June 18, 2022. URL: https://www.liagame.com/

[3] Traning ML-Agents. Accessed June 18, 2022. URL: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-ML-Agents.md

[4] Karting Microgame. Accessed June 18, 2022. URL: https://assetstore.unity.com/packages/templates/karting-microgame-150956

[5] Juliani, A., Berges, V., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., Lange, D. (2020). Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627. https://github.com/Unity-Technologies/ml-agents.

[6] Knox, W. Bradley and Allievi, Alessandro and Banzhaf, Holger and Schmitt, Felix and Stone, Peter (2021). Reward (Mis)design for Autonomous Driving. https://arxiv.org/abs/2104.13906
[7] Kiran, B. Ravi, et al. "Deep reinforcement learning for autonomous driving: A survey." IEEE Transactions on Intelligent Transportation Systems (2021).
[8] Hussein, Ahmed, et al. "Imitation learning: A survey of learning methods." ACM Computing Surveys (CSUR) 50.2 (2017): 1-35.
[9] Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." Journal of artificial intelligence research 4 (1996): 237-285.