


# Double Q-Learning for a Simple Parking Problem: Propositions of Reward Functions and State Representations

Przemysław Klęsk<sup>1</sup>

Faculty of Computer Science and Information Technology, West Pomeranian University of Technology in Szczecin,  
ul. Żołnierska 49, 71-210 Szczecin, Poland

**Keywords:** Car Parking, Double Q-Learning, Neural Approximations, Parameterized Reward Functions, State Representations.

**Abstract:** We consider a simple parking problem where the goal for the learning agent is to park the car from a range of initial random positions to a target place with front and back end-points distinguished, without obstacles in the scene but with an imposed time regime, e.g. 25 s. It is a sequential decision problem with a continuous state space and a high frequency of decisions to be taken. We employ the double Q-learning computational approach, using the bang–bang control and neural approximations for the  $Q$  functions. Our main focus is laid on the design of rewards and state representations for this problem. We propose a family of parameterized reward functions that include, in particular, a penalty for the so-called “gutter distance”. We also study several variants of vector state representations that (apart from observing velocity and direction) relate some key points on the car with key points in the park place. We show that a suitable combination of the state representation and rewards can effectively guide the agent towards better trajectories. Thereby, the learning procedure can be carried out within a reasonably small number of episodes, resulting in high success rate at the testing stage.

## 1 INTRODUCTION

Driving a car can be seen as a *sequential decision problem* that can be subjected to reinforcement learning (RL) algorithms, Q-learning in particular. Decisions in such a task are commonly called “micro-decisions” because of their high frequency<sup>1</sup>. Human drivers also take many small decisions in order to: correct velocity or direction, glance in the mirror, start to brake, react to other vehicles movement, etc. Some of those are half-conscious or reflexive decisions (Sall et al., 2019; Sprenger, 2022).

In this paper we consider a simplified variant of the car parking problem, where the goal for the learning agent is to drive the car from an initial position (drawn from a certain random distribution) to the target park place with front and back end-points distinguished, and to fully stop the car there. No obstacles are present in the scene but the task must be completed within an imposed time limit, e.g. 25 s. Fig. 1 provides an example illustration of this problem setting — the non-filled light blue rectangle represents

the park place (with ‘F’ and ‘B’ letters denoting its front and back) and the dashed border marks the region of random initial positions for the car<sup>2</sup>.

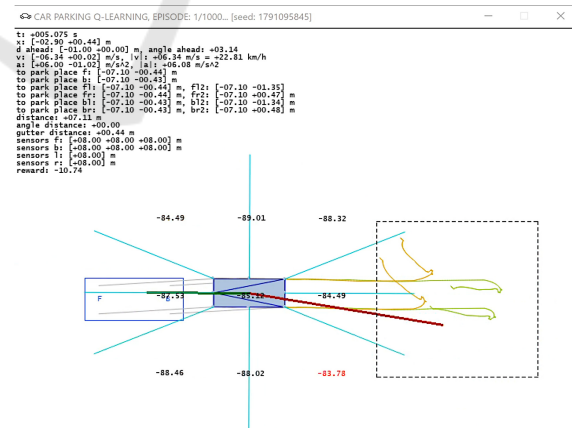


Figure 1: Example illustration of the simple car parking problem.

We implement the *bang–bang* control scenario in this paper. This means that there exists a finite set

<sup>2</sup>The car’s initial direction angle is also drawn from a certain random range.

<sup>1</sup><https://orcid.org/0000-0002-5579-187X>

<sup>1</sup>Despite the name, gaps between decision moments are not necessarily at the level of microseconds.

of acceleration vectors (or equivalently force vectors) that can be applied on the car at decision moments. An applied acceleration can be either completely on or off. Fractional application is not permitted. In Fig. 1, the dark green line is the current velocity vector — the car is traveling roughly forwards (to the left of the image); whereas the red line, directed almost oppositely, represents the imposed acceleration vector — the car starts to brake and gently turn left as it is approaching the park place. Light orange and green curves mark traces of front and back wheels, respectively. Gray lines represent vectors pointing from the car corners to their ideal positions in the park place.

