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

ACM Reference Format:
Laurens Neinders. 2023. Improving Trackmania Reinforcement Learning Performance: A Comparison of Sophy and Trackmania AI. 1, 1 (February 2023), 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

In 2021, an AI named "Sophy"[4] achieved a remarkable feat by outperforming the best human players in the racing game "Gran Turismo Sport". Another racing game, Trackmania [7], has also seen the development of AI systems.

Another racing game, Trackmania, has also seen the development of AI systems. A French group of developers have created multiple AI systems for Trackmania using different techniques[3]. In a French show, they demonstrated one of their AI models (TMAI) setting a time of around 45 seconds on a test track, which is already a strong performance, but still not as fast as the human record of 35 seconds.

The goal of this research is to improve TMAI performance by analyzing the differences between Sophy and Trackmania AI. The purpose of this study is to identify changes that can be transferred to TMAI and to analyze the difference in performance. Finding differences between the two systems will drive the understanding of reinforcement learning algorithms forward. This in turn will enable the creation of higher-performance systems.

Fig. 2. Gran Turismo Sport

Fig. 3. Trackmania

1 INTRODUCTION

1.1 Background

1.1.1 what is Trackmania? Trackmania is a racing game where players try to drive from start to finish in the shortest amount of time. Players can create their own tracks (important). Gran Turismo Sport is a realistic racing game the goal of which is to win races.

A group of developers has created a framework called tmrl (Trackmania Reinforcement Learning)[3], with which you can build and

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

Fig. 4. Tmrl-test track

train AIs that drive trackmania. The AI model that we use as a baseline for this thesis is their lidar-based model. For simplicity, we will call this model TMAI (Trackmania AI) from now on.

The French developers created a track 4 (tmrl-test) on which they drove with their AI model 10 times. They set an average time of 47,176 seconds.

I tried this track myself and completed it on average in 38 seconds. Therefore, there is room for improvement.

Another AI that can drive in a racing game is Sophy AI[4], developed by the Sony AI group. This AI is developed for a different racing game, namely "Gran Turismo Sport" (GTS) [6].

Gran Turismo Sport is a very different game than Trackmania, the cars, physics and track are all much more realistic. Tracks are only made by the developers, where track creation is an important part of Trackmania.

Sony managed to create an AI that could rival and beat the best humans in GTS, which leads to the question: Can we learn from how Sophy works and improve TMAI?

1.2 Research questions

In order to test if it possible to improve TMAI based on Sophy AI, we have to answer the following research questions:

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

2 METHODOLOGY

The approach to answering research questions starts by comparing and analyzing the two AI agents, Sophy and TMAI, that were trained to play racing games using reinforcement learning. The study will then focus on identifying key differences between the two models and how they were trained, with the goal of understanding how Sophy was able to outperform human players in "Gran Turismo Sport". After this, the study will analyze which of these differences can be transferred from Sophy to Trackmania AI. And finally, we will implement the changes to evaluate the difference they make in performance.

This leads to the following steps:

(1) Data collection: Collect data on the performance and inner workings of Sophy and TMAI, including their lap times, driving lines, algorithm used, input data and other relevant metrics.
(2) Identifying key differences: Analyze the data from the first step and identify the key differences between the two systems.
(3) Transfer differences: Identify changes that can be transferred from Sophy to TMAI to improve its performance, such as using the same type of input data, using similar algorithms, or adjusting other parameters used.
(4) Evaluation of changes: Implement the identified changes on TMAI and evaluate the impact of these changes on its performance.

3 RESULTS

3.1 Data collection

In this section we will collect all relevant information about the two systems: Sophy and TMAI

3.1.1 Lap times. The first objective is to prove that Sophy is, in fact, much better than TMAI. 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 directly compare them on the same tracks or environments; what we can do is compare the two with human performance in their respective games. This will still give a good indication of the strength of the AIs, especially when a comparison is made between the AIs and the best human players.

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

Lap times[sec]: TMAI vs humans			
	TMAI	**fastest human**	**median humans**
tmrl-test track	45.643	34.308	37.895

3.1.2 driving lines. When driving, a factor that has a great impact on lap times is: "driving lines", the way the car drives through corners. You want to drive through a corner as fast as possible and to do this often and out-in-out approach is the fastest. This means that you drive from the outside of the corner to the inside and then to the outside of the upcoming track.

