Reinforcement Learning with AWS DeepRacer

Exploring the Potential of Reinforcement Learning in Autonomous Driving

By: Tristan Dewing

Stats 199
Professor Miles Chen
May 27, 2022

I. Introduction/Background

In the current age of technology, engineers are constantly looking for new ways to automate technology and human processes. Machine learning and artificial intelligence have been leading the charge as of late in reducing our own input and intervention in our daily lives. One area of automation that machine learning is being applied to is in autonomous driving or self-driving vehicles, in which a vehicle by sensing its environment is able to drive on its own without human assistance or control. Autonomous driving provides an alternative solution to preventing vehicle accidents and reducing emission of carbon dioxide ("Automated Vehicles"). Supervised learning has been used in the past with self-driving cars, but the use of labeled training data inherently requires human bias that autonomous driving is supposed to avoid (Chopra). Recently, one method has come to the forefront as a new potential solution to cars driving without needing humans behind the wheel: reinforcement learning.

Reinforcement learning is a type of machine learning that unlike supervised learning does not use labeled training data. Instead, reinforcement learning involves an intelligent agent that performs specific tasks within an environment. The agent is then either rewarded or punished via a "reward function" based on how well it is able to perform its tasks. In a given situation or "environment state", the agent will perform an action from a set of possible choices called an "action space" based on a strategy called a "policy", which will either help or hinder it in completing its tasks ("Reinforcement Learning"). If the agent performs well, it will be rewarded through the reward function, which will make it more inclined to perform similar actions in the future and thus make it more successful at performing its tasks. Conversely, if the agent performs poorly, the reward function will penalize the agent so that it is less inclined to perform similar actions in the future. The agent works to take actions that will maximize its total reward, and it is through this system of rewards and penalties that the agent learns to successfully perform tasks within its environment (Verma).

To test the effectiveness and viability of reinforcement learning in autonomous driving, we chose Amazon Web Service's (AWS) DeepRacer vehicle to be our agent. AWS DeepRacer is "an autonomous 1/18th scale race car designed to test RL models by racing on a physical track" ("AWS DeepRacer"). The car uses a camera to view the environment of the track and a reinforcement learning model to guide its speed and direction (Mwiti). Through training the miniature race car with a customizable reward function, the car learns to properly drive on a track and stay on course. AWS DeepRacer allows students to construct their own tracks for their cars to drive on and provides an online portal to train their models with. The program also provides students and educators the opportunity to

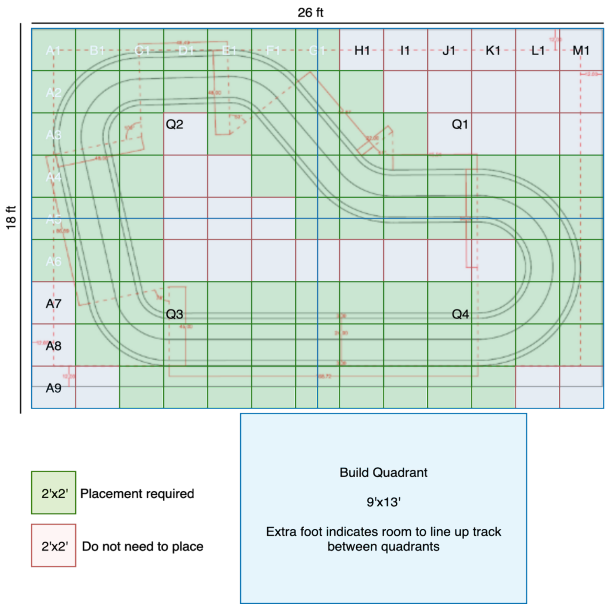
put their DeepRacer cars to the test and race them against other DeepRacers around the world in real-life competitions.

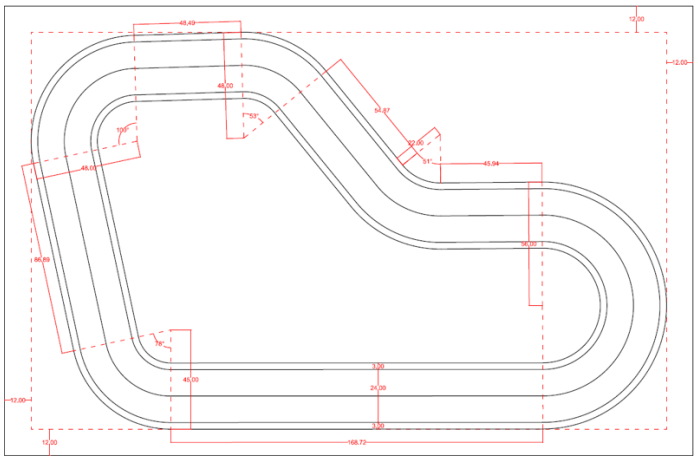
Our goal for this project was to train a reinforcement learning model that would allow the AWS DeepRacer car to successfully perform multiple laps around a track while staying within the driving lane at all times. We also hoped by customizing our reward functions to learn more about what goes into creating effective reward functions in reinforcement learning. Ultimately, we wanted to test the potential of using reinforcement learning for autonomous driving on a small scale, which may have implications for autonomous driving with standard-sized vehicles.

