Improving Trackmania Reinforcement Learning Performance: A Comparison of Sophy and Trackmania AI

LAURENS NEINDERS, University of Twente, The Netherlands

Fig. 1. Trackmania, a challenging racing game

1 ABSTRACT

Virtual car racing is popular among millions of people in the world. Some developers have created software to drive racing cars automatically using artificial intelligence. Gamers can race against these autonomous cars. I am interested in car racing and I discovered a significant difference in the performance of two AI racing systems. These systems are Sophy AI and Tmrl-lidar, which are made for the racing games Gran Turismo Sport [11] and Trackmania[19] respectively. Tmrl-lidar was significantly slower than an average player, whereas Sophy was able to beat the best human racers.

In this study, the performance gap between Sophy AI and Tmrl-lidar is analyzed by literature research and an investigation of the AI systems. The most important difference is in the input data that the systems are getting. Tmrl-lidar is not getting sufficient information about the upcoming track as it is using a lidar system, which only can measure a limited distance ahead in corners. Sophy AI gets the borders of the track 6 seconds ahead. In this study a method is developed that gives Tmrl-lidar similar input data as Sophy AI.

The new input data resulted in an improved Tmrl-lidar lap time from 44.6s to 35.4s on a racing circuit made by the developers of Tmrl. Human players were invited to compete against the newly trained AI racing model. The new model got the 7th fastest time from 60 participants. It was ±1.6s slower than the world record holder, a professional player who took 33.8s to complete the track.

The result of the test showed that the new method for input data resulted in a significant improvement of performance of Tmrl-lidar.

Artificial Intelligence is a great tool for autonomous racing. This study showed the importance of the input data that is offered to the AI system. The old Tmrl-lidar could not perform better since the input data was constraining the performance. A chain is as strong as its weakest link.

Author's address: Laurens Neinders, l.j.neinders@student.utwente.nl, University of Twente, P.O. Box 217, Enschede, The Netherlands, 7500AE.

2 ACKNOWLEDGEMENT

Major thanks to my supervisor: Dr. Mannes Poel, the Data Science Track Chair, Dr. Estafanía. Talavera Martínez. Also major thanks to Yann Boeteiller, for support with Tmrl, Mika Kuijpers and .Bux for support with tmdojo and Max Kaye for the Archivist plug-in.

3 INTRODUCTION

Autonomous car racing refers to self-driving cars that can safely and efficiently navigate a racetrack at high speeds without human intervention. Development of such systems requires overcoming numerous technical hurdles, such as real-time perception, decision-making, and control in a dynamic and unpredictable environment. The ability to autonomously race a car can have implications beyond the race track, as experiences in this field can be useful for autonomous driving in general.

In 2021, the autonomous driving AI, "Sophy"[6] achieved a remarkable feat by outperforming the best human players in the racing game "Gran Turismo Sport". To achieve this performance it used the AI technique: reinforcement learning, specifically the Soft-Actor-Critic (SAC)[10] algorithm.

Another racing game, Trackmania, has also seen the development of AI systems. A group of developers have created a framework (Tmrl [3]) and multiple AI systems for Trackmania using SAC, the same reinforcement learning algorithm used for Sophy. In a French show, the group demonstrated their best current model, Tmrl-lidar, setting a time of around 45 seconds on a test track, which is a solid performance, but still a long way of the human record of ±34 seconds.

The purpose of this study is to identify key differences between Sophy and Tmrl-lidar and improve Tmrl-lidar based on those findings.

3.1 Research questions

To test whether it is possible to improve Tmrl-lidar based on Sophy AI, the following research questions need to be answered:

(1) What are the key distinctions between Sophy and Tmrl-lidar?
(2) What modifications can be transferred from Sophy to Tmrl-lidar?
(3) How do these changes affect Tmrl-lidar's performance?

The paper is structured as follows: Section 3 is about the methodology. Section 4 is a literature review on AI racing algorithms and the specific algorithm used in the two systems. In Section 5, key differences between Sophy AI and Tmrl-lidar are identified, it also formulates a hypothesis to explain the performance gap. Section 6 focuses on transferring these differences to Tmrl-lidar, new methods will be developed to accomplish this. Section 7 contains information about training of the AI. The results will be presented in Section 8. Section 9 is a discussion on the study and section 10 is the conclusion.

4 METHODOLOGY

The methodology of this research consists of four main steps: literature review/data collection, identifying key differences, transferring differences, and evaluation.

A literature review and data collection were conducted to gather information on AI training and reinforcement learning. Papers on reinforcement learning, SAC, and Sophy AI were analyzed, along with information from the Tmrl-github and Sophy-blog.

The focus of the literature study and the comparison between Sophy and Tmrl-lidar was to identify key differences that can explain the difference in performance. Key difference can be found in algorithm, reward function or input data. As both systems use the same algorithm (Soft Actor Critic) and similar reward functions an hypothesis was developed that differences in input data are the biggest cause of the difference in performance between the two systems.

To test the hypothesis, the input data of Tmrl-lidar is changed in such a way that it is comparable with input data of Sophy. Tmrl-lidar was retrained using the new input data until lap times did not show any more improvements.

The developed model was evaluated by comparison of the fastest lap against the baseline Tmrl-lidar and a group of 60 human players.

5 LITERATURE REVIEW

5.1 Approaches to autonomous racing

Autonomous racing is complex and has been tackled using various approaches. In this chapter, we will explore three different solutions to the autonomous racing AI problem: classical approaches, imitation learning, and reinforcement learning.

