Reinforcement-Learning for Autonomous Parking

Deniz Hofmeister

Delft University of Technology

March 29, 2018

Abstract

Artificial intelligence will soon be able to automate a lot of our daily tasks. An obvious task which artificial intelligence will be able to do is automated driving and parking. This paper aims to autonomously park a car using reinforcement-learning techniques in a simulated environment. Using a model-free discretized simulation of two parking scenarios, it trains Q-learning, SARSA and TD(λ) algorithms. The hyper-parameters linked to all three of the algorithms are optimized and compared. In the end all algorithms achieve the goal, but with different convergence speeds. Q-learning and SARSA perform relatively similar, but out-perform TD(λ). The real-world viability of these methods in low due to computational limitations.

keywords: AI, Automated Parking, Reinforcement-Learning

I. Introduction

Artificial intelligence is applied in a wide range of fields, but one of the most obvious implementations is in the automated driving environment. The idea of having an AI drive our cars is a daunting one. Having to forfeit control of your vehicle to an AI is something a lot of people might not be comfortable with. Therefore, it is done in baby-steps; starting at automated parking, up to full AI control in highly chaotic and busy city centres. This was already touched on in previous work[1] and was an inspiration for this paper, however that paper discussed the viability of using neural-network assisted controllers in contrast to the reinforcement-learning (RL) based controller that this paper will focus on.

This paper aims to train and discuss the viability of such an AI in the automated parking context. Using the most common environments for parking and common vehicle characteristics, an AI is trained to parallel park and to perpendicular park. This is done using various approaches that RL has to offer. A simple scenario is simulated where there is one free parking spot available between two already parked cars. It is the task of the AI to avoid the cars and park correctly in the designated area. The limitations of applying RL in this scenario is the size of the state-action space and computing power. Ideally, one would have an infinite accuracy, but that would in turn require infinite computing power. A balance between realism and computing was used.

Another limitation that this paper puts in place is that only variations of the variables ϵ and γ for 3 different learners are considered[1]. Other hyper-parameters of the algorithms were set and never modified.

The main questions that this paper tries to answer are: is it possible to use RL to park a car correctly? if so, which algorithm of RL is the most effective? Q-learning, SARSA and TD(λ) will be evaluated. What are the effects of changing the hyperparameters ϵ and γ on these learners? Will this type of controller be practical to deploy in the real world?

This paper is structured following the IMRaD convention.

[1]The role of these two variables is described in ii.3

II. Methods

To properly address these questions, one needs to create a simulation environment and implement the proposed learners. The three considered learners are Q-learning, SARSA and TD(λ).

This section is split into the multiple parts. Firstly, The simulation environment in subsection i: Simulation Environment. This will be the world where the AI will reside in and do its computations. This subsection will also justify its design and discuss its limitations. Secondly, the AI's structure will be designed and implemented in ii: AI Design. There the chosen learners and algorithms will be explained. Afterwards the results of the simulations will be discussed in section III.

i. Simulation Environment

The simulation environment will play a big role in the performance of the algorithm, since too small of a simulation space will result in obvious solutions that could have been found by hand and, on the other hand, too big of a space will result in that the computational complexity explodes and no solutions can be found in reasonable time. Secondly, a scenario which has no correlation with the real world will render the solution useless, since the resulting controller cannot be used in reality. The simulation environment will mainly consist of the following four parts:

- Scenario
- Car Dynamics Model
- Mapping into discrete space
- Configuration space

These parts will be discussed below in further detail.

i.1 Scenario

To ultimately be able to provide an AI controller for automated parking, one needs to define a scenario to be realistic and applicable, yet also prevent it to be more complex than necessary.

This report will consider two scenarios. One setting for perpendicular parking and one setting for parallel parking. In either of the cases 3 parking spaces are simulated with the centre parking space being free. On top of that, the driving lane adjacent to the parking spots is also added, along with some margins to allow the AI to slightly cross into the other lane and have some freedom of movement. The resulting parking scenarios after discretization, which is discussed in i.3, are shown in figure 1.

i.2 Car Model

The car's dynamic model consists of three states: x,y-coordinates and an angle θ. Using these coordinates one can determine the next state of the car after a control input u. The car is assumed to be a point in its workspace and the car will have no model for its acceleration/breaking behaviour, rather it will simply move a predetermined distance and angle for a time-step Δt after a certain input u. After an input u is chosen, the car will move circularly over a curve defined by the turning radius of an small car: 5 meters. [2]