The general computational approach that we have employed in the research underlying this paper is compliant with the *double Q-learning* technique (Hasselt, 2010; Hasselt et al., 2016). This means there are two approximators of the  $Q$  function involved. We use several-layers-deep, dense neural networks with ReLU activations as approximators. One network is used for making decisions during the parking attempts, and simultaneously for learning via *experience replay*. Weights of that network are updated quite frequently. The second network, with the same structure, stays frozen for long periods and plays the role of a “pseudo-oracle”. It serves the purpose of providing stable target values for the supervised learning of the first network. After suitably many training steps, the target network gets switched — that is, the weights from the online network are copied to the target network.

## 1.1 A “Needle in a Haystack” Motivation

The main points of interest in this paper are *reward functions* and *state representations* for the defined parking problem. It should be well understood that in rich environments with continuous state spaces, simple non-informed reward functions do *not* work well. Imagine a car parking agent that starts wandering randomly 10 meters away from the park place. If, for example, that agent is rewarded solely with a  $-\Delta t$  value after each small time step in which he did not succeed to “land” in the park place (e.g.  $-0.1s$  reward for the time consumed per step), then clearly he should not be expected to learn anything. The function being approximated is flat almost everywhere. The fact that the success event requires the car to be accurately placed (with only a small deviation tolerated distance- and angle-wise), and moreover fully stopped, makes such an event extremely unlikely to be discovered accidentally — just like a “needle in a haystack”. Naturally, one might try to extend the  $-\Delta t$

reward with an extra summand that estimates the remaining time needed to reach the target. This could be done e.g. based on the remaining straight-line distance and assuming some average velocity while maneuvering. As we show in the paper, such an extension is only a minor improvement, not sufficient to significantly speed up the learning process.

## 1.2 Main Contribution

In the paper we propose and study a *parameterized reward function* for the parking problem, with three penalty terms pertaining to: (a) the straight-line distance, (b) the angular deviation between direction vectors of the car and the park place, and (c) an additional quantity named the “*gutter distance*”. We indicate good proportions between those terms, represented by suitable penalty coefficients discovered in experiments.

Furthermore, we experiment with several variants of *state representations with information redundancy* that constitute the input to neural models. Apart from the car direction and velocity, the representations include additional vector-based information that relates key points on the car with key points in the park place. Our results demonstrate that a suitable combination of the state representation and rewards can effectively guide the parking agent towards better trajectories. Thereby, the learning quality is improved and the process can be carried out within a reasonably small number of episodes.

## 2 PRELIMINARIES

### 2.1 $Q$ Functions

Given a policy  $\mathcal{P}$  — a mapping from the set of states to the set of actions, the  $Q_{\mathcal{P}}(s, a)$  function is defined to return the *expected* sum of all future rewards  $R_{t+1}, R_{t+2}, \dots$  (random variables), discounted exponentially by a decay rate  $\gamma \in [0, 1)$ , conditional on the fact that at the starting state  $S_t = s$  one performs action  $a$ , and from thereafter follows the policy  $\mathcal{P}$ :<sup>3</sup>

$$Q_{\mathcal{P}}(s, a) = \mathbb{E}(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s, A_t = a; \mathcal{P}). \quad (1)$$

This means that for all attained subsequent states  $s_{t+k}$ ,  $k = 1, 2, \dots$ , the agent takes actions  $a_{t+k}$  yielded by the policy:  $a_{t+k} = \mathcal{P}(s_{t+k})$ , whereas the initial action

<sup>3</sup>Capital letters under the expectation  $\mathbb{E}(\cdot)$  in (1) represent random variables and lowercase ones their realizations i.e. the values attained.

$a$  can be arbitrary, not necessarily compliant with the policy.

In environments with stochastic transitions and rewards, an action  $a$  performed by the agent in a state  $s$  can result in various pairs  $(r, s')$  of the reward value and the next state; sometimes differing only slightly. In such a case, the recursive definition of the optimal action value  $Q^*(s, a)$ , known as the Bellman optimality equation, takes into account the probabilities (or densities) of transitions, and is:

$$Q^*(s, a) = \sum_r \sum_{s'} P(r, s' | s, a) \left( r + \gamma \max_{a'} Q^*(s', a') \right), \text{ or} \quad (2)$$

$$Q^*(s, a) = \iint_{r, s'} p(r, s' | s, a) \left( r + \gamma \max_{a'} Q^*(s', a') \right) dr ds'. \quad (3)$$

In deterministic environments, a fixed  $(s, a)$  pair always produces a specific next pair of  $(r, s')$ , and the Bellman equation reduces to:

$$Q^*(s, a) = r(s') + \gamma \max_{a'} Q^*(s', a'), \quad (4)$$

where for our purposes we choose to write  $r(s')$ , understood as  $r(s') \equiv r(s, a)$ , to represent the fact that our deterministic rewards shall be implied by the reached state  $s'$  alone.