Classical approaches involve breaking down the problem into submodules that consist of perception, trajectory planning, and control. Model Predictive Control (MPC) is one of the most promising classical approaches for controlling high-speed autonomous vehicles. MPC has shown impressive results in controlling vehicles in the real world [20]. Another classical approach, Model Predictive Path Integral Control (MPPI) [21], is a more flexible approach that can be combined with complex cost formulations and neural network vehicle models. However, both MPC and MPPI have limitations, such as the lack of flexibility in the cost function design or the requirement of highly parallel computations.

Imitation learning is another approach that directly learns a mapping from observation to control action in a supervised way. This approach requires labeled data, which is typically provided by human experts. One of the first autonomous driving systems to use a neural network to follow a road is the Autonomous Land Vehicle in a Neural Network (ALVINN,1989) [13]). Similarly, Pan et al. [12] used imitation learning for off-road autonomous driving. This approach involves training a machine learning model using expert data, either from humans or algorithms, to learn a policy. By doing so, imitation learning has the potential to overcome the real-time constraint of traditional methods.

Reinforcement learning (RL) optimizes parameterized policies based on sampled trajectories, without the need to solve nonlinear optimizations online or rely on labeled training data. Several studies have demonstrated the success of using deep RL for end-to-end driving and racing [8] [15]. Recently, the Soft Actor-Critic (SAC) algorithm was used to develop a high-speed autonomous drifting system in simulation [4]. The SAC algorithm employs off-policy training, which plays a crucial role in reducing the high sample complexity that has previously limited the widespread adoption of deep RL methods in high-dimensional domains [7].

Each approach to autonomous racing has its own strengths and weaknesses. Classical approaches offer a modular and interpretable framework for autonomous racing, but they require highly parallel computations and lack flexibility in the cost function design. Imitation learning is a promising approach to overcome the strict real-time constraint of classical approaches, but its performance is limited by the quality of the training data. Reinforcement learning provides an approach that optimizes policies based on sampled trajectories and has shown impressive results in driving and racing tasks. However, it suffers from high sample complexity and the challenge of balancing exploration and exploitation.

5.2 Soft Actor Critic algorithm

Soft Actor-Critic (SAC) is a deep reinforcement learning algorithm that has shown impressive performance on a range of challenging control tasks. SAC aims to maximize the expected cumulative reward over a given time by learning a stochastic policy and value function. The policy is represented by a neural network that outputs the mean and standard deviation of a Gaussian distribution, from which actions are sampled. The value function estimates the expected cumulative reward starting from a given state, and it is used to update the policy via a critic loss. The critic loss is a mean squared error (MSE) between the estimated value function and the discounted cumulative reward obtained from the next state. The policy is updated by minimizing a combination of the actor loss, which encourages actions that maximize the expected cumulative reward, and an entropy term that encourages exploration and prevents the policy from being overly confident.

One of the main advantages of SAC is that it uses a soft value function update, which means that it uses a target entropy term to regularize the policy towards being more stochastic. This allows SAC to learn more robust policies that can handle uncertainty and variations in the environment. Additionally, SAC employs off-policy updates, which means that the policy can learn from past experiences collected from a replay buffer, leading to more efficient and stable learning.

SAC has been successfully applied to a range of challenging control tasks, such as robotic manipulation [10], quadrupedal locomotion [9], and autonomous driving [4].

SAC is a powerful deep reinforcement learning algorithm that has shown impressive performance in challenging control tasks. Its soft value function update and off-policy learning make it suitable for handling uncertainty and variations in the environment, and its success in autonomous driving demonstrates its potential for real-world applications. However, like other deep reinforcement learning algorithms, SAC can suffer from high sample complexity and the challenge of balancing exploration and exploitation, which may limit its scalability to larger and more complex domains [5].

6 COMPARISON

In this section, Sophy AI and Tmrl-lidar are compared

6.1 Sophy AI

Sophy AI is a racing AI for the game "Gran Turismo Sport" created by SONY in 2021. It was trained using reinforcement learning to drive on a number of track and car combinations and has managed to get performance on par or better than humans in these scenarios.

6.2 Tmrl

Tmrl (Trackmania Reinforcement learning) is a framework for reinforcement learning in the racing game "Trackmania". The developers have created 2 AI models for Trackmania, both of which use reinforcement learning. The difference between the two is that 1 uses images as input data and the other uses lidar distances. Since the lidar system is more developed, has more in common with Sophy and requires much less training time, we will just focus on that system for this paper.

6.3 Comparison: Sophy vs Tmrl-lidar

In this section we will collect all relevant information about the two systems: Sophy and Tmrl-lidar.

6.3.1 Lap times. The first objective is to prove that Sophy is, in fact, much better than Tmrl-lidar. To show this, we will compare lap times (the time it takes to drive from start to finish) between the two.

Since the two systems are built for different games, we cannot compare them directly. What we can do is compare each system with human performance in their respective games. This will still give a good indication of the strength of the AIs.

Sophy was tested in three different settings. Setting A and B on the same track, but using different cars, setting C on a more challenging track with the same car as setting A.