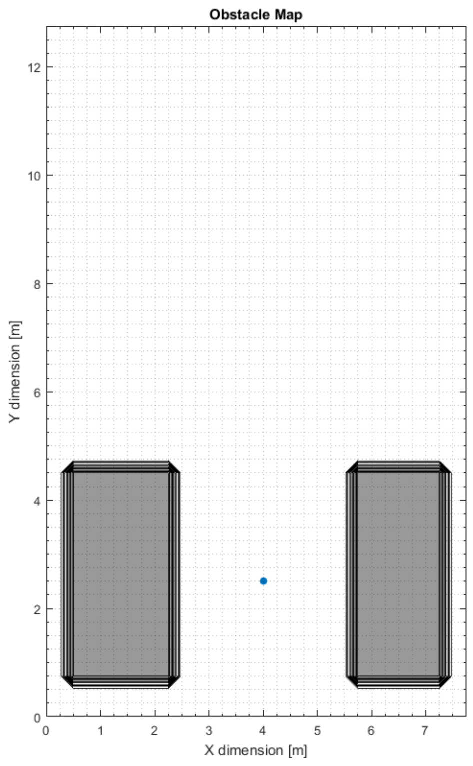
The available inputs u of the model are: $[forwards, backwards]$ and $[left, right]$. In this report these are binary inputs, meaning that the combination of these two gives only 4 possible combinations of said inputs, for example: $u(forwards, right) = 2$. This is shown in table 1. No gradient in speed or turning angle is possible. This is done to reduce the state-action space to a bare minimum by only having 4 possible actions and ultimately save on computational complexity.

This model will supply the new state of the car in *continuous space*.

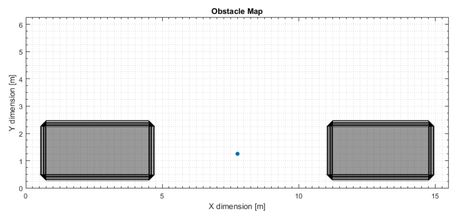
i.3 Mapping into Discrete Space

The curse of dimensionality prevents the use of a high resolution discrete space. Therefore,

[2] An average car has a turning radius of 10 meters, however this resulted in having to have a high accuracy in the discretization to allow the precise movements. This resulted in too high of a computational complexity and therefore the turning radius was set to 5 meters to reduce the state space size.

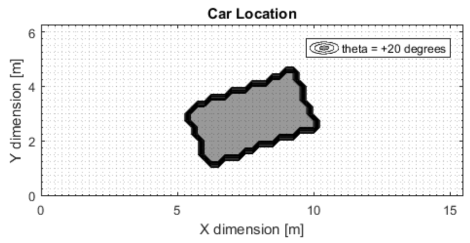


(a) *perpendicular parking scenario*

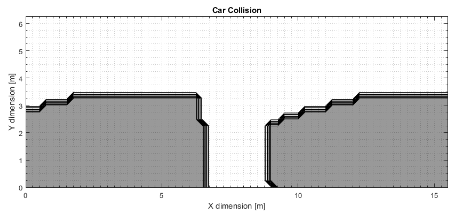


(b) *parallel parking scenario*

Figure 1: *The resulting scenarios after discretization*



(a) *car at an angle in the parallel parking scenario*



(b) *Collision map when car is reduced to a point for given angle*

Figure 2: *Collision map*

‘

Table 1: *Available actions u*

Action Number	Left	Right
Forwards	1	2
Backwards	3	4

a trade-off has to be made between accuracy and computing power.

The workspace will be discretized orthogonally and equally in both the x as y directions, with a different scale for the discretization of the angle θ. It was chosen to have a resolution of 0.5 meters and 10 degrees[3]. The resulting discretization error of 0.15 meters for x and y is rather coarse for a task of automated parking, however this is a necessary measure to prevent extreme computation times. The scenario described in i.1 is shown in discrete space in figure 1.

The resulting new state supplied by the car model after an action u was chosen in i.2 by the AI in ii is now rounded by this mapping into the discrete state space.

i.4 Configuration Space

It is possible and justifiable to reduce the car's size to a point to describe its dynamics, however it is not when determining its collisions with objects. In such a case, one needs to map as accurately as possible its collisions with the obstacles and bounds.

[3]These values are based on preliminary sizing of the discrete state space to prevent it becoming too large.