## 2.2 Double Q-Learning

In contrast to small grid environments, where  $Q$  values can be updated in lookup tables inductively without special risks, reinforcement learning for larger problems, based on  $Q$  function approximation, requires caution due to potential violations of the elementary i.i.d. principle of machine learning. It states that data examples ought to be independent and identically distributed. There are three elements that may contribute to the violation of i.i.d.

Firstly, consecutive states along a certain trajectory are obviously not independent, but highly correlated. An approach that diminishes this problem is *experience replay* (Mnih et al., 2013; Mnih et al., 2015; Hasselt et al., 2016). It collects the experience quadruplets — state, action, reward, next state — into a large buffer, and from time to time triggers the training based on a random data subsample drawn from this buffer independently and with repetitions (a bootstrap batch). This effectively decorrelates the data.

Secondly, suppose that  $\widehat{Q}(s, a; \widehat{w})$  denotes our working model parameterized by weights  $\widehat{w}$ . With this model we would like to approximate  $Q^*(s, a)$  for all  $(s, a)$ , i.e. to have  $\widehat{Q}(s, a; \widehat{w}) \approx Q^*(s, a)$ . Suppose

also that  $(s_i, a_i, r_i, s'_i)$  is a single  $i$ -th experience drawn from our buffer. To train  $\widehat{Q}$  using this experience, one needs to prepare a certain target value  $y_i^*$  for the input pair  $(s_i, a_i)$ , so that the supervised regression task can be performed. Mathematically, such a target value is implied by the right-hand-side of Bellman equation (4) — the ideal target value. Therefore, it may seem the task is to minimize the following squared error

$$\left( \widehat{Q}(s_i, a_i; \widehat{w}) - \underbrace{(r_i + \gamma \max_{a'} Q^*(s'_i, a'))}_{y_i^*} \right)^2 \quad (5)$$

with respect to  $\widehat{w}$ . Doing so, e.g. by means of the stochastic gradient descent, would mean to update weights e.g. as follows:

$$\widehat{w} := \widehat{w} - \eta \left( \widehat{Q}(s_i, a_i; \widehat{w}) - (r_i + \gamma \max_{a'} Q^*(s'_i, a')) \right) \cdot \nabla_{\widehat{w}} \widehat{Q}(s_i, a_i; \widehat{w}), \quad (6)$$

where  $\eta \in (0, 1]$  stands for a learning rate constant and  $\nabla_{\widehat{w}} \widehat{Q}(\cdot)$  for the gradient of our approximator. Yet, in practice it is impossible to prepare the target  $y_i^*$  as shown in (5), because true oracles  $Q^*$  do not exist and we simply do not know the value of  $Q^*(s'_i, a')$ . . . One could be tempted to set  $y_i^* := r_i + \gamma \max_{a'} \widehat{Q}(s'_i, a'; \widehat{w})$  instead. But that would mean that the target values one tries to approximate with model  $\widehat{Q}$  are themselves computed using  $\widehat{Q}$ . This phenomenon, known as a ‘pursuit of non-stationary target’, may lead to an unstable learning process.

The idea behind *double Q-learning* solves this problem by introducing the second model, say  $\widetilde{Q}(\cdot; \widetilde{w})$ , that can be regarded as a pseudo-oracle serving the purpose of providing a stable second summand for the target values as follows (Hasselt, 2010; Hasselt et al., 2016; Mnih et al., 2013):

$$y_i^* := r_i + \gamma \max_{a'} \widetilde{Q}(s'_i, a'; \widetilde{w}). \quad (7)$$

For neural networks applied as approximators, the  $\widetilde{Q}$  model is often referred to as the ‘target network’, while the main operating model  $\widehat{Q}$  as the ‘online network’.