Lap times[sec]: Sophy vs humans			
	Sophy	**fastest human**	**median humans**
Setting A	1:15.913	1:16.602	1:22.300
Setting B	1:39.408	1:39.445	1:47.259
Setting C	2:07.701	2:07.319	2:13.980

Table 1. Lap time comparison of Sophy AI compared to humans in Gran Turismo in three different settings

Lap times[sec]: Tmrl-lidar vs humans			
	Tmrl-lidar	**fastest human**	**median humans**
Tmrl-test track	44.631	33.798	38.748

Table 2. Lap time comparison of Sophy AI compared to humans in Gran Turismo in three different settings

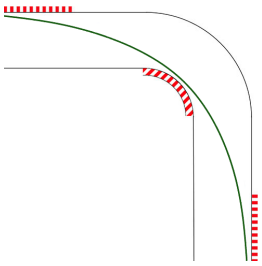

Fig. 2. Racing lines Sophy

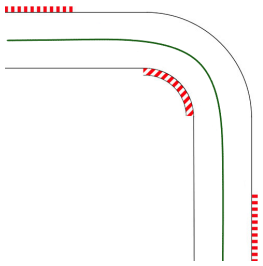

Fig. 3. Racing lines Tmrl-lidar

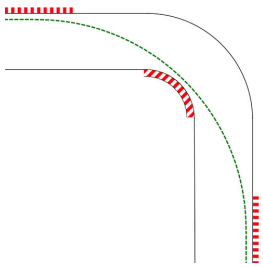

Fig. 4. The out-in-out approach

Table 1 compares the best lap times of Sophy AI with those of the best humans. Here you can see that Sophy managed to beat the best humans by a large margin in setting A. It was slightly ahead in setting B and was beaten in setting C.

Table 2 gives a comparison of lap times between human players and Tmrl-lidar on the test map for which Tmrl-lidar was trained. I shows that there is a significant gap between human performance and that of Tmrl-lidar.

6.3.2 Driving lines. When driving, a factor that has a great impact on lap times is: "driving lines", the way the car drives through corners. if you want to drive through a corner as fast as possible, an out

in out approach is often the fastest way. This approach is illustrated in figure 4.

When comparing the racing lines of the two systems, it is evident that Sophy follows the out-in-out approach a lot better than Tmrl-lidar. Tmrl-lidar has a more save approach to driving in general; it does not take much risk by trying to drive close to the walls but rather stays in the middle of the track.

6.3.3 AI Algorithms. Now that we know what both systems are capable of, the next step is to understand how they work. Both systems use an AI technique called Reinforcement Learning[1]. Reinforcement learning is a type of machine learning in which an agent learns to make decisions by interacting with its environment. The agent receives rewards or punishments based on its actions and uses this feedback to update its decision-making strategy. The goal of the agent is to maximize the total reward over time.

The specific reinforcement learning algorithm that is used is the same for both Tmrl-lidar and Sophy. This algorithm is Soft Actor-Critic[10] (SAC). One of the benefits of using SAC is that it can reduce the amount of data and resources needed for training by using a technique called off-policy training. This means it can learn from previously recorded actions and experiences, rather than needing to constantly collect new data. Additionally, SAC can be effective in high-dimensional environments, where there are many different variables that the system needs to take into account.

Since both systems use the same training algorithm, we assume that the main difference between the two will be in a different area.

6.3.4 Input data. The reinforcement learning algorithm receives a set of variables that represent the environment (the game). It has to make a decision on what output to give, for example: pressing forward, left, right, or break, which controls the car in the game. After this a reward will be given to the algorithm, based on how well it is driving. From this the algorithm will try to understand its environment and how to maximize the reward it will receive. The algorithm needs to know enough about its environment to make a solid decision that will give it the maximum reward.

We will now compare the input data (observation spaces) of Sophy and Tmrl-lidar.

Sophy uses the following observation space:

(1) Linear velocity
(2) Linear acceleration
(3) Euler angle between the car's rotation and the direction of the centerline of the track
(4) Distance measurements that measure the distance from the center point of the car to objects, such as the edge of the track.
(5) The previous steering command
(6) A binary flag indicating wall contact
(7) Curvature measurements in the forms of a series of discrete points of the sides and middle of the course ahead. Six seconds of track, based on the current speed of the car.

Tmrl-lidar uses uses the following observation space:

(1) Linear velocity forward
(2) 19 Distance measurements that measure the distance in pixels on camera, measured from the lower middle (50 pixels above the edge of the window) to the walls of the track, distributed across 180°.
(3) History of last 4 distance measurements.
(4) The previous steering command

The observed difference will be further analyzed in chapter 5.4.

6.3.5 Reward function. The reward function defines how well an AI agent is doing. For example, it will give a positive reward when the agent is driving in the correct direction and a negative reward when driving in the wrong direction.

A naive approach to creating a reward function would be to give a reward based on the speed of the car. However, this approach is not optimal, as the goal of the agent should not be to drive as fast as possible, but rather to complete the track as fast as possible. The AI will have to consider optimal trajectory and speed, which may imply slowing down for corners etc.

That is why both Sophy and Tmrl-lidar use a reward function that gives a reward based on progress on the track. The AI receives reward based on how much of the track has been covered since the last timestamp.

6.4 Identifying key differences

In this chapter, the main aspects of the two AI systems are discovered. In the AI algorithm and reward function, no notable differences were observed; there are minor differences in the exact implementation, but these do not seem likely to be the reason for the major difference in performance.

That is why we should have a better look at what the differences are between the systems in terms of input data. What we observed is that almost all of the variables from the observation space of Tmrl-lidar are also in the observation space of Sophy, with the exception of the history of 4 distance-measurement observation. However, Sophy has a few extra variables, namely: acceleration, Euler angle, binary flag for wall contact, and measurements of track ahead. Of these four, the one that catches the attention the most is "the measurements of the track ahead". This is because this information gives insight in what the upcoming track will look like, which could be very useful as the AI agent would be able to anticipate how the next corner will look and adjust its current driving to accommodate for this upcoming corner.