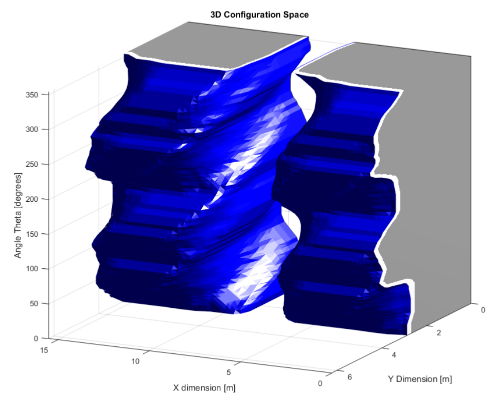


Figure 3: *3D Configuration Space for parallel parking*

Wether a car will collide with obstacles is dependent on a car's orientation. One can imagine that a car orientated in parallel to a parked car can be located closer to the parked car (in terms of distance between the centre of masses) than a car being orthogonal to the parked car. This difference needs to be taken into account. An example of the discretized collision space of a car oriented at +20 degrees is shown in figure 2.

Using this information one can now compute the configuration space and define collisions for every possible angle θ, across the entire workspace. The configuration space in 3D is shown for the parallel parking scenario in figure 3. Any point located inside the volumes represents an illegal state, a collision.

The process is identical to generate the configuration space for the perpendicular scenario.

ii. AI Design

Reinforcement learning is based on trial and error. The AI will at first perform random actions and if the AI gets rewarded it remembers it. Different algorithms apply different techniques for "remembering". There are three steps in the pipeline for a discrete, model-free, learning algorithm: The policy, the state or state-action value algorithm and the reward function. Each of these play a crucial role in ability of the AI to optimally choose actions from its current state.

The different steps in the AI's algorithm are the policy(ii.2), the learner(ii.3) and the reward(ii.4). They are each discussed separately below, but firstly a justification is supplied on why a model-free learner was chosen(ii.1).

ii.1 Model-free Reinforcement Learning

Reinforcement learning can be applied in various contexts. At the start of the design one needs to decide on if the transition model from one state to the other should be known to the AI or not. Should the AI know in advance which action will lead to which state?

In this context it was decided to make the AI model-free. This is done to allow a continuous-space dynamic model which is disconnected from the discrete-space of the AI. This also more accurately reflects reality since one can now model the dynamics more closely to reality. In future work one can expand on this model with stochastic models of state transition and implement velocity (for example) as an additional state which would require minimal modifications to the AI's algorithm. These expansions are not done in this report, however.

This results in that only model-free algorithms can be chosen.

ii.2 Policy

The policy of the AI determines the action that the AI will choose. It is the algorithm that reads the current state and the current state-value or state-action-value (explained in ii.3) and determines the next action. This action does not have to be the optimal action for a given state and state(-action)-value. This report focuses on a single type of policy, namely the ϵ-greedy policy.

The ϵ-greedy policy is an evaluator that will perform a random action, regardless of the situation an ϵ-percentage of the time. This ϵ-policy will override the greedy policy. The greedy part of the policy will choose the optimal action for a given situation. Optimal is defined as the action which will result in the highest value for the next state.

ii.3 Learning Algorithm

The learning algorithm describes in what way the received reward from the Reward Function (ii.4) is processed and stored. There will be three different algorithms of learners compared in this report. The following paragraph each discuss the different learners and their differences compared to each other.

Q-Learning This is a learning algorithm which considers its current value, the reward of the next state and the value of the next state-action pair to induce the value of the current state-action pair. For example, in state $'a'$ the policy chooses action $'u'$. The value of this state-action is dependent on what its value was in the previous iteration, the reward of the resulting state and value of the next state-action. The action is determined by which action has the highest state-action value, based on previous iterations. In function form this is described as shown in equation 1.

$$\begin{aligned} Q(s_t, a_t) = (1 - \alpha) Q(s_t, a_t) ... \\ + \alpha(R_t + \gamma \max_a Q(s_{t+1}, a)) \end{aligned} \tag{1}$$

Here the variable α describes the learning rate. It is the ratio between how much of value it will retain from previous iterations and how much it will take over values from the reward and next state-action. The other parameter is γ. This parameter describes how much weight is given to the value of the next state-action pair, if this weight is low, then the next state-action pair will return a low value after multiplication with γ. This results in that the current reward R_t will, in ratio, weigh in more than future states. This makes the state-action matrix Q be more influenced by local rewards. Thus γ is to be chosen such that the desired balance is found between short and long term rewards.

