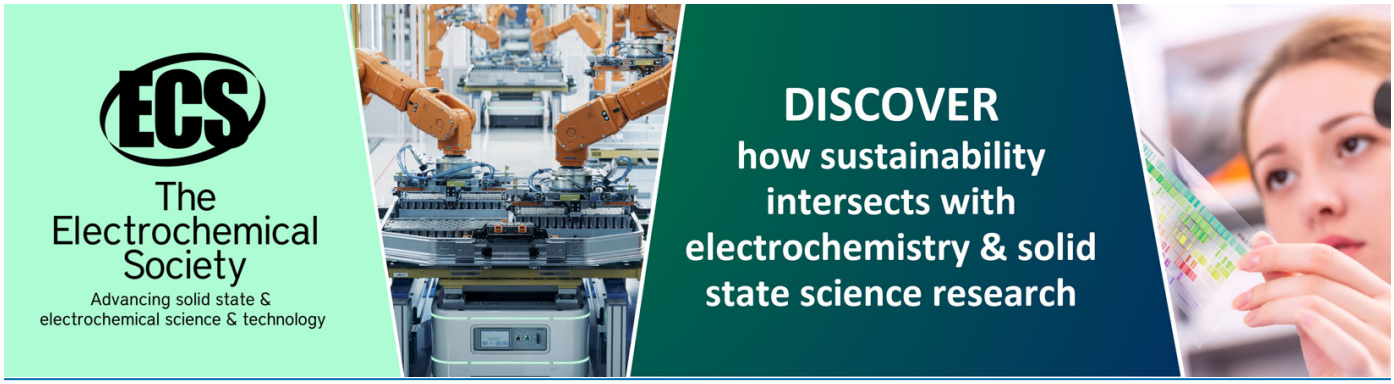
Modelling the situation of driving on the grip limit with DDPG algorithm



IOP Conf. Series: Materials Science and Engineering **393** (2018) 012031 doi:10.1088/1757-899X/393/1/012031

Modelling the situation of driving on the grip limit with DDPG algorithm

G Bári[1]

[1]John von Neumann University, 6000 Kecskemét, Izsáki út 10

E-mail: bairg82@gmail.com

Abstract. The number of papers in the topic of autonomous vehicle research is growing exponentially. This paper addresses the problem of self driving a car on tire grip limit, in other words it gives a simple model of a race car driver. Driving is transformed into a simple deep learning problem, where the agent has one action, that is the direction in which the actual speed vector needs to be modified for the next step, and the environment state contains of the actual position and speed. The environment models the race track as a two colour map, to decide on- and off track positions, and the car as a point mass with maximal possible acceleration according to the so called GG diagram. Results show that the agent can learn how to drive on the track under the described circumstances.

1. Introduction

Self-driving is a widely researched field. In this work, we deal with the situation of self-driving a car on the tire grip limit. Although under normal driving conditions, this is a rare situation, and it is mainly interesting in racing, but in case of collision avoidance, when the vehicle must use its maximum potential to change its actual motion state, maneuvering capabilities on the grip limit has a big importance in production vehicles also.

In general, during self-driving the task is to control vehicle inputs, (e.g. pedals, and steering) based on information coming from sensors. (LIDAR/RADAR, Cameras, GPS/INS, etc.) This can be performed in several ways. [1] One extreme approach is the so called end-to-end solution, where neural networks are used to map sensor signals to vehicle controls [2] In case of other, more conservative approaches, the self-driving task is usually decomposed into sub-functions. [3]