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

3.1.3 AI Algorithms. Now that we know what both systems are capable of, the next step is to understand how they work. Both

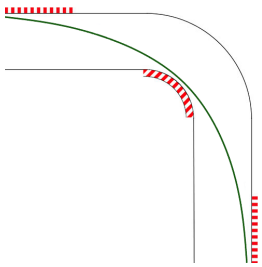

Fig. 5. Racing lines Sophy

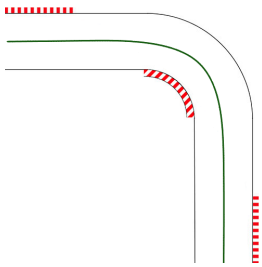

Fig. 6. Racing lines TMAI

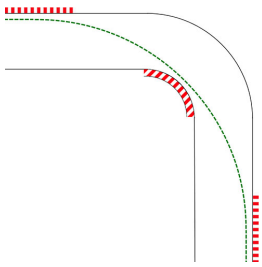

Fig. 7. The out-in-out approach

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 happens to be the same for both TMAI and Sophy. This algorithm is soft actor-critic[5]. 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.

3.1.4 Input data. The reinforcement learning algorithm receives a set of variables representing the environment, it then 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 the environment to make a solid decision that will give it the maximum reward.

We will now compare the environments of Sophy and TMAI.

Sophy uses an environment consisting of 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 course, based on the current speed of the car.

The environment that TMAI 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

We observe a difference in observation space between the two systems, which we will analyze in chapter 3.2.

3.1.5 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 two are in fact not equivalent as one will have to consider optimal trajectory, which may imply slowing down for corners etc.

That is why both Sophy and TMAI use a reward based on progress of the total track. This means that the AI will receive reward based on how much of the track has been convert since the last timestamp.

3.2 Identifying key differences

In the last chapter, we discovered the main aspects of the two AI systems, Sophy and TMAI. When comparing the two, the three main aspects are: type of algorithm, reward function, and input data. In the first two, we do not observe notable differences; there are minor differences in exact implementation, but these do not seem likely to be the sole 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 observe is that almost all of the variables from the observation space of TMAI are also in the observation space of Sophy, with the exception of the history of 4 distance-measurement observation. On top of that, 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 to

next corner look and might be able to adjust its way current driving to accommodate for this upcoming corner.

Figure 8 gives a visual representation of the information that distance measurements provide compared to what the measurements of the track ahead provide. You can see 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 9.

The theory that arises from these findings is that the key difference between Sophy and TMAI is the lack of information of the track ahead as input data for TMAI.

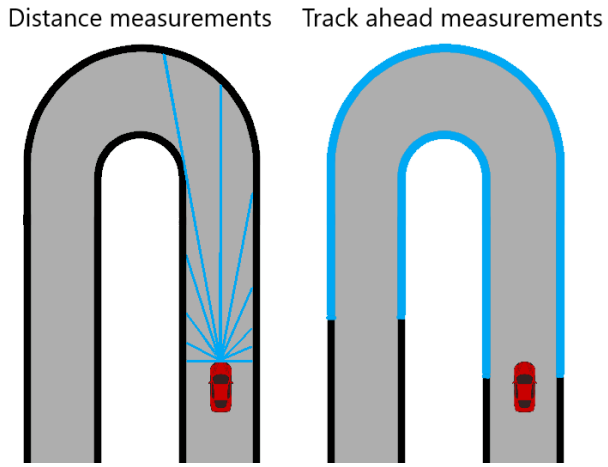

Fig. 8. Distance measurements only give information about a short piece of track ahead, whereas curvature measurements of the track ahead gives more information to make a better decision.