SARSA State-Action-Reward-State-Action is a modification of Q-learning. Instead of assuming that the agent will pick the action that has the highest value, it first computes what the agent would do in the next state. This does not necessarily needs to be the action with the highest value. This is so called on-policy, since SARSA considers the agents policy in the evaluation of the state-action values. The expression for SARSA is given in equation 2.

$$\begin{aligned} Q(s_t, a_t) = Q(s_t, a_t) ... \\ + \alpha(R_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \end{aligned} \tag{2}$$

So the value of $Q(s_t, a_t)$ is dependent of the next action. furthermore, $Q(s_t, a_t)$ updates based on the difference between $Q(s_{t+1}, a_{t+1})$ $Q(s_t, a_t)$ in addition to possible rewards.

TD(λ) Q-learning is a special form of Temporal Difference mapping, namely TD(0). It only considers the current state in its evaluation for the previous state. This can be generalised to a form where it can update an λ amount of previous states based on its current rewards and state-action value. It has to remember its previous actions to do so.

ii.4 Reward Function

This function determines for which state the learning algorithm should be rewarded. It can also be negative to give a penalty. They provide the objective for the AI to achieve.

In this paper the reward is granted when the AI has reached the optimal parking spot with the correct orientation.

III. Results

The results of the simulations show that it is able to find the shortest path if given sufficient time. The convergence is shown in figure 4. It concerns the case of perpendicular parking for each individual learner. After which, the tuning of the hyper-parameters is discussed and the performance is analysed.

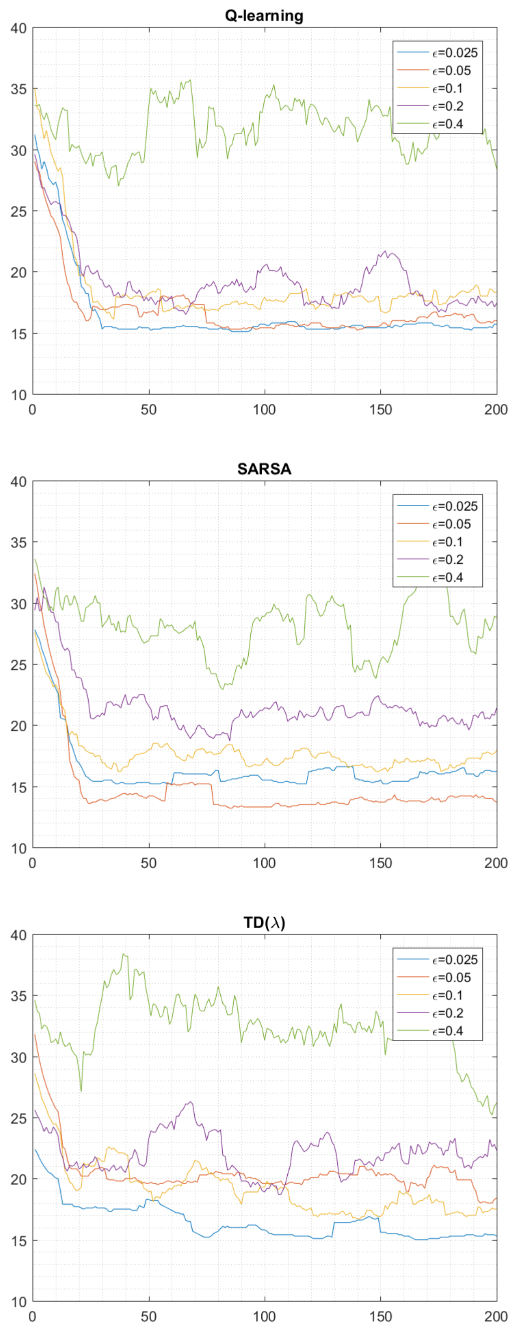


Figure 4: *Perpendicular parking scenario. Path length over iterations, moving average taken over 20 iterations. Every learner shown per sub-figure and tested over varying values of ϵ.* ($\gamma = 0.9, \alpha = 0.2$)

Optimal ϵ From the graph one can observe that setting the ϵ-value too high is detrimental to the convergence of the AI. $\epsilon = 0.4$ consistently performs worse than any other tested value. Another bad performed is $\epsilon = 0.2$. This suggests that these values are still to high.

Another observation one can make is that the optimal performer seems to differ per learning algorithm. Q-learning and TD(λ)[4] performs best with $\epsilon = 0.025$ while SARSA does better with $\epsilon = 0.05$.