Two approaches for updating  $\widetilde{Q}$  are met in practice. In the first one,  $\widetilde{Q}$  stays frozen for a long period (so that many training steps of  $\widehat{Q}$  can take place using stable targets) and then  $\widetilde{Q}$  gets replaced by making a hard switch of weights:  $\widetilde{w} := \widehat{w}$ . In the second approach,  $\widetilde{Q}$  is updated slowly, but continuously, towards  $\widehat{Q}$  via a moving average, e.g.  $\widetilde{w} := 0.999 \widetilde{w} + 0.001 \widehat{w}$ . This way or another, the idea with two models stabilizes significantly the learning process (Fujita et al., 2021; Kobayashi and Ilboudo, 2021).

The third element that endangers the i.i.d. is more subtle. It pertains to the max operator present in (7), which under the stochastic setting is *biased* towards overestimations of future rewards. Both mathematical and empirical evidence support this claim (Thrun and Schwartz, 1993; Hasselt, 2010; Hasselt et al., 2016; Sutton and Barto, 2018). The following simple, yet elegant, trick circumvents the problem. Noting that (7) can be equivalently rewritten as

$$y_i^* := r_i + \gamma \tilde{Q}\left(s'_i, \arg \max_{a'} \tilde{Q}(s'_i, a'; \tilde{w}); \tilde{w}\right), \quad (8)$$

one prefers instead to use the following estimation of the target value:

$$y_i^* := r_i + \gamma \tilde{Q}\left(s'_i, \arg \max_{a'} \hat{Q}(s'_i, a'; \hat{w}); \tilde{w}\right). \quad (9)$$

It means the online model  $\hat{Q}$  is still used to pick the ‘best’ next action, but the value of that action is estimated with the target model  $\tilde{Q}$ . An apt analogy to it is a situation where two measurement instruments (e.g. scales for measuring weights in kilograms), each biased with a certain stochastic error, are used to discover a maximum value over a certain collection of objects. Using one instrument to indicate the ‘arg max’ and the other to indicate the value for this argument is a practical protection against the statistical overestimation.

### 3 RELEVANT WORK

First of all, it should be clearly remarked that our direct research interests are *not* located in the domain of AGVs (autonomous ground vehicles) but in reinforcement learning itself. We focus on the design of reward functions in RL problems, where an immediate reward signal is not present or is extremely sparse (“needle in a haystack”). Therefore, the simple parking problem should be treated as a toy example serving that purpose in this research. Readers directly interested in recent advances in the realm of AGVs and APSs (automatic parking systems) can be addressed e.g. to two works (Zeng et al., 2019; Cai et al., 2022) that employ geometric methods and control theory, or to (Chai et al., 2022; Li et al., 2018) where neural networks are applied to approximate near-optimal trajectories; the latter involves also RRTs (Rapidly-exploring Random Trees).

As regards the general topic of rewards in RL (without connection to the parking problem), worth attention is a recent paper due to Sowerby, Zhou and Littman (Sowerby et al., 2022). It discusses how reward-design choices impact the learning process

and tries to identify principles of good reward design that quickly induce target behaviors. Moreover, using concepts of *action gap* and *subjective discount*, the authors propose a linear programming technique to find optimal rewards for tabular environments.

As regards works with similarity to ours and pertaining to both contexts — rewards and RL-based parking — the following two can be pointed: (Aditya et al., 2023; Zhang et al., 2019). In (Aditya et al., 2023), Aditya et al. also employ the double Q-learning. Their simulated parking lot contains obstacles (other cars parked), but it has a fixed layout and a predefined loop path along which the car cruises before it finds a free spot. Only then, the actual RL-based parking maneuvers begin. The reward function from (Aditya et al., 2023) involves distance and angle deviation and decays exponentially as those quantities increase. The authors report 95% success rate after 24 hours of training (Intel Core i5-1135G7 CPU, IRIS Xe GPU). Unfortunately, no details on the neural network structure are provided. In (Zhang et al., 2019), Zhang et al. apply the policy gradient approach using a two-layer deep network with 100 and 200 ReLU-activated neurons, respectively. Their hand-crafted reward function contains four pieces of information: closeness to the slot center, ‘parallelism’ of car and park main axes, penalties for line-pressing and side deviations. Experiments include scenes with 60°, 45° and 30° initial angles between the car and the parkplace. The reported results focus on final deviations from the wanted goal position which range from about 0.37 m to 0.48 m sideways and are up to  $\approx 1$  m lengthwise.