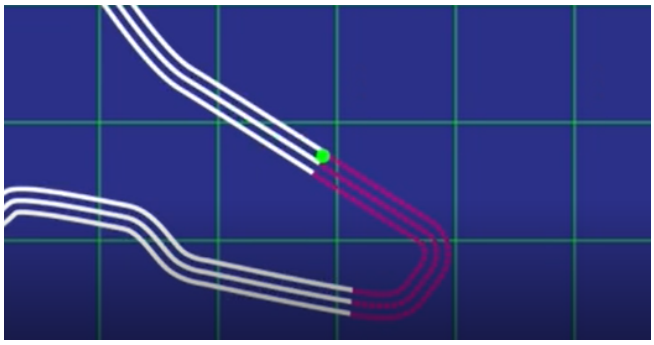

Fig. 5. Sophy has information of the track, 6 seconds ahead, based on the current speed

Figure 5 shows the part of the track that Sophy can see ahead. The green dot is the position of the car, the lines are the two sides and middle of the track and the purple piece is the part of the track that Sophy is given as input data.

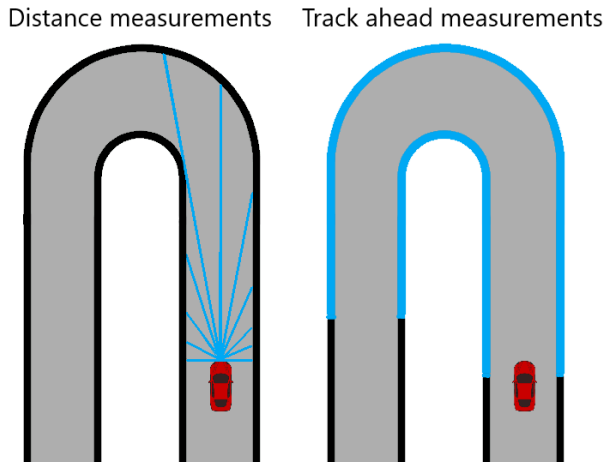

Fig. 6. Distance measurements only give information about a short piece of track ahead, whereas curvature measurements of the track ahead give more information to make a better decision.

Figure 6 gives a visual representation of the information that distance measurements provide (the data that Tmrl-lidar uses) compared to what the measurements of the track ahead provide (what Sophy uses). The figure shows that the distance sensors are only able to see the first part of the corner, which means that the AI cannot know how the track will look after this, see fig 7 as well.

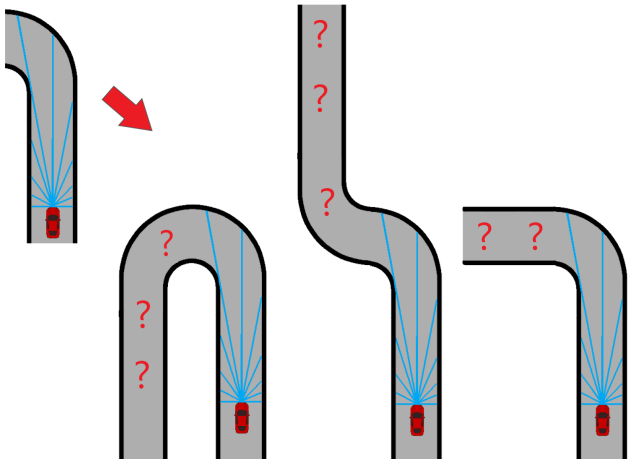

Fig. 7. Tmrl-lidar doesn't know what the next corner will look like.

The hypothesis that arises from these findings is that the key difference between Sophy and Tmrl-lidar is the lack of information of the track ahead as input data for Tmrl-lidar.

7 TRANSFER DIFFERENCES

To test the hypothesis of the previous chapter, a system should be implemented that can provide information about a piece of the track ahead of the car. The Sophy system uses a built-in feature in Gran Turismo Sport to get this information. However, it is not available in Trackmania. To test the hypothesis, a similar system to that of Sophy had to be made.

To generate this upcoming track data, a recording of the track had to be made by determining the coordinates of the walls.

7.1 Calculation of the track.

This paragraph explains the calculations for creating a representation of the track.

The goal is to create a representation of the track by driving on it once. This is just track recording, so driving can simply be done by a human although automation of this step could be interesting for the future. The data of the distance sensors (see figure 8) is used to calculate the position of the walls.

At first glance this seemed to be a simple calculation. One of the first results is shown in figure 10 where the dots simulate the walls. From the picture it is evident that the walls are not clear at all. Research showed that this was due to the following causes:

(1) The distance was measured under an angle (see figure 9);
(2) Some distances could not be measured because the walls were not on the screen. In figure 8 the wall for the horizontal distance sensors is not visible on the screen;
(3) The camera makes small movements and rotations that add to the playing immersion, but it also distorts the calculations. To combat this, the rotation and angle (roll and pitch) must be taken into account.
(4) Calculation of wall coordinates is dependent on more variables than originally taken into account.