From the second sub-figure of figure 4 one can see that SARSA($\epsilon = 0.05$) performs best out of all three by reaching the shortest path length sooner than the other two learners and remaining stable at a lower value. Q-learning and TD(λ) have consistently a higher moving average and converged to this higher average slower. Both these criteria make SARSA($\epsilon = 0.05$) the best learner for this scenario.

Optimal γ Using the findings of the previous paragraph, the optimal γ is chosen by fixing $\epsilon = 0.05$. Figure 5 shows these results.The optimal value for this parameter seems to differ for each learner. for Q-learning the optimal is when $\gamma = 0.6$, for SARSA $\gamma = 0.9$ and for TD(λ) $\gamma = 0.3$.

TD(λ)'s case seems to infer that a long history of actions and visited states, which a high λ would have, is not beneficial to its performance. Only a short-term memory gives a good performance, but still worse compared to Q-learning and SARSA.

Optimal Learner Now one can see that Q-learning($\gamma = 0.6 \land \epsilon = 0.05$) and SARSA($\gamma = 0.9 \land \epsilon = 0.05$) perform well according to the criteria of convergence speed and stability once converged. The comparison of these two curves is shown in figure 6.

[4] $\lambda = 4$, in this report the variation of this value was not tested. A seemingly intuitive value for lambda was chosen without a strong mathematical basis since this is beyond the scope of this report.

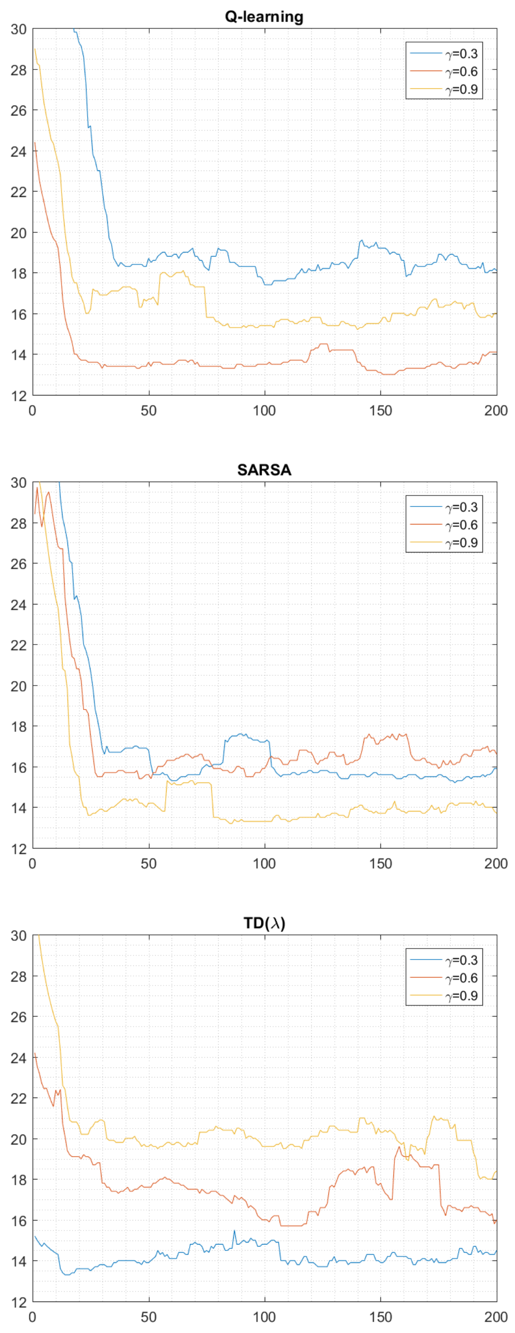


Figure 5: *Perpendicular parking scenario. Path length over iterations, moving average taken over 20 iterations. Every learner shown per sub-figure and tested over varying values of γ. ($\epsilon = 0.05, \alpha = 0.2$)*

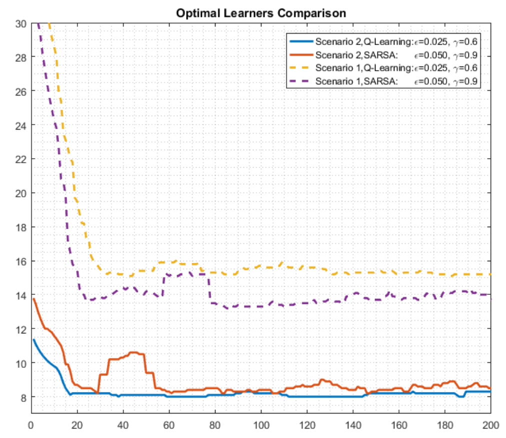


Figure 6: *Path length over iterations. Direct comparison of the two optimal learners. Both scenario 1 (=perpendicular parking) and 2 (=parallel parking) are compared.*

From figure 6 one can see that the two learners are relatively equal in performance. SARSA performs best in scenario 1, but Q-learning performs better in scenario 2.