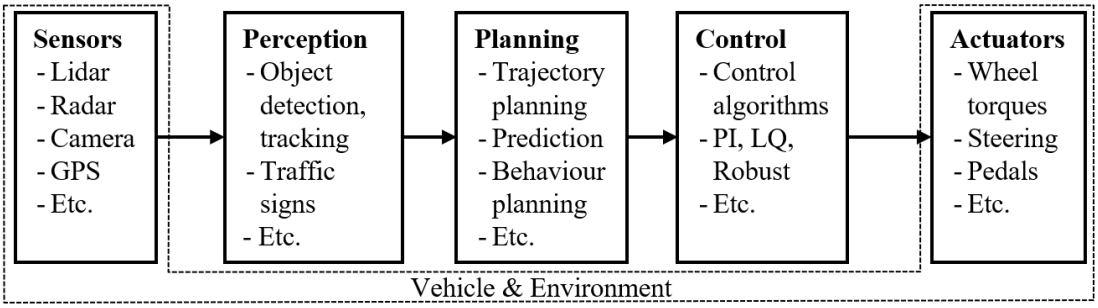


Figure 1. A common way for self-driving problem decomposition.

One of the sub-functions during self-driving is trajectory planning. [4] In case of trajectory planning, one interesting issue is to understand how rapid changes in vehicle motion are possible, what trajectories can the vehicle perform. Tire grip level will have big effect on this. As grip level is always varying, and it is a parameter that is basically impossible to predict it in advance (with the level of accuracy, required in racing), it is a common solution to aim for a worst-case grip level, and plan trajectory with that.

2. Race car driving

In contrast to road car driving, understanding the grip limit and being able to describe its effecting variables, is critical for winning in racing environment. To put grip sensitivity in perspective, 1% change in grip level results 0,15% change in lap time. [5] In numbers, this means that on a 1minute 40second lap, dropping average grip level from 1.0 to 0.99 will result in 0.15sec increase in lap time. So, if we want a racing driver model, that can use this 0.15% time difference, we should first understand how real race drivers drive their vehicles in the limit.

2.1. Acceleration direction on the GG diagram

There are several works addressing racing lines, and race car driving [5][6][7] A common thing to note in these works is, that in the world of racing, when racing drivers talk about issues in their laps, about car behavior, or they just want to identify, when a given phenomenon occurred, they use corner numbers, and phases of those corners. For example, they will say “During turn in of corner number 5 the car has huge snap oversteer” or “In corner 12, mid corner the car is understeering a lot”. These cornering phases are described in depth in [8] For current work the important point is, that these phases basically show the direction of the resultant acceleration in the center of gravity. This is shown in Figure 2. The diagrams, shown in Figure 2 are the so-called GG diagrams [8]. A common conception in racing, is to plot the longitudinal and lateral acceleration in X-Y scatter plot, that is called the GG diagram. This diagram has a lot of information about the track-vehicle-driver system, and the available grip.

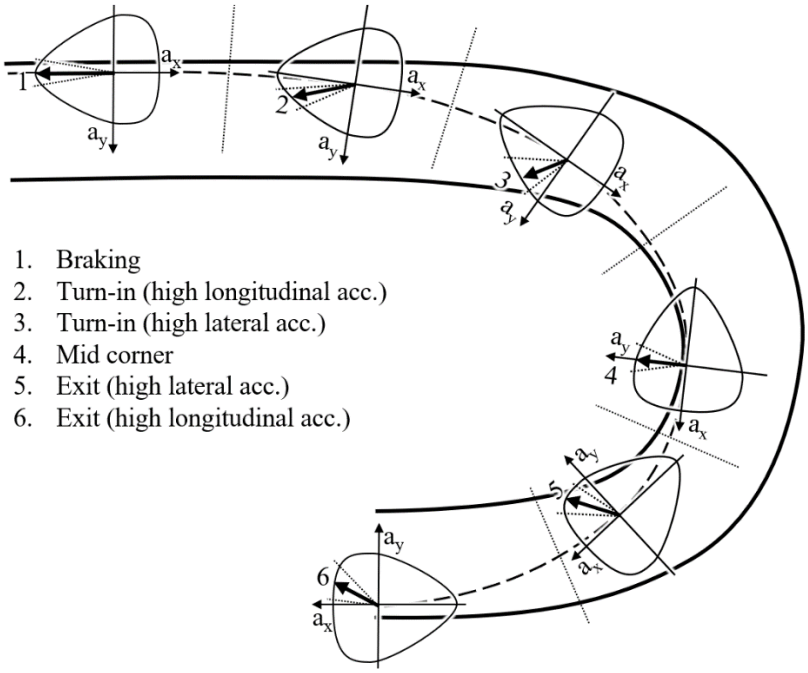


Figure 2. The phases of a general corner in racing. The resultant acceleration in the centre of gravity is also shown in a vehicle fixed frame.