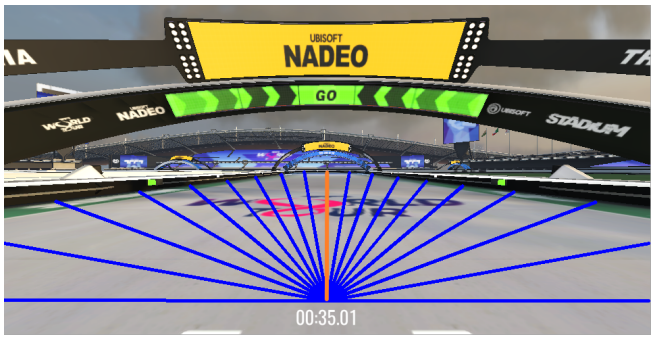

Fig. 8. The distance sensors are measurements from a point below the center of the image, to the walls of the track.

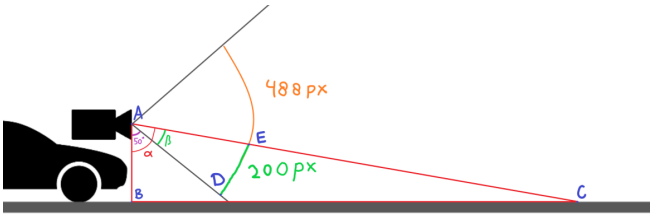

Fig. 9. Side view of car, goal is to calculate BC since that can be used to determine the coordinate of the end of a distance sensor

The problems with the map are solved solved with the following measures:

(1) Discarding some of the distance sensors;
(2) By measuring each wall separately on a short distance;
(3) By filtering out points that are clearly wrong;
(4) By drawing a mostly likely curve through the remaining points. This is done using a Bezier curve [2];
(5) By improving the mathematical calculation taking into account more variables (see appendix for more details calculation)

Figure 10 is one of the first attempts to create a coordinate map. The general shape of the track is visible, but there are many outliers and errors.

Figure 11 shows the map in a later development phase. The accuracy of calculations has improved and outliers are filtered out. There are still some inaccuracies that need to be addressed before it can be used.

Figure 12 is the final representation of the track using all the measures together.

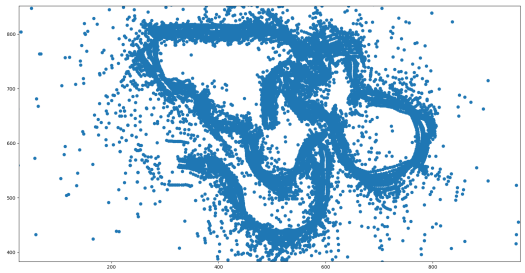

Fig. 10. The first try on programming a representation of the track. The map has a lot of noise and outliers, but the general shape is visible

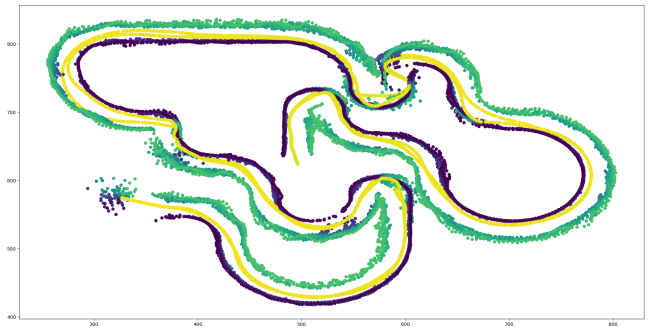

Fig. 11. This how the map looks later in the process, a lot of outliers are gone and both sides of the track are clearly distinguishable. The yellow line is the trajectory of the car, purple and green are the left and right side of the track respectively.

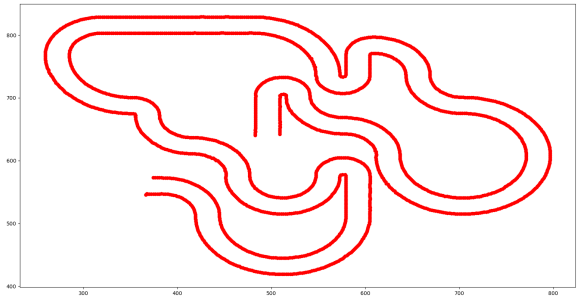

Fig. 12. This is the final version of the map, all outliers are removed and the points that make up the track are equally spaced out

7.2 Input for the AI

The final map (figure 12) looks like a continuous curve but, in fact, it consists of individual points. The final step before we can use the map to train an AI, is to cut out the part which is directly in front of the car. This is the part that the AI gets as input data at a given time.

The length of the input track is selected at 15 points ahead of the car. This means that the distance that the AI can see in front is fixed. This is different from the way Sophy AI is given data, as the input data for Sophy AI depends on the speed of the car. For our model a fixed distance was chosen because a variable distance would increase complexity of implementation.

The input data should be converted from absolute coordinates to coordinates in which the position of the car is taken into account. The second aspect that should be taken into account is the direction in which the car is driving. This direction is visualized by the blue line in figure 13. Figure 13 shows that the curve in the green box should be rotated by roughly 20 degrees clockwise. This conversion of the coordinates into relative and rotated coordinates is given as input data for the new Tmrl-lidar model.

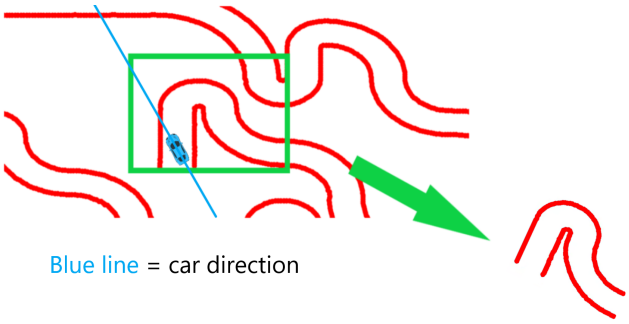

Fig. 13. A part of the track-map is cut out, normalized, rotated and given to the AI as input.