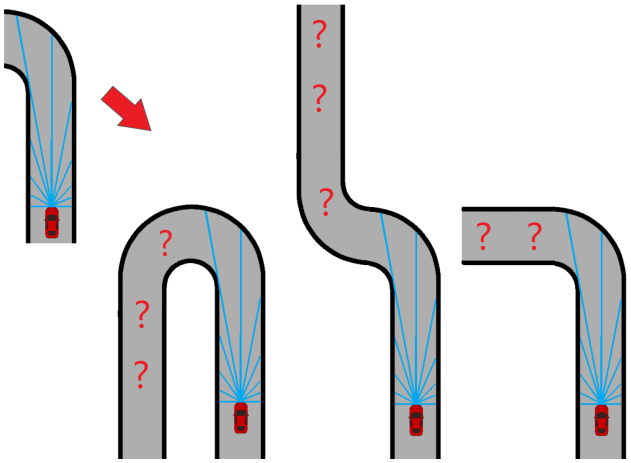

Fig. 9. TMAI can't know how the next corner will look.

3.3 Transfer differences

To test the theory from the last chapter, we will have to implement a system that can provide information about the piece of track ahead of the car. The Sophy system makes use of a built-in feature of the game it plays, "Gran Turismo Sport". This feature provides the upcoming piece of track, which Sophy can use to predict the best way to drive. However, this information is not available in the game TMAI was made for, "Trackmania". Therefore in order to test the theory, a similar system had to be made. details about the implementation can be found in the appendix.

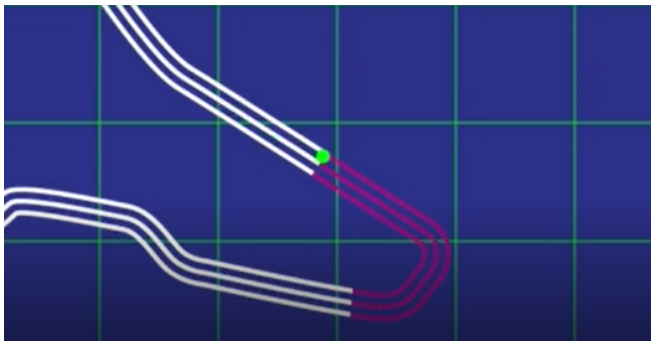

Fig. 10. Sophy has information of the track, 6 seconds ahead, based on the current speed

3.4 Comparison lap times

This section contains the results of training an AI with the upcoming track as input. The newly trained AI will here be called UpcomingTrack AI (UTAI for short).

These are the lap times driven on the tmrl-test track, the track for which TMAI is trained.

tmrl-test track, Lap times[sec]: TMAI vs. Upcoming Track AI				
track	**TMAI**	**UTAI**	**Best Human**	**Average Human**
Fastest lap	44.413	38.402	34.308	36.985
Average (std. dev.)	47.176 s (0.769)	40.178 (0.517)	Unknown	38.126 (0.531)

4 CONCLUSION

4.1 RQ1: What are the key distinctions between Sophy and TMAI?

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

4.2 RQ2: What modifications can be transferred from Sophy to TMAI?

While not natively supported in Trackmania, we managed to implement a function that replicates similar input data about upcoming track as Sophy uses.

4.3 RQ3: How do these changes affect TMAI's performance?

The AI trained using the upcoming track as input data affected TMAI's performance in a positive way. The lap time comparison

shows that this new solution managed to beat TMAI's records by a significant margin.

5 DISCUSSION

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

5.0.1 Significance of obtained results. This study shows the importance of carefully chosen input data in the field of reinforcement learning. The analyzed AI, TMAI, used many state-of-the-art technologies and techniques but left potential performance on the table by not having enough information for the AI to correctly predict how to drive in the fastest way possible.

5.0.2 Limitations. While the study helped proved insight in the importance of input data in reinforcement learning, the use-case for the specific AI that was created is narrow. Not only does it just work in the trackmania game, which is to be expected, but it also works only in very specific scenarios. For the AI to work, a track needs 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 would 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 intricacies of Trackmania.

5.0.3 Future work. This study's focus was on upcoming track information. This subject can be delved further into, one could for example study what exact representation of data would yield the best results, and shortest training time. This study used 30 coordinates of wall up ahead, but a different amount of, or with different distance between points could be better. There are also other possibilities, such as drawing a curve through the points and providing the AI with the parameters of the curve.

It would also be beneficial to research optimal training parameters for the tmrl framework, or reinforcement learning in general.