Obviously, due to different environment settings and the problem itself, it is impossible to fairly compare our results against the ones from (Aditya et al., 2023; Zhang et al., 2019). On one hand, the lack of obstacles is a clear simplification in our problem variant; on the other, we take into account a demanding time limit and a broader range of initial angles and positions (some of them remote) — hence, an environment that requires more exploration from the RL perspective and more capable models to suitably approximate  $Q$  values over a richer state space. We remind that in (Aditya et al., 2023; Zhang et al., 2019) the actual parking maneuvers start in the close proximity of the goal. Yet, if despite all differences the raw numbers were to be compared, then we can report obtaining 7 models with success rates exceeding 95% (two of them reaching 99%) having the final deviations of at most 41 cm for both axes. Each model was trained for only 10 k episodes and  $\approx 5$  h (all details in Section 7).

We believe an important distinction of our research is that we do *not* work with a single reward function with fixed coefficients as it was the case in (Aditya et al., 2023; Zhang et al., 2019) (a dogmatic approach). Instead, we work with a parameterized family of reward functions and discover good proportions between the involved quantities over multiple experiments.

## 4 NOTATION, SETTINGS, PHYSICS

We start this section by providing the link to the repository of the project associated with this paper: <https://github.com/pklesk/qlparking>. It provides access to source codes, logs for selected models and videos with parking maneuvers. The implementation has been done in the Python programming language. All settings pertaining to physics and car behavior are defined in the script `/src/defs.py`.

Our car model for simulations is of length  $l = 4.405$  m and width  $w = 1.818$  m.  $\mu_0 = 0.6$  and  $\mu_1 = 0.3$  denote its coefficients of static and kinetic friction, respectively. We do not specify the car's mass, since we are going to operate only on accelerations rather than forces (masses not needed). The gravitational acceleration constant is denoted by  $g$  and for computations equal to  $9.80665$  m/s<sup>2</sup>.

The complete information about the car's position and orientation is defined by its current center point  $\vec{x}$  and a unit direction vector  $\vec{d}$  (ahead direction at which the car is looking). Additionally, where useful, we write  $\vec{d}_R$  to denote the unit vector pointing to the right of the car.  $\vec{d}$  becomes  $\vec{d}_R$  after 90° clockwise rotation. The car's velocity vector is denoted by  $\vec{v}$ . All of the above are planar vectors (with two Cartesian coordinates), which means the surface of simulated scenes is treated as flat. Where mathematically needed, the notation additionally includes a time subscript e.g.  $\vec{x}_t, \vec{d}_t, \vec{v}_t$ , or is overloaded to writings such as  $\vec{x}_{|s}, \vec{d}_{|s}, \vec{v}_{|s}$ , to be read as 'in state  $s$ ', if we need to explicitly emphasize the dependence on a state.

As regards the park place, it is of length  $l_p = 6.10$  m and width  $w_p = 2.74$  m, with position and orientation described, analogically, by vectors  $\vec{x}_p, \vec{d}_p, \vec{d}_{p,R}$ , remaining constant in time (the park place does not move). Therefore, the ideal position for the car to be parked at is described by the following corner

points:

$$\vec{x}_{p,FL} = \vec{x}_p + 0.5l\vec{d}_p - 0.5w\vec{d}_{p,R}, \quad (\text{front-left})$$

$$\vec{x}_{p,FR} = \vec{x}_p + 0.5l\vec{d}_p + 0.5w\vec{d}_{p,R}, \quad (\text{front-right})$$

$$\vec{x}_{p,BL} = \vec{x}_p - 0.5l\vec{d}_p - 0.5w\vec{d}_{p,R}, \quad (\text{back-left})$$

$$\vec{x}_{p,BR} = \vec{x}_p - 0.5l\vec{d}_p + 0.5w\vec{d}_{p,R}. \quad (\text{back-right}).$$