8 TRAINING

The old Tmrl-lidar could not cope with the new input data. Therefore the code of the AI system is changed to ignore the lidar measurements and to accept the new input data.

The total data (observation space) that the AI has access to is: current car speed, gear, rpm, track information, acceleration, previous steering command, steering angle and whether the tires are slipping.

The AI was trained at a frequency of 20 hz. Every timestamp (1/20th of a second) the AI generates an output (steering command) based on the total input data. After this, the AI will receive a reward based on the performance of the car during that timestamp. If the car made progress, the AI receives a positive reward. If it does not make progress, the reward will be 0.

9 RESULTS

This section contains the results of training the AI with the upcoming track as input. Here, the newly trained model is called trackmap AI.

9.1 Comparison lap times

This section compares the lap times of trackmap AI with those of Tmrl-lidar and human players. To obtain human lap times, the website Trackmania.io[18] was used. It stores and displays all records (the fastest lap time that a player managed to drive) for every Trackmania map. The map that was used for trackmap AI, Tmrl-test was played by around 20 people at the start of March 2023. To get a higher sample size and therefore a clearer picture of human performance, a request on the social media platform Reddit[14] was posted asking people to try to beat the best time of the AI. The number of records increased from 20 to 60.

Figure 14 shows in blue the human lap times on the track, in yellow the Tmrl-lidar lap time, and in red the trackmap AI lap time. It shows that trackmap AI managed to get significantly closer to the best human players' performance than Tmrl-lidar was able to do.

Tmrl-test track, Lap times[sec]: Tmrl-lidar vs. trackmap AI				
track	**Tmrl-lidar**	**trackmap AI**	**Best Human**	**Average Human**
Fastest lap	44.631	35.465	33.798	38.748

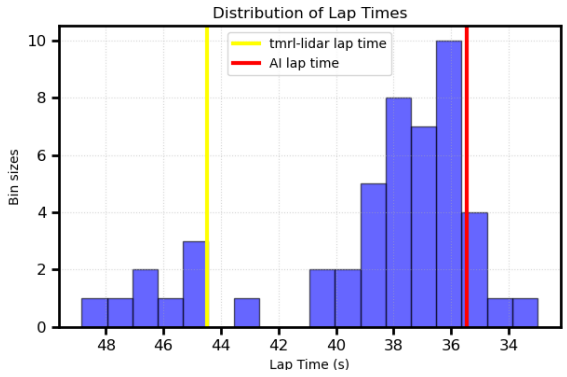

Fig. 14. Comparison of all human lap times (blue bars) and trackmap AI (red) and Tmrl-lidar (yellow) on the Tmrl-test track

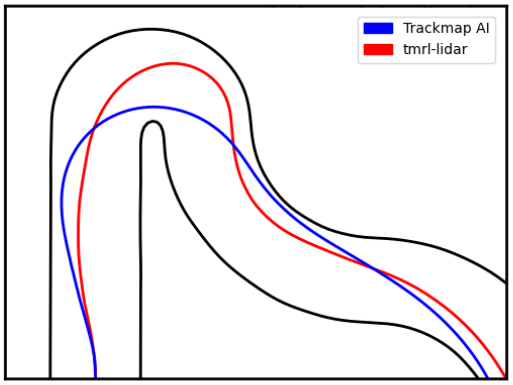

Fig. 15. Comparison of racing lines between Trackmap AI (new model)in blue and Tmrl-lidar in red

9.2 Racing lines

The trained model learned to drive much more efficient and therefore faster racing lines than the old model did. Figure 15 shows in blue the way the newly trained model drove and in red how Tmrl-lidar drove through this corner. It shows that the new model managed to make much better use of the available space on the track (less sharp turns).

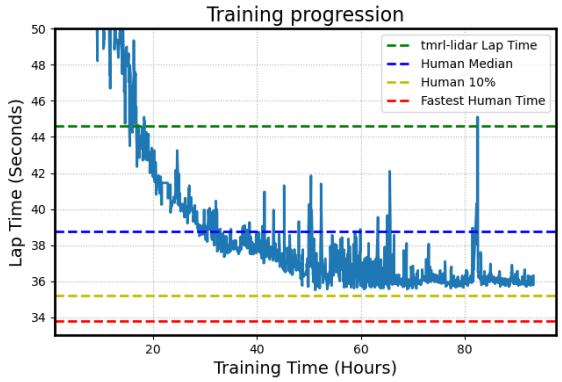

Fig. 16. The training progress of trackmap AI. The blue line shows the lap time of the AI at a certain point in time during training.

9.3 Training

During training, two types of data are generated: game data and training data. Game data is data such as car speed, car position and lap times. Training data is for example the amount of reward. For the game data, TMDojo [16] (a website and plugin that lets users save and playback runs) and Archivist (a plugin by XertroV [22] that saves a replay file for every run) were used. For training data, the Tmrl framework saves and stores all training data on WandB.io [17]. The training process was characterized by trial and error mostly due to specifics in programming and bugs in both newly implemented as well as existing code. During the project, several bugs in the Tmrl-framework were discovered and fixed. The final AI took 93 hours to train. Peak performance was reached after around 50 hours (see figure 16). After 50 hours the peak did not improve, but the AI did get more stable. The total training time during this research was around 700 hours.