II. Methodology

First, we constructed a track for the AWS DeepRacer car to drive on. The AWS DeepRacer program comes with a selection of different track layouts that students can select from and recreate. We chose to recreate a track called “re:Invent 2018” as shown in Figure 1 below using 24” x 24” x 0.5” foam interlocking tiles as well as white and yellow duck tape (ARCC). The re:Invent 2018 track was approximately 26’ long and 18’ wide and consisted of 82 foam tiles (9 rows and 13 columns). The tiles used to create the track are shaded green in Figure 1 below. After connecting all the required tiles, we created the inner and outer boundaries of the track by sticking white duck tape to the tiles, which represented the boundaries of the track. Finally, we used repeating 3.5” short pieces of yellow duck tape to make a dashed centerline for the track.

Figure 1: re:Invent 2018 Race Track Dimensions (ARCC)





While we were constructing the track over a four week period, we trained our reinforcement learning models in the AWS DeepRacer student portal. AWS requires that you make several different specifications for the model, including choosing the track that the car will train on, the type of algorithm you will use, and the training time.

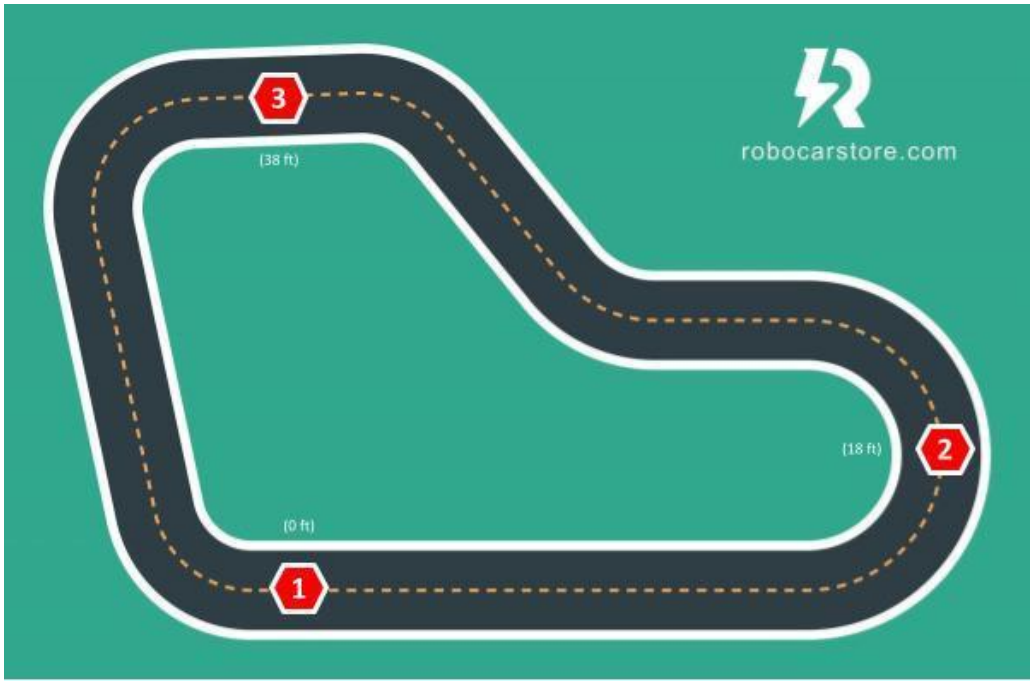
There were two main types of algorithms to choose from: PPO and SAC. Their main difference lies in their trade-off of exploring the environment and time exploiting the information from previous experiences to learn the optimal policy that will maximize reward (Guerrerio). PPO (Proximal Policy Optimization) uses on-policy learning, which prioritizes exploration in the current environment state and taking actions from the current policy. On the other hand, SAC (Soft Actor Critic), uses off-policy learning, which also includes exploration, but also balances the agent being able to exploit observations made by previous policies' exploration of the environment. In addition, PPO uses entropy regularization, which incentivizes the agent to choose actions with higher entropy (uncertainty) while SAC adds entropy to the maximization objective, which discourages using policies that choose unpromising behavior ("Reinforcement Learning"). As a result, while PPO is more stable, SAC requires less data. We ended up using only PPO algorithms, as multiple AWS DeepRacer users reported issues with the SAC algorithm being able to converge.