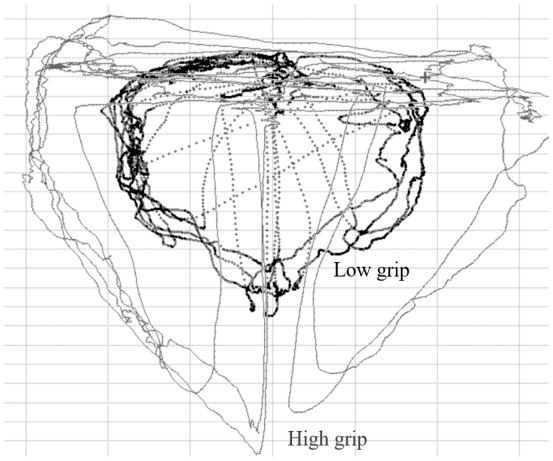


Figure 3. GG diagrams for a fast lap, under low grip e.g. wet (black) and high grip e.g. dry (grey) conditions.

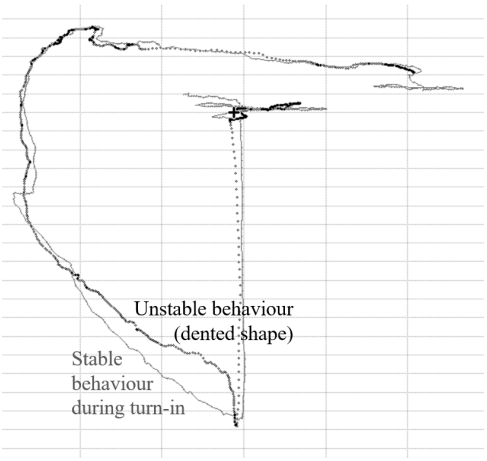


Figure 4. GG diagrams for stable (grey) and unstable (black) vehicle behaviour during the turn-in phase of a corner.

In Figure 3 two GG diagram can be seen, one for low, another for higher grip conditions. Obviously, the size of the GG diagrams correlates well with the grip level. Higher grip will result in higher accelerations for the vehicle. In Figure 4 the effect of vehicle stability can be seen. When a vehicle motion is unstable in a given direction of the GG, the driver cannot use the full potential of the vehicle-track system, so the magnitude of the GG in that direction is smaller than it could be. This unused grip potential can result in a "dented" shape, as shown in Figure 4. Although only these two parameters are mentioned here, but almost every important vehicle or driver or track parameter effect the GG shape. This is the main reason, why it is a good and simple concept to model the vehicle dynamic properties with it.

The important conclusion from above is, that during race car driving there is a layer of decision making in the driver's mind, that is deciding about the desired direction on the GG diagram. Racing drivers decide in each moment about how they want to use the vehicle's motion-state-changing capability, which direction they want to change the speed vector. Then, the next layer in their driving process is that the driver needs to actuate the vehicle inputs, (e.g., steering wheel, pedals) in a way that the desired acceleration direction is created, in the shortest time, with the maximum possible magnitude.

During this process, the racing driver does not have a clear trajectory in mind. He is continuously trying to find the proper direction, and maximum magnitude of the GG, and the trajectory evolves while the boundary of the GG diagram is tracked. During this boundary tracking, there is a certain level of risk arising from the fact that the car can slide off the track when crossing its limits. For example, during a qualifying session, when the drivers usually have 3 – 4 attempts to make their quickest lap time, it is usual that we see laps getting quicker and quicker from attempt to attempt. There can be technical reasons for this (e.g. track evolution) but the fact that drivers accepting higher and higher chances of making mistakes, is another key reason for this. Managing this risk is also important, to have better and better lap times, and it is done by driver intuition during race car driving.