10 DISCUSSION

In this chapter, the significance of the obtained results, the limitations of the study, and suggestions for future work are discussed.

10.1 Significance of obtained results

This study shows that AI (specifically SAC) is a great tool for autonomous car racing, but that performance depends greatly on the right input data. The analyzed AI, Tmrl-lidar, could not correctly predict the upcoming walls. The performance of AI can be compared to a chain which is as strong as its weakest link.

10.2 Limitations

While the study helped to gain insight in the importance of input data in reinforcement learning, the use case for the specific AI that was created is narrow. It only works in the Trackmania game, which is to be expected, but it also works only on very specific tracks. Tracks need to be flat with black borders all around it and with a normal road surface. This might not sound that specific, but these types of track are uncommon in Trackmania. Tracks in the game consist of a wide variety of track surfaces, height-differences, loops and boosters.

If you want to build an AI that works on all types of track, it would have to make use of a more advanced structure for input data that can capture all the details of Trackmania.

In the new model the lidar system in no longer used. This can work on a racing track with no other cars. With multiple cars on the track, a detection system (lidar) is needed to prevent collision.

10.2.1 neo-sliding. An important note explaining the difference between the human record and the developed model's best time, is that Trackmania is a game containing many "tricks" that can help a player to drive faster. One of these is called the neo-slide, a type of drift that allows the player to make the car take sharp turns at a higher speed than usual. Since the test track contains many of such low-speed sharp turns, using this drifting technique can allow the driver to set much faster times than using normal steering. Neo-sliding is a difficult and precise technique that only very good Trackmania players can do reliably. But since, especially on this specific track, neo-sliding is so important, it would be interesting to see if this technique can also be learned by AI, by for example using more training time or a different set of input-variables. It would also be interesting to test the newly developed model on a track that does not involve neo-sliding curves.

10.3 Future work

The focus of this study was on input data containing the upcoming track. The study was valid for a 2d map. Since tracks in Trackmania are most of the times not flat, this study could be expanded upon by researching the possibility of a 3D representation of a track. Different track surfaces, for example a dirt road instead of a tarmac road, could also be taken into account.

In this study lidar sensors were ignored. In future work, a combination of lidar and track information as input data could be investigated. This makes it comparable to autonomous driving in the real world.

11 CONCLUSION

11.1 RQ1: What are the key distinctions between Sophy and Tmrl-lidar?

The key distinction that was found between Sophy and Tmrl-lidar is a difference in input data. The most notable difference in input data was the lack of information about upcoming track in Tmrl-lidar.

11.2 RQ2: What modifications can be transferred from Sophy to Tmrl-lidar?

While not natively supported in Trackmania, a function was implemented that generates the upcoming track in a similar way to Sophy.

11.3 RQ3: How do these changes affect Tmrl-lidar's performance?

The AI that was trained using the upcoming track as input data affected the performance of Tmrl-lidar in a positive way. The lap

time comparison shows that this new solution managed to improve Tmrl-lidar's record from 44.631 to 35.465 seconds.

12 REFERENCES

REFERENCES

[1] Kai Arulkumaran et al. "A Brief Survey of Deep Reinforcement Learning". In: *IEEE Signal Processing Magazine* 34 (Aug. 2017). DOI: 10.1109/MSP.2017.2743240.

[2] Senay Baydas and Bulent Karakas. "Defining a curve as a Bezier curve". In: *Journal of Taibah University for Science* 13.1 (2019), pp. 522–528. DOI: 10.1080/16583655.2019.1601913.

[3] Yann Bouteiller, Edouard Geze, and Andrej Gobe. *TMRL, Reinforcement Learning for real-time applications*. URL: https://github.com/trackmania-rl/tmrl. (accessed: 23/11/2022).

[4] Peide Cai et al. "High-Speed Autonomous Drifting With Deep Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1247–1254. DOI: 10.1109/LRA.2020.2967299.

[5] Yan Duan et al. *Benchmarking Deep Reinforcement Learning for Continuous Control*. 2016. arXiv: 1604.06778 [cs.LG].

[6] Florian Fuchs et al. "Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4257–4264. DOI: 10.1109/LRA.2021.3064284.

[7] Scott Fujimoto, Herke van Hoof, and David Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. arXiv: 1802.09477 [cs.AI].

[8] Kıvanç Güçkıran and Bülent Bolat. "Autonomous Car Racing in Simulation Environment Using Deep Reinforcement Learning". In: *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*. 2019, pp. 1–6. DOI: 10.1109/ASYU48272.2019.8946332.

[9] Tuomas Haarnoja et al. *Soft Actor-Critic Algorithms and Applications*. 2019. arXiv: 1812.05905 [cs.LG].

[10] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

[11] Sony Interactive Entertainment Inc. *Gran Turismo Sport*. URL: https://www.gran-turismo.com/nl/products/gtsport/. (accessed: 24/01/2023).

[12] Yunpeng Pan et al. "Agile Off-Road Autonomous Driving Using End-to-End Deep Imitation Learning". In: *CoRR* abs/1709.07174 (2017). arXiv: 1709.07174. URL: http://arxiv.org/abs/1709.07174.

[13] Dean A Pomerleau. "ALVINN: An autonomous land vehicle in a neural network". In: *Advances in neural information processing systems*. 1989, pp. 524–532.