Along with choosing between the PPO and SAC algorithms, AWS comes with a choice of three different pre-written reward functions: "follow the centerline", which sections the track into three reward zones and penalizes the car the further it is from the centerline, "stay within the borders", which rewards the car for keeping all four wheels within the borders of the lane, and "prevent zig-zag", which modifies the centerline reward function by limiting the range of the car's steering angle ("Reinforcement Learning"). However, it also gives students the ability to customize these reward functions by adjusting the reward values or adding different parameters from a built-in dictionary, one of which was a waypoints parameter which marked the car's location on the track that we used in one of our models to determine when the car should pick up speed or slow down.

The model is then trained in a virtual environment where the car drives on a simulated version of the track. Once the models finished training, they were tested on a featured track that AWS selects each month, which is evaluated by giving the total time it takes for the car to complete 3 full laps on a given track as well as the model's place in the global standings for the month. After training and testing, the models were uploaded as tar.gz files into the car's compute module through a USB flash drive. For our final test on the real-life track, we trained 6 different models with differing training times and customizations, the information for which can be seen in Table 1 below.

When it came to testing the models, we tried to control for two potential confounding factors: the speed and the starting location of the car. For the speed, we noticed that the car had trouble controlling itself at speeds above 50% of its maximum speed, though we wanted the car to go at a faster pace than a crawl. We decided to set the speed for all test runs at a constant speed of 38% which allowed us to control for speed while allowing the car to drive at a moderate speed that would give it the best chance to succeed. We also noted that the car could perform differently depending on its starting location. To work around this, we broke up our experiment into three different blocks in which each model would be tested starting at 3 different waypoints of the track as shown below in Figure 2.

Figure 2: Starting Markers for re:Invent2018 Track. (RoboCar Store)



We evaluated our models using a metric we called %PTC, or percent of track completed, in which by using marked waypoints around the track we measured how much of the track the car completed before all four wheels went completely off the track. To account for the variation in results, we ran each model five different times at each of the three waypoints and took the average %PTC of the remaining three runs after the smallest and largest %PTC values were taken out. We also calculated the grand average %PTC for each model, averaging the results for each of the three waypoints.

III. Results

Table 1 below shows the six models we trained for the final test run on the physical track. In the virtual environment, each of the models were trained on the re:Invent 2018 track which we replicated for our real-life track, but they were all tested on a track called "Ace Speedway" where they were timed for completing three laps. The models vary in training time and the different customizations we made, both of which are listed in Table 1.

Table 1: Summary of models on virtual track (sorted by "3 Lap Time").

Model Name	**Algorithm**	**Reward Function**	**Train Time (minutes)**	**Test Track**	**3 Lap Time (minutes)**
ppo-centerline-4hour-reward	PPO	Default Centerline ● Broke center into four sections (within 0.1, 0.25, 0.5, 0.75), given 1.8, 1.2, 0.5, 0.1 reward for each	240	Ace Speedway	03:00.448
ppo-zigzag-1hour-v2	PPO	Default Prevent Zig-zag ● Broke center into four sections (within 0.1, 0.25, 0.5, 0.75), given 1.2, 0.8, 0.4, 0.1 reward for each	64	Ace Speedway	03:07.873
keep-wheels-on-track	PPO	Default Centerline ● Added penalty for getting a wheel off the track	120	Ace Speedway	03:08.727
ppo-centerline-custom-3hour	PPO	Default Centerline ● Broke center into four sections (within 0.1, 0.25, 0.5, 0.75), given 1.2, 0.8, 0.4, 0.1 reward for each	180	Ace Speedway	03:11.290
ppo-centerline-custom-4hour	PPO	Default Centerline ● Broke center into four sections (within 0.1, 0.25, 0.5, 0.75), given 1.4, 0.9, 0.4, 0.1 reward for each	240	Ace Speedway	03:11.598
ppo-waypoints-2hour	PPO	Custom ● Added specific waypoints for when to go fast vs slow	120	Ace Speedway	03:16.271

Table 2 below shows the results of our final test runs of each model. The average %PTC for each starting waypoint is listed as well as the grand averages of the three different waypoint results for each model are included below.

Table 2: Summary of models on physical track.