2.2. The D2G2 decomposition

For being able to model this intuition in risk management, and skip the problematic trajectory planning step (that needs deterministic motion states at some point of the driving algorithm), we suggest a new way of decomposing the driving task, as shown in Figure 5.

In the first sub-function, based on sensor information, the desired direction of the GG (D2G2) is defined. After, in the second, so called vehicle dynamic control (VDC) sub-function, the actuators are controlled with a goal to realize x-y acceleration in this direction, with the highest possible magnitude,

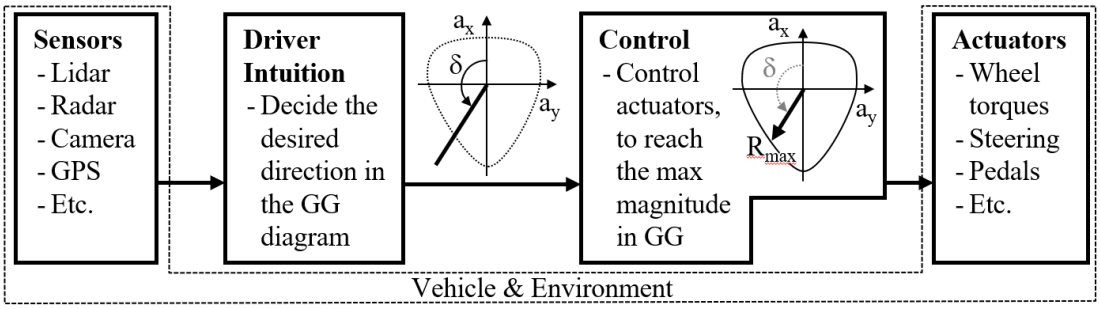


Figure 5. The "desired direction on GG" (D2G2) type problem decomposition for self driving.

in the shortest possible time. This second, VDC part is not presented in this work, the unperfect behavior of this layer, is handled with the driven vehicle, through the shape, and uncertainness of the GG diagram. In the followings, a solution is presented for the first, Driver Intuition part of the algorithm.

3. Deep learning background

Modelling this kind of human intuition with conventional model-based control techniques is not easy. In the past years however, results in the field of deep learning shown promising solutions, without hand crafted features, or models, with new machine learning techniques. For example, levels of image and speech recognition, that were impossible to reach before, become possible with these methods, and in [9] an artificial intelligence (A.I.) agent was trained with deep learning methods, to play table games like Chess and Go. The A.I. played higher level than human players do, and it was shown that some level of intuition evolved in the agent.

These kind of "playing" agents usually trained with so called reinforced learning techniques, using deep neural networks.[10] Reinforcement learning (RL) is an area of machine learning inspired by behaviourist psychology, concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. In machine learning, the environment is typically formulated as a Markov decision process (MDP) where an episode of this process (e.g. one game) forms a finite sequence of states, actions and rewards. A Markov decision process relies on the Markov assumption, that the probability of the next state, depends only on current state and action, but not on preceding states or actions.[11]

So, if we want to model race car driving on the grip limit, turning race car driving into such a game, and defining a suitable MDP as environment, seems to be a step in a good direction.

4. Defining the race car driving environment

In general, an MDP is defined as follows: The environment is in a certain state and the agent can perform certain actions in the environment. On the one hand, these actions result in a reward, on the other hand actions also transform the environment and lead to a new state, where the agent can perform another action, and so on. The rules for how actions are chosen are called policy. The environment in general is stochastic, which means the next state may be somewhat random.
Generally, in case of race car driving, the followings could be chosen:

- The state of the environment can be described by sensor signal values from the car, (e.g. LIDAR, Camera pixels, Inertial sensors, GPS position, etc.).
- The action taken by the agent could be the desired direction of the GG diagram (D2G2).
- As reward, the negative value of the time spent between two steps (timestep), could be used.

4.1. The environment

Using the state definition presented above would require a relatively complicated environment model. In this work we chose the simplest possible case to be able to test our idea.

4.1.1. The state

To test our approach, we defined the simplest environment that is still able to represent the mentioned phenomena of racing. In this simple case, the race track is given by a three-colour image defining on and off-the-track positions. The vehicle is treated as a point, and its position is defined by the coordinates of the pixels.