In accordance with the bang–bang control scenario, we define the set of three magnitudes for the forward-backward accelerations:

$$\{a_{-,1} = -7.0 \text{ m/s}^2, a_{0,.} = 0.0 \text{ m/s}^2, a_{+,1} = +8.0 \text{ m/s}^2\}, \quad (10)$$

and three magnitudes for the side accelerations (turning):

$$\{a_{-,1} = -1.0 \text{ m/s}^2, a_{,0} = 0.0 \text{ m/s}^2, a_{,1} = +1.0 \text{ m/s}^2\}. \quad (11)$$

Therefore, the following set of 9 possible *actions* becomes generated via the Cartesian product of the above two sets:

$$\mathcal{A} = \left\{ \vec{a}_{j,k} = a_{j,.}\vec{d} + a_{,k}\vec{d}_R : (j,k) \in \{-1,0,1\} \times \{-1,0,1\} \right\}. \quad (12)$$

More simply, the set of actions can be treated equivalently to 9 symbolic directions  $\{\swarrow, \downarrow, \searrow, \leftarrow, \circ, \rightarrow, \nearrow, \uparrow, \nearrow\}$ , meaning: 'backwards left', 'backwards', ..., 'forwards right', with the empty action 'o' representing no acceleration applied.

The time step chosen for simulations was  $\delta t = 25$  ms. Algorithm 1 presents the basic computations that take place when the time moment becomes switched from  $t$  to  $t + \delta t$ , performed to update the car's velocity, position and direction, based on the applied acceleration vector  $\vec{a}_t \in \mathcal{A}$  with static and kinetic friction taken into account.<sup>4</sup>

The chosen  $\delta t = 25$  ms means that 40 'microdecisions' per second could potentially be taken by the agent. Such a granularity seems a bit too fine for the parking task. One reason is that the imposed accelerations would change very fast and unrealistically under the bang–bang control. The second is that human drivers do not take decisions so frequently. Therefore, to make the agent more human-like, the time gap between steering steps was defined as  $\Delta t = 4\delta t = 100$  ms. This implies ten decisions per second taken by the agent.<sup>5</sup>

<sup>4</sup>To avoid modelling static frictions directed sideways, a simple condition prevents the vehicle from making instantaneous turns at low velocities (so that it does not behave like a military tank): if  $\|v_t\| < 0.75$  m/s (see `defs.CAR_MIN_VELOCITY_TO_TURN`) then only longitudinal components of imposed side components are ignored.

<sup>5</sup>see constants: `QL_DT = 0.025` and `QL_STEERING_GAP_STEPS = 4` in the `/src/main.py`

```

if  $\|\vec{v}_t\| = 0.0$  and  $\|\vec{d}_t\| > 0.0$  then
    |  $\alpha_0 := \min\{\mu_0 g / \|\vec{d}_t\|, 1.0\}$ ;
    |  $\vec{a}_t := (1.0 - \alpha_0) \vec{a}_t$ ;
end
 $\alpha_1 := 0.0$ ;
if  $\|\vec{v}_t\| > 0.0$  then
    |  $\|\vec{v}\| := \|\vec{v}_t + 0.5 \vec{a}_t \delta t\|$ ;
    |  $\alpha_1 := \min\{\mu_1 g \delta t / \|\vec{v}\|, 1.0\}$ ;
end
 $\vec{x}_{t+\delta t} := \vec{x}_t + (1.0 - \alpha_1)(\vec{v}_t \delta t + 0.5 \vec{a}_t \delta t^2)$ ;
 $\vec{v}_{t+\delta t} := (1.0 - \alpha_1)(\vec{v}_t + \vec{a}_t \delta t)$ ;
if  $\|\vec{v}_{t+\delta t}\| > 0.0$  then
    |  $\vec{d}_{t+\delta t} := \text{sgn}(\langle \vec{v}_{t+\delta t}, \vec{d}_t \rangle) \vec{v}_{t+\delta t} / \|\vec{v}_{t+\delta t}\|$ ;
end
    
```

Algorithm 1: Single step of simulation (updates of car physics).

## 5 PARAMETERIZED REWARD FUNCTION

As stated in the motivation, our goal is to propose such a reward function that shall guide the agent towards better trajectories more quickly while performing the Q-learning. Apart from  $-\Delta t$  (time consumed between decision steps) and the remaining distance  $\|\vec{x} - \vec{x}_p\|$ , two more quantities will be taken into account. Moreover, we want our final function to preserve a consistent physical *interpretation* of the “negative time”, and thus be expressible, e.g., in seconds.