[14] Reddit. *r/TrackMania*. Reddit. May 7, 2023. URL: https://www.reddit.com/r/TrackMania/ (visited on 05/07/2023).

[15] Ahmad El Sallab et al. *End-to-End Deep Reinforcement Learning for Lane Keeping Assist*. 2016. arXiv: 1612.04340 [stat.ML].

[16] TMDojo. *TMDojo*. URL: https://tmdojo.com/.

[17] TMRL. *TrackMania Reinforcement Learning*. Weights & Biases. 2021. URL: https://wandb.ai/tmrl/tmrl (visited on 05/07/2023).

[18] Trackmania.io. *Trackmania.io*. 2023. URL: https://trackmania.io/ (visited on 05/07/2023).

[19] Ubisoft. *Trackmania 2020*. URL: https://www.ubisoft.com/nl-nl/game/trackmania/trackmania. (accessed: 24/01/2023).

[20] Haoyue Wang et al. "Model Predictive Control of Autonomous Vehicles Using Sequence Convex Programming". In: *2022 37th Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. 2022, pp. 925–929. DOI: 10.1109/YAC57282.2022.10023886.

[21] Grady Williams, Andrew Aldrich, and Evangelos A. Theodorou. "Model predictive path integral control: From theory to parallel computation". In: *Journal of Guidance, Control, and Dynamics* 40.2 (2017), pp. 344–357. DOI: 10.2514/1.G001921. URL: https://doi.org/10.2514/1.G001921.

[22] XertroV. *XertroV (GitHub profile)*. 2012–present. URL: https://github.com/XertroV (visited on 05/07/2023).

13 APPENDIX

13.1 Distance calculations

In this appendix the calculation of the map is explained. The first part we want to calculate is the vertical direction of each of these lines, the most obvious line to start is the one in the middle, here made orange. We want to know how long the piece of road from the camera to the wall is. What we know about this line is that it has a length of a certain amount of pixels, for example, 150. the line

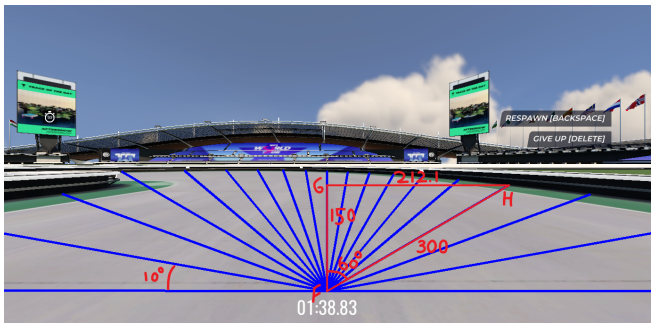

Fig. 17. We need to calculate the vertical part of each line in order to calculate how many meters the point is forward in 3d space.

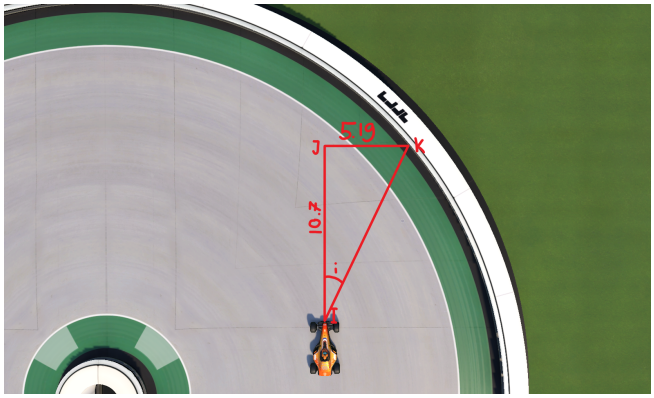

Fig. 18. IJ is the forward distance in 3d space, JK is the horizontal distance is 3d space.

starts 50 pixels from the bottom and the picture has a height of 500 pixels (488 in practice, but 500 for explanation sake).

In fig 9 you can see the car and camera from the side. The camera is on the front of the car and slightly higher. What we need to know is the length BC. To do this, we will first need to know the total vertical angle of the camera, which is 80 degrees. Now we can calculate the angle beta, ((150px+50px)/500px)*80 deg = 32 degrees

The total angle, alpha, will then be 50 + 32 = 82 deg AB is known to be 1.5 meter, so BC is tan(82 deg)*1.5 = 10.7 meter

We now know how to use the vertical distance in pixels to calculate the distance in the 3d space. For the distance sensor in the middle this approach works, but the other sensors are angled with increments of 10°. This means that we will have to calculate both the vertical part of each line and the horizontal part to find the coordinates of the point. The calculation of the vertical part (FG) is illustrated in figure 17. We use the same calculation from before to calculate how many meters this in in 3d space in the forward direction.

The next step to finding the coordinate where the distance sensor hits the wall, is to calculate the horizontal distance in 3d space, JK in Figure 18. To calculate this, we need to know the angel i. We can calculate i from figure 17 as follows. We know that the total width of the window is 958 pixels, together with the height of the windows,

488 pixels and the vertical total angle of the camera, the horizontal angle of the camera is 117°. To obtain i, we have to divide GH by the total width of the window, times 117°. In this case, 212.1/958 * 117°, in this case i is 25.9° With this value we are now able to calculate JK, JK = tan(i)*10.7m = 5.19 m

We now calculate the relative position of the walls from the camera. We can use this information together with the coordinates of the camera and the angle which it is pointing, to find the coordinates of the walls.