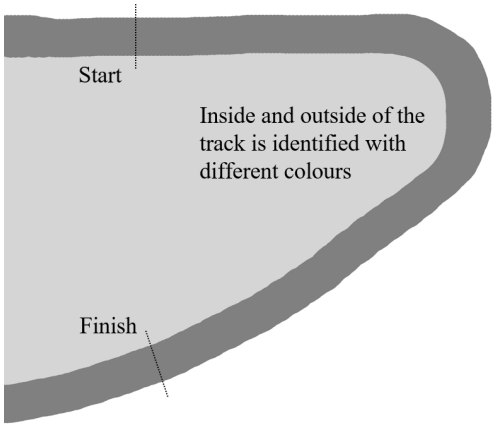


Figure 6. Track representation. Hungaroring 1st corner, 1800 by 1600 pixels. (100 pixels equals 10m).

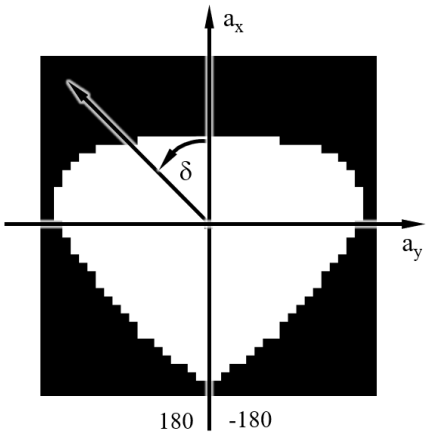


Figure 7. GG diagram representation 41 by 41 pixels. (18 pixels equal 20m/s^2) Also D2G2 representation is shown.

Although this game definition does not require physically valid scale, we wanted to test our algorithm in scale that is relevant to real world conditions. We have chosen to test our A.I. in a track line that is a copy of the Hungaroring race track's first corner. Our track picture size will be 1800 by 1500 pixels, where one pixel equals 0.1 meters. As the real track width is approximately 13 meters this means 130 pixels, and the area covered by this corner is 180 m by150 m. We've chosen the time between two steps as 0.3 seconds, that mean that a 120 km/h motion equals to 100 pixels displacement in one step. Using these numbers, if a speed vector is changed by 18 pixels between two steps, it reflects to a 20 m/s^2 acceleration that is normal for a racing car. Although it might seem too simple for the first sight, but the vehicle's dynamic properties are all modelled purely with a GG diagram. The GG is a black and white, 41 by 41 pixel image, describing the maximal resulting acceleration in longitudinal and lateral acceleration plane. For describing the state, a 4-element state vector is chosen, that is 2 coordinates (x, y) of vehicle speed, and 2 coordinates (x, y) of vehicle position in the track (global) coordinate system, given with pixels.

4.1.2. The action

The action is then given by a single real number, in the -180 … +180 range, that defines the D2G2. Here 0 is the forward direction, -90 refers to pure lateral acceleration with turning right, (+90deg is left) and, ±180 is pure longitudinal acceleration in the direction of braking. The fact that the real and the desired GG directions will be different in real world, is modelled with an additive error with normal distribution in -3 ... +3 sigma.

4.1.3. The reward

The reward is defined as -1 for each step. When the vehicle goes off the track, the episode is terminated, and the reward is defined as the negative value of distance remaining to the finish line multiplied by a given constant (in our case 2). This definition will give higher episode reward if the agent chooses actions that results in less steps (less time) and leaving the track closer to the finish line. In the step, when the vehicle crosses the finish line, meaning that the speed vector in the given step

crosses the finish line, the reward is defined by the portion of the speed vector that is before the finish. E.g. the finish line cuts the speed vector in half, the reward is -0.5.

4.1.4. Step calculation. In each step, the agent decides an action. Then the error representing the uncertainness in the resulting GG is added. As this action is the desired GG direction (D2G2), the change in speed vector can be calculated in vehicle fixed reference frame. Next, the change of speed is transformed into global coordinates and added to the current speed. New speed, and new position can be calculated.