First, we define the *angular deviation*  $\phi(\vec{d})$  between direction vectors of the car and the park place, based on their inner product:

$$\phi(\vec{d}) = \arccos(\vec{d}, \vec{d}_p) \in [0, \pi]. \quad (13)$$

$\vec{d}_p$ , being a constant, is not specified as an explicit argument of  $\phi$ .

The car is considered successfully *parked* if the following conditions are met:

$$\|\vec{x} - \vec{x}_p\| \leq 0.15 w_p \text{ and } \phi(\vec{d}) \leq \pi/16 \text{ and } \|\vec{v}\| = 0.0. \quad (14)$$

The tolerance constants<sup>6</sup> present in (14) imply that the car’s center can deviate from the park place center by at most 15% of the place width ( $\approx 41$  cm) and also that the angle between the car’s direction and the optimal direction can deviate by at most  $11.25^\circ$ .

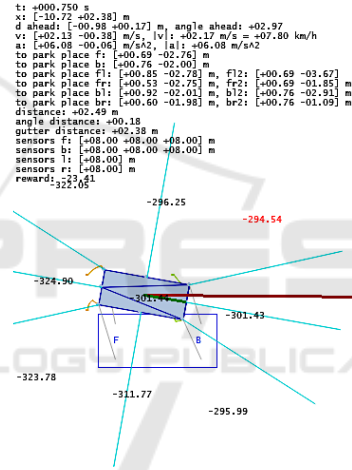
We are now going to define a quantity named the “*gutter distance*”. It represents the distance between the car’s center point and its *projection* onto the axis

defined by the park place<sup>7</sup>, and can be computed as:

$$g(\vec{x}) = \left| -\langle \vec{d}_{p,R}, \vec{x}_p \rangle + \langle \vec{d}_{p,R}, \vec{x} \rangle \right| / \|\vec{d}_{p,R}\| \\ = \left| \langle \vec{d}_{p,R}, \vec{x} - \vec{x}_p \rangle \right|. \quad (15)$$

Intuitively, the presence of this quantity in the reward should help the learning agent distinguish better between less and more promising states as regards the future trajectory needed to complete the task. For example, states where the car is quite far from the park place but approximately on the right track to approach the target (small “gutter distance”) are more promising (Fig. 2b) than states where the car is near the park place but deviated sideways (Fig. 2a). In the latter case, more maneuvering needs to be invested to complete the task (note distances and  $\hat{Q}$  values reported in the figure).

(a) dist.  $\approx 2.49$  m, gutter dist.  $\approx 2.38$  m,  $\max_a \hat{Q}(s, a; \hat{w}) \approx -294.54$



(b) dist.  $\approx 16.11$  m, gutter dist.  $\approx 0.39$  m,  $\max_a \hat{Q}(s, a; \hat{w}) \approx -242.57$

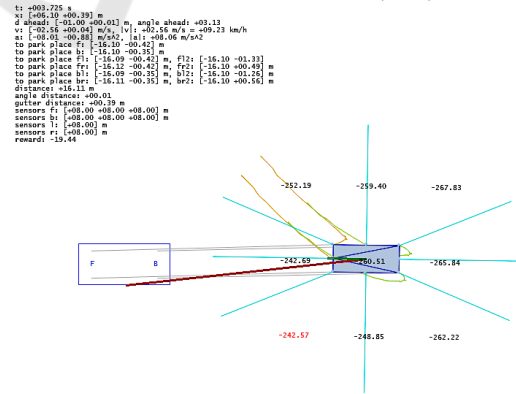


Figure 2: Comparison of two states: (a) with small straight-line distance but future trajectory demanding, (b) with large straight-line distance but future trajectory easy (small gutter distance).