My last suggestion for future work would be to investigate different representations that can capture different aspects of the track, such as the difference in height and road surface. This would enable the creation of AIs that can drive on a wider variety of tracks.

6 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] 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.

[5] 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.

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

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

7 APPENDICES

7.1 Calculation of the track.

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

The goal was to create a representation of a track by driving on it once, then using the data that the distance sensors provide to calculate the position of the walls. This will create a representation of the track that can then be used to train TMAI on.

So, the first part is to calculate the position of wall based on the measurements of distance sensors. The next part will be to convert the many data points that emerge from recording the track and doing the calculations, into an input that is useful for the AI.

7.1.1 Distance calculations. To calculate where the position of the walls of the track are, we start by looking at the distance sensors that are implemented in the tmrl framework.

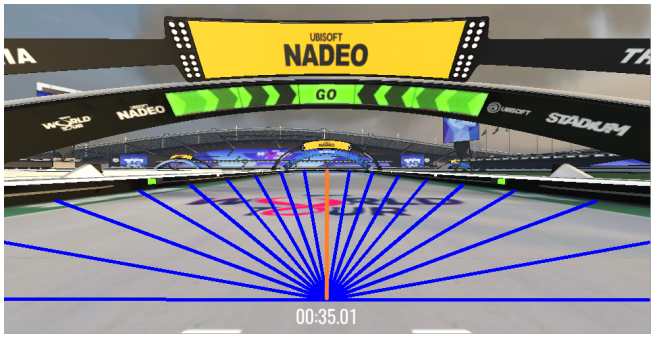

Fig. 11. the distance sensors are lines from a point below the center of the image, to the walls of the track

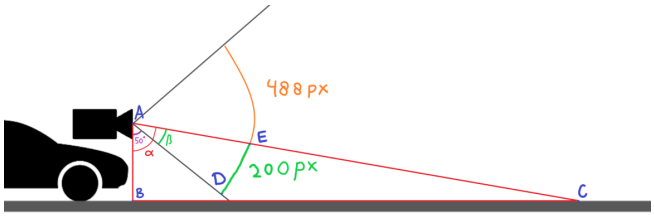

Fig. 12. side view of car, goal is to calculate BC

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 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 12 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

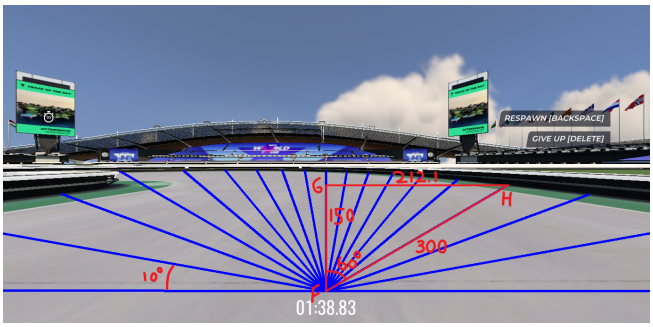

Fig. 13. We need to calculate the vertical part of each line in order to calculate how many meters the point is forward in 3d space.

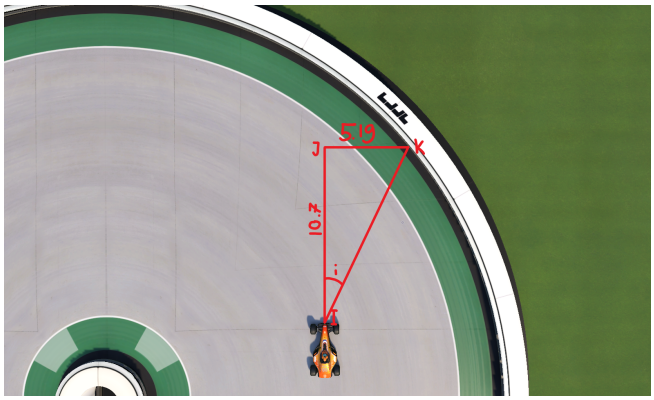

Fig. 14. IJ is the forward distance in 3d space, JK is the horizontal distance is 3d space.