Note, that the agent has no initial information about the track and car. Our goal is to train an agent for a given track, with a vehicle described with a given GG diagram. At this point we don't need an agent that can drive any track, and any car. Although we will use constant GG diagram, but in this concept, it is easy to model speed dependent GG for downforce effects or use GG diagram as location dependent to deal with track surface variations, or vary the GG's stochastic properties depending on the desired direction, to model e.g. unstable vehicle behavior in the turn in phase of the cornering. We plan to address these cases in a later work.

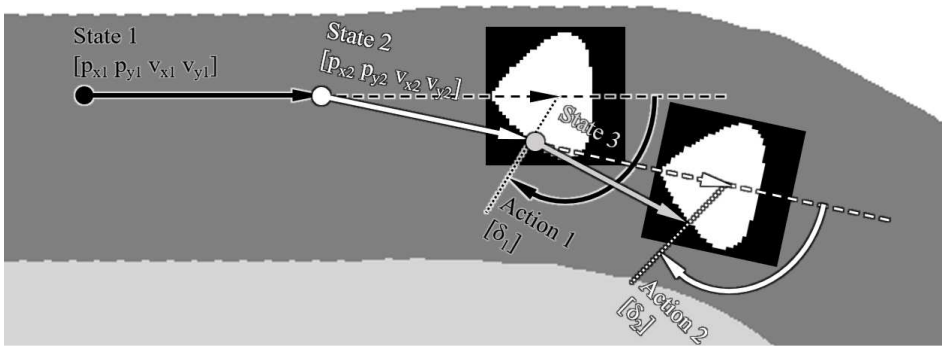


Figure 8. Explanation of the environment, the state and action definitions.

5. Defining the net for the agent

After defining the environment, the next step is to decide how to model the A.I. agent. There are several different solutions for reinforcement learning problems. The first breakthrough in this filed, [13] used so called Q-Learning, while in [9] a model based on actor-critic method is presented.

Driving in simulation environment was solved in several ways, [12],[14], but those works presented end-to-end strategies, so the agent controlled steering and pedal positions, while the state was defined as screen pixels. We've chosen different, much more simple state, and actuator space definitions. Although our current goal is to solve a quite simple game, to test the theory of the proposed problem decomposition, we'd like to have a framework, that is able to scale to higher dimension actuator, and state spaces. Among the methods we found in literature, we've chosen deep deterministic policy gradient (DDPG) method. [15]

In DDPG there are two neural networks. One is the so-called actor network, that is defining the policy to map states into actions, and there is a so called critic network, that is basically telling the goodness of an action in a given state.

In our case, the actor network has 4 fully connected layers, with 400, 100, 30, 10 neurons, all of them with ReLU activations. Input for the 1st layer is the 4 dimensional state vector [speedX, speedY, positionX, positionY], and the output of the last layer is a single action value, defining the D2G2.

The critic network has 5 fully connected layers, with ReLU activations, using 300, 200, 90, 40, 20 neurons. The critic network is using the state and action values as inputs, to predict "goodness" of an action in a state. The state is used as an input for the 1st layer, but the action is injected only in the 2nd layer. This is a commonly used trick in the literature, that we applied too. The output of the last layer is the temporal difference value, that is used to calculate the gradients of both the actor, and critic networks.

6. Training and results

There are several tricks and thumb rules that make training of such networks effective. One of these techniques, is to use experience replay. It is highly suggested to use a replay buffer to store the experiences of the agent during training, and then randomly sample experiences to use for learning. This technique is known as experience replay.

It is also common, to use human played episodes during the learning process, although newer techniques based on self-playing [9] do not use human played samples or other prior knowledge. In this work we did chose to use these, but basically in a really small, number. We defined only 7 human played episodes, with the goal to have some action sequences that lead to crossing the finish line, and we used these samples to help the training process. In the first 0.5% of the total training episodes, we took away control from the agent, and we used the recorded human episodes to generate actions.