There are noticeable 'bumps' in the figure for scenario 1 and 2 of SARSA. These are artefacts from the moving average sliding window. Since the original signal is noisy, on the off-chance that the algorithm takes for example 950 steps before reaching the target, it will 'bump' the average up for the duration of the sliding window.This could be solved in the future by removing outliers.

A final figure showing how the AI travels through the configuration space using the optimal route for the parallel parking scenario is shown in figure 7.

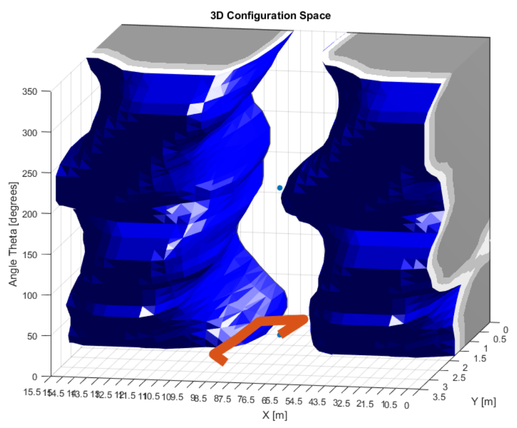


Figure 7: *Parallel parking scenario. 3D space of [x,y,θ] shows which states the agent visits while travelling to the target. Blue dots signify target states which would incur a reward, the blue solids show the collision space. Rewards at* $\theta = 0 \vee \theta = 180$ *degrees.*

IV. Conclusion

The AI is able to park the car. In both scenarios it managed to converge to a minimum path length within a finite amount of time for either of the simulated scenarios.Since these algorithms behaves the same regardless of discretization scale, one can infer that such an AI will also be able to park given a more precise discretization. The only limitation being computing power. The following is split into two parts, the Algorithm section to discuss the details of the algorithm and after that the real-world applicability is discussed.

i. Algorithm

Optimal Learner The optimal learner proved to be either SARSA or Q-Learning. Which of these is optimal is dependant on the scenario and the hyperparameters chosen.

The Temporal Difference learner was performing worse. This could be due to improper tuning of the variable λ. Since a value for λ was chosen arbitrarily, one cannot completely rule out this algorithm and conclude that this algorithm will perform worse every time.

Hyper-parameters The hyper-parameters proved to play a crucial role in the convergence of the learning parameters. An improper tuning of these parameters will cause the learning algorithms to perform sub-optimally. Therefore, regardless of which algorithm is chosen, its performance is only good if such parameters are tuned accordingly.

ii. Real-world Applicability

The discretization error of the chosen simulation environment is significant and therefore difficult to be put into practice. Having a car parked with an accuracy of ± 0.25 meters is too inaccurate for the real world. One would desire close to *cm*-level accuracy for such an AI. The discretization will lead to obstacles and the car being rounded away, leading to a cumulative inaccuracy of up to 0.5 meter. This inaccuracy drives the chances of collision up. This is an unacceptable to risk and therefore such an AI needs to be trained on a higher precision and penalised for coming too close to other objects.

The other limiting factor which prevents application in the real world is computation time. For a sufficiently accurate discretization, the computational time increased drastically. A car cannot stand still in front of a parking lot for over 10 minutes while it calculates the optimal route to take.

V. Discussion

A lot of limitations had to be applied to this report to reach the goal. The largest concession made was the loss of real-world applicability due to either a highly inaccurate model or too much computation time. These two orthogonal problems were balanced and a model was chosen which was in between these two problems. The results are therefore intuitive, yet not ideal.

Furthermore, once either more efficient algorithms have been developed or computational power has increased then these algorithms will become useful for real-world applications.

References

[1] Darío Maravall, Miguel Ángel Patricio, and Javier De Lope. Automatic car parking: A reinforcement learning approach. *Computational Methods in Neural Modeling Lecture Notes in Computer Science*, page 214–221, 2003.