Model Name	**3 Lap Time (minutes)**	**Waypoint 1 Average %PTC**	**Waypoint 2 Average %PTC**	**Waypoint 3 Average %PTC**	**Grand Average %PTC**
ppo-centerline-4hour-reward	03:00.448	29.31%	6.03%	9.20%	14.85%
ppo-zigzag-1hour-v2	03:07.873	25.29%	4.31%	9.20%	12.93%
keep-wheels-on-track	03:08.727	36.78%	47.13%	14.94%	32.95%
ppo-centerline-custom-3hour	03:11.290	31.32%	12.07%	13.79%	19.06%
ppo-centerline-custom-4hour	03:11.598	7.76%	6.03%	12.93%	8.91%
ppo-waypoints-2hour	03:16.271	28.45%	20.69%	12.93%	20.69%

IV. Discussion

As shown in Table 2, the keep-wheels-on-track model performed the best, achieving the highest average %PTC for all three waypoints and thus having the highest grand average %PTC of the six models at 32.95%. The next best models were ppo-waypoints-2hour with a %PTC of 20.69% and ppo-centerline-custom-3hour with a %PTC of 19.06%. The worst model was the ppo-centerline-custom-4hour with a %PTC of just 8.91%.

To explain why certain models performed better than others, we can look to their differing customizations and training times. The best model, keep-wheels-on-track, was a customized version of the default centerline model that used a built-in parameter called "all_wheels_on_track". This parameter checked whether or not the car had all four of its wheels between the borders of the track. Along with being rewarded or penalized based on its distance from the centerline, the car would be rewarded for keeping all four of its wheels on the track while being penalized if any one of its wheels went off the track. The second best model, ppo-waypoints-2hour, used the "waypoints" parameter to denote specific waypoints on the track where the car should drive faster, such as along the straightaways, or slower, such as on the curves. The rest of the models did not use custom parameters but instead simply had the reward values from the default reward functions slightly adjusted to give both stronger rewards and stronger penalties.

At the start, we assumed that the models would be more successful if given more training time. However, contrary to our previous assumption, models that were trained for less time actually tended to perform better than those that were trained for more time. The two best models were only trained for 2 hours while the worst model was trained for 4 hours. For an even more direct comparison, the ppo-centerline-custom-3hour model performed better than the ppo-centerline-custom-4hour model when there was only an hour difference in training time and slight differences between the reward values in their reward functions. Overall, we found that making use of the custom parameters rather than simply adjusting the reward values and using less training time seemed to lead to more successful models.

Yet, while the possible reasons why certain models performed better than others can be discussed further, it cannot be ignored that the results as a whole were not satisfactory considering the original goal was for the car to be able to complete multiple laps around the track. The best model on average was only able to complete a third of the track while the rest of the models struggled to even complete a fifth of the track before completely driving off course. There was only one instance when the car was able to complete an entire lap around the track, and not only was it during a practice run before we determined how to systematically test and record the performance of our models, but the car was also going at significantly less than 38% maximum speed.

There were several limitations and confounding factors that could have led to the car's poor performance. For one thing, our real-life track was always set up outdoors on a rooftop, meaning it was subject to whatever weather conditions took place on test days, from high temperatures to gusty winds. It is hard to know how much the weather conditions affected the car's performance, but it certainly never enhanced it. Had the

track been set up indoors in a classroom, any surrounding conditions that possibly impacted the car's ability to drive could have been controlled for.

Additionally, while the AWS DeepRacer Student portal is simple to understand and use, its high level nature may limit the full potential of the car. The training portal allows students to edit or customize the pre-written reward functions to an extent, but the fact that students are required to use one of the pre-written reward functions as opposed to coding one completely from scratch forces the car to drive using one of those few models. Based on the less than satisfactory results, it is worth questioning whether the car's poor performance can be linked to the limitations the portal puts on student input.

Perhaps the biggest impediment to the car's ability to perform was itself, even more so than the models. Throughout the course of the project, the car experienced many technical issues including failing to boot up to the point of requiring a factory reset, having trouble making turns, and struggling to control stay on track at anything above a moderate speed. Its biggest weakness of all though seemed to be replicating its performance in a virtual simulated environment in real life, as while it was able to routinely perform multiple complete laps around a virtual track, it could barely complete a fifth of the real physical track most of the time. Because of this, there was an element of randomness to the car's performance that made it difficult to attribute its ability to stay on track to the models rather than other confounding factors.

V. Conclusion

By the end of the project, with the exception of the miracle run mentioned before, the car had failed the original goal of being able to complete multiple laps around the track successfully. The models that performed the worst did not utilize custom parameters and tended to have shorter training times than the more successful models. Even the best model was only capable of completing a third of the track on average before driving off of it. However, it is difficult to say how much the unsuccessful runs can be attributed to the quality of the models, as it seemed no matter what model was used, the car's ability to drive successfully was even more impacted by extraneous factors such as the surrounding weather conditions, the lack of user input in model training, and the car's own technical issues along with its struggles with replicating its performance in the virtual environment in real life on the physical track.