In these "human" episodes, we randomly choose one of the 7 recorded human sequences, and changed each of its actions slightly during the episode, by adding some random noise to the actions. Because of this randomness, it could happen that we run out of recorded actions without reaching the finish line. In this case we just simply chose random actions till finish or running off the track. This way it was possible to create huge number of different experiences that coming from episodes that stay on track for more steps, and this was really useful at the beginning of the training.

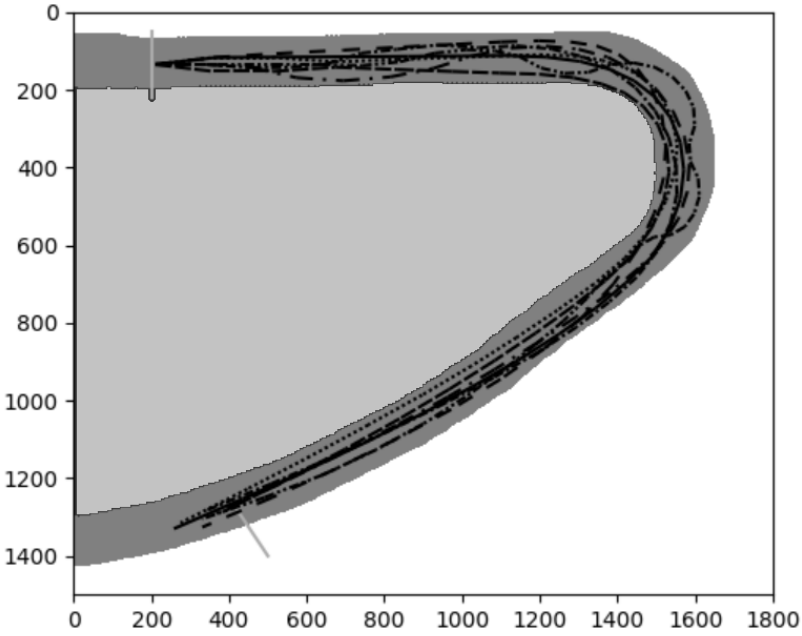


Figure 9. All of the human played episodes (7 pcs.), used to generate exploration experience at the beginning of the training

During training we used the first 0.5% of total number of training episodes to generate these human played experiences in the experience replay. As mentioned earlier, during these episodes we added random noise to the human played actions. This noise was a uniform random number. In the beginning of the 0.5% training period, this random uniform number was between -20 … +20 deg range, and at the end of the 0.5% period, and it decreased linearly to -1 ... +1 deg. This assured that at the beginning of the training we had exploration, but in somewhat controlled, not totally random way. The total number of training episode were 150 000, so the first 0.5% equalled 750 episodes. Total training run on a single CPU core about 8 hours. It is necessary to mention, that the training time was basically slow because of the environment was programmed not efficient, there is a lot of potential in optimising the environment functions.

After the training described before, results showed that the agent learned how to pick actions to reach finish line. It learned the boundaries of the track. It also learned the concept of braking multiple steps before a corner and learned to reach way better lap times that were shown in the human played samples.

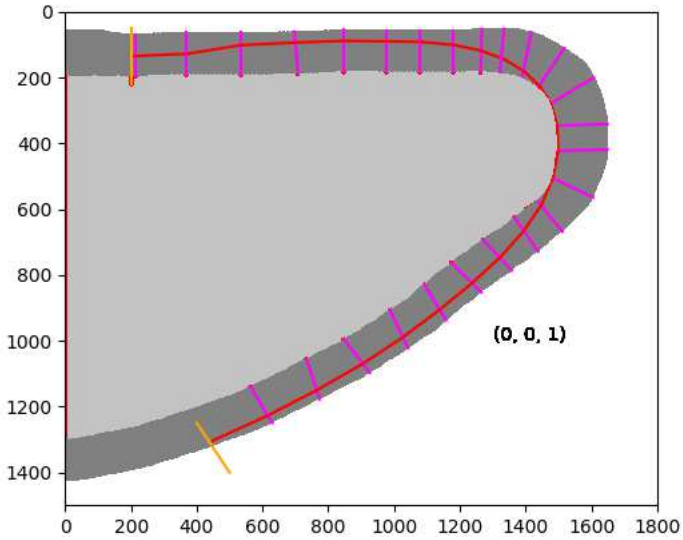


Figure 10. Steps of the learned agent.