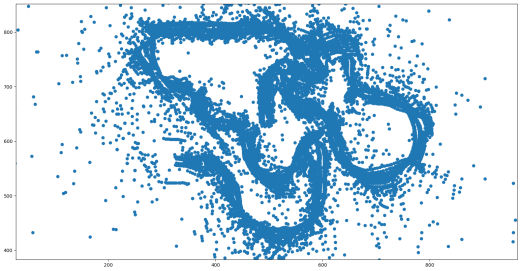

Fig. 15. The first try on programming a representation of the track. The map has a lot of noise and outliers, but the general shape is visible

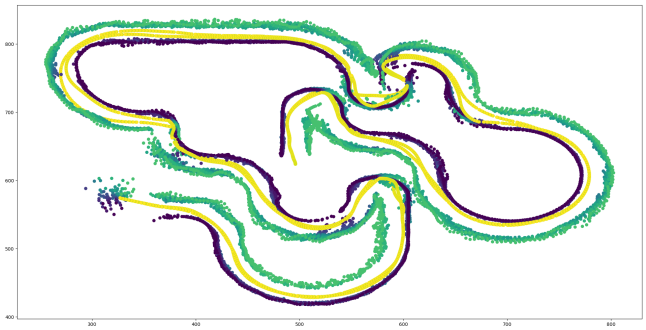

Fig. 16. This how the map looks later in the process, a lot of outliers are gone and both sides of the track are clearly distinguishable. The yellow line is the trajectory of the car, purple and green are the left and right side of the track respectively.

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 13. 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 **??**. To calculate this, we need to know the angel i. We can calculate i from figure 13 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.

7.1.2 Creating the map. The next step is to record a large series of images and distance measurements while driving the track and combine the coordinates found.

Something to note here is that while driving, 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.

Figure 15 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 16 shows the map in a later phase of development. 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 17 is the final representation of the track. In this version, the inaccuracies from before are hammered out. To reach this point, a Bezier curve [2] is drawn through the points, which fills any gaps

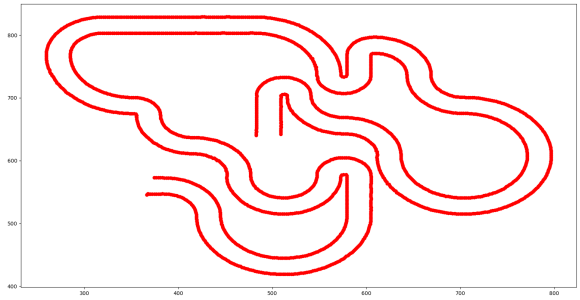

Fig. 17. This is the final version of the map, all outliers are removed and the points that make up the track are equally spaced out

in the data, and then this curve is divided by a number of points along it.

7.1.3 Input for the AI. 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 which will be what the AI gets to see of the track at any given time.

To do this, we take the point closest to the position of the car, and from there we take the next n points along the track. This means that the distance that the AI can see infront is fixed. This is different from how Sophy does this; instead of a fixed distance, they provide a part of the track based on the current speed of the car.

We chose a fixed distance because a variable distance would increase complexity in both implementation and AI training.

After a small part of the track is cut out, we subtract the position of the car such that the track is always centered the (0,0) point. This is useful for AI training because, say you have the same corner but on a different physical place, both will be normalized to (0,0), and the AI will not have to learn both corners separately. We also angled the track based on the rotation of the car, which means that the direction that the car is pointing will become the vertical axis in the input. Figure 18 shows this process.

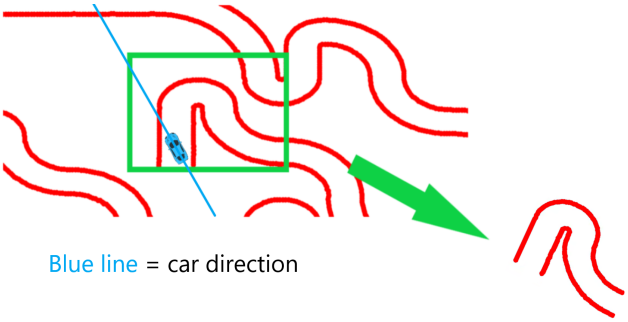

Fig. 18. A small part of the track-map is cut out, normalized, rotated and given to the AI as input.