Nevertheless, even with all of the car's mishaps, that is not to say that it completely failed, as it showed that there is room for further exploration and improvement. There were a few standout models that allowed us to learn a few factors that could make future models more successful, including allotting shorter training time as well as utilizing

custom parameters such as “all_wheels_on_track” and “waypoints”. It also showed that if we keep continuing to improve our models and control for confounding factors, the car has the capability to properly drive multiple laps around the track. What is most exciting is that there is the possibility of writing code for and training our own models and reward functions from scratch outside of the AWS DeepRacer Student portal on a local computer or using AWS SageMaker, a platform made for developers to build and train ML models (“Reinforcement Learning”).

VI. Recommendations

Had we done the project over again, we would have taken more time early on to figure out how to use the AWS DeepRacer Student portal and how we could customize our reward functions, as we did not find out about the custom parameters until halfway through the project. This was important as models with custom parameters produced improved results from the base models. We also did not create an organized experimental process to test out our models until late, which had we constructed earlier on we could have tested more models and yielded more conclusive results. However, now that we have a set experimental procedure to go off of, we can not only test more models in the future but also potentially replicate our results.

From here, the next step would be to use a local computer or AWS SageMaker to code and train our own models locally. This would give us more control over the structure of the reward functions and hopefully allow us to achieve more effective models. Being able to train our own models locally would also free us of the limitations that the AWS DeepRacer student portal places on user input. These solutions would require more money and resources, but could be worth the investment if they can yield better performances from the car. Another step would be to potentially move the track indoors. Due to limited room space and resources we had to set up and use our track outdoors, which left the car subject to some difficult weather conditions. However, if we can continue this project with the track being set up indoors, the car will be able to drive without the weather or other surrounding conditions holding it back.

The models we trained are definitely not ready for the racetrack in a competition, much less for our own track, but could potentially be improved using local training so that they can be ready. In terms of the effectiveness of reinforcement learning and autonomous driving versus that of humans driving, the results were not much better than if we drove the AWS DeepRacer car ourselves using its manual mode. Still, the times the car did succeed or improve in its performance gave us some promise that reinforcement learning can not only make autonomous driving possible on a small scale, but potentially on a larger scale than a miniature car driving on a makeshift track.

VII. References

ARCC. “Guide to Creating a Full Re:Invent 2018 Deepracer Track in 7 Steps.” Medium, Medium, 13 July 2019, https://medium.com/@autonomousracecarclub/guide-to-creating-a-full-re-invent-2018-deepracer-track-in-7-steps-979aff28a6f5.

"Automated Vehicles for Safety." NHTSA, United States Department of Transportation, www.nhtsa.gov/technology-innovation/automated-vehicles-safety.

“AWS Deepracer - Aws.amazon.com.” AWS, https://aws.amazon.com/deepracer/.

“AWS DeepRacer Standard Track.” RoboCar Store, https://www.robocarstore.com/products/aws-deepracer-standard-track.

Chopra R., Roy S.S. (2020) End-to-End Reinforcement Learning for Self-driving Car. In: Pati B., Panigrahi C., Buyya R., Li KC. (eds) Advanced Computing and Intelligent Engineering. Advances in Intelligent Systems and Computing, vol 1082. Springer, Singapore. https://doi.org/10.1007/978-981-15-1081-6_5.

Guerriero, Edoardo. “How Can I Handle Overfitting in Reinforcement Learning Problems?” Artificial Intelligence Stack Exchange, 9 Apr. 2020, https://ai.stackexchange.com/questions/20127/how-can-i-handle-overfitting-in-reinforcement-learning-problems.

Mwiti, Derrick. “10 Real-Life Applications of Reinforcement Learning.” Neptune.ai, 8 Nov. 2021, https://neptune.ai/blog/reinforcement-learning-applications.

“Reinforcement Learning in AWS Deepracer - Docs.aws.amazon.com.” AWS DeepRacer, 2019, https://docs.aws.amazon.com/deepracer/latest/developerguide/deepracer-how-it-works-overview-reinforcement-learning.html.

Verma, Piyush, and Stelios Diamantidis. “What Is Reinforcement Learning? – Overview of How It Works.” Synopsys, 27 Apr. 2021, https://www.synopsys.com/ai/what-is-reinforcement-learning.html#:~:text=Definition,environment%20to%20obtain%20maximum%20reward.