7. Conclusion

Although current work showed that the D2G2 decomposition of the self-driving problem is feasible, to tackle real life problems, scaling the method to bigger state space, and more complex environment is necessary. For this we need to create and test a simple version of the second, VDC part of the D2G2 method, in a proper vehicle dynamic simulation environment, and test the proposed D2G2 method with higher dimensional state vector, e.g. camera pixel signals, vehicle state sensors etc.

Another way would be the development of the current A.I. network and environment, to achieve quicker training, and test the resulting “ideal line” variation with different shaped GG diagrams. Also preparing the algorithm to handle not only one corner, but a whole lap would be an interesting step.

Acknowledgement

The research presented in this paper was carried out as part of the EFOP-3.6.2-16-2017-00016 project in the framework of the New Széchenyi Plan. The completion of this project is funded by the European Union and co-financed by the European Social Fund.

References

[1] Xu H, Gao Y, Yu F, and Darrell T 2017 End-to-end Learning of Driving Models from Large-scale Video Datasets, *arXiv Preprint*

[2] Bojarski M, del Testa D, Dworakowski D, Firner B, Flepp B, Goyal P, Jackel L D, Monfort M, Muller U, Zhang J, Zhang X, Zhao J and Zieba K 2016 End to End Learning for Self-driving Cars, *arXiv Preprint*

[3] Silver D, Bagnell J A, and Stentz A 2010 Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain, *International Journal of Robotics Research* **29**(12) 1565-1592

[4] Wang D and Qi F 2001 *Trajectory Planning for a Four-wheel-steering Vehicle*, IEEE International Conference on Robotics and Automation ICRA, Seoul, South Korea, May 21-26, pp. 3320-3325

[5] Kelly D P and Sharp R S 2010 Time-optimal Control of the Race Car: a Numerical Method to Emulate the Ideal Driver, *Vehicle System Dynamics* **48**(12) 1461-74

[6] Katzourakis D I, Velenis E, Abbink D, Happee R and Holweg E 2012 Race-Car Instrumentation for Driving Behavior Studies, *IEEE Transactions on Instrumentation and Measurement* **61**(2) 462-74
[7] Perantoni G and Limebeer D 2014 Optimal Control for a Formula One Car with Variable parameters, *Vehicle System Dynamics* **52**(5) 653-78
[8] Milliken W F and Milliken D L 1995 *Race Car Vehicle Dynamics*, SAE International
[9] Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, Lanctot M, Sifre L, Kumaran D, Graepel T, Lillicrap T, Simonyan K and Hassabis D 2017 Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, *arXiv Preprint*
[10] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D and Riedmiller M 2013 Playing Atari with Deep Reinforcement Learning, *arXiv Preprint*
[11] Goodfellow I, Bengio Y and Courville A 2016 *Deep Learning*, MIT press Cambridge
[12] Mnih V, Badia A P, Mirza M, Graves A, Lillicrap T, Harley T, Silver D and Kavukcuoglu K 2016 *Asynchronous Methods for Deep Reinforcement Learning*, 33rd International Conference on Machine Learning ICML, New York, NY, USA, June 19-24, pp. 1928-1965
[13] LeCun Y, Bengio Y and Hinton G 2015 Deep Learning, *Nature* **521** 436-444
[14] Koutník J, Schmidhuber J, and Gomez F 2014 *Evolving Deep Unsupervised Convolutional Networks For Vision-Based Reinforcement Learning*, Annual Conference on Genetic and Evolutionary Computation GECCO, Vancouver, Canada, July 12-16, pp. 541-548
[15] Lillicrap T P, Hunt J J, Pritzel A, Heess N, Erez T, Tassa Y, Silver D and Wierstra D 2015 Continuous Control with Deep Reinforcement Learning, *arXiv Preprint*