<sup>7</sup>the axis passing through points F (front) and B (back)

<sup>6</sup>defs.CONST\_PARKED\_MAX\_RELATIVE\_DISTANCE\_DEVIATION = 0.15  
 defs.CONST\_PARKED\_MAX\_ANGLE\_DEVIATION = np.pi / 16

Finally, we propose the following **parameterized function** that **rewards** the agent for being in state  $s'$  (after taking action  $a$  in  $s$ ):

$$r(s') = \begin{cases} 0; & \text{if car successfully parked in } s', \\ - \left( \underbrace{\Delta t + \lambda_d \|\vec{x}_{|s'} - \vec{x}_p\|}_{\text{distance}} + \underbrace{\lambda_\phi \frac{\phi(\vec{d}_{|s'})}{\pi}}_{\substack{\text{normed} \\ \text{angular deviation}}} + \underbrace{\lambda_g g(\vec{x}_{|s'})}_{\text{gutter distance}} \right); \end{cases} \quad (16)$$

where parameters  $\lambda_d, \lambda_\phi, \lambda_g \geq 0$  are penalty coefficients. Figures 3, 4 show example plots of (16) when  $\vec{x}_p = (0.0, 0.0)$ ,  $\vec{d}_p = (-1.0, 0.0)$ .

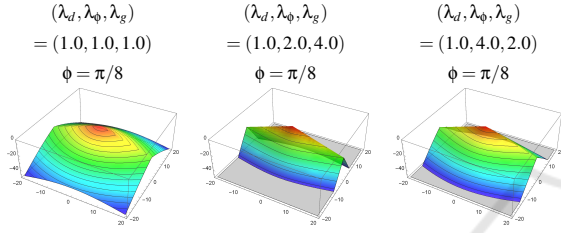


Figure 3: Shapes of reward function (16) for different penalty coefficients  $\lambda_d, \lambda_\phi, \lambda_g$  and a fixed angle  $\phi = \pi/8$ .

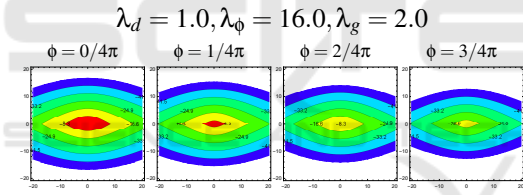


Figure 4: Contours of reward function (16) for different angles  $\phi$  and penalty coefficients fixed to  $\lambda_d = 1.0, \lambda_\phi = 16.0, \lambda_g = 2.0$ .

One should note that  $-\Delta t$  summand, rather than  $-\delta t$ , takes part in (16). This means that for  $s_t = s$ , the actual reward is collected in state  $s' = s_t + \Delta t = s_t + 4\delta t$ , i.e., at the next decision moment, and the intermediate physical states  $s_t + \delta t, s_t + 2\delta t, s_t + 3\delta t$  are not rewarded.

Though a meaningful interpretation of reward is not a must in many RL applications, we note that (16) is interpretable and expressible in time units. If e.g.  $\lambda_d = 2.0$  then the second summand rewritten equivalently to  $\|\vec{x}_{|s'} - \vec{x}_p\| / (1/\lambda_d)$  can be seen as the ratio: distance / velocity, and interpreted as estimated future time needed to cover the remaining distance when traveling on average at velocity  $1/\lambda_d = 0.5$  m/s while maneuvering. Similarly, the reciprocal  $1/\lambda_g$  can be understood as average velocity at which the agent is able to correct one unit of the gutter distance. The smaller that velocity is, the higher the importance of the gutter summand. Deviation  $\phi$  is a unitless real

number, hence  $1/\lambda_\phi$  can be seen as an angular velocity expressed in 1/s units. We decided to norm this quantity to  $[0, 1]$  interval (division by  $\pi$ ). Thus, the  $\lambda_\phi$  itself can be also interpreted as the time required to correct the maximum possible deviation of  $180^\circ$ .

## 6 STATE REPRESENTATIONS

In this section we propose 13 variants of state representations for the considered parking problem. The representations are feature vectors that constitute inputs to neural models  $\widehat{Q}$  and  $\widetilde{Q}$ . The nature and informativeness of inputs obviously have a strong impact on the quality of approximations. Yet, without an experiment it is usually not clear whether particular pieces of information facilitate or hinder the task.

We take under consideration both the globally-oriented features that capture a bird's-eye view of the environment, and locally-oriented features computed relative to the agent. A minimalistic representation could include: the car's heading angle  $\Psi$  (computed with respect to the global coordinate system by means of  $\arctan2$  function), the signed magnitude of velocity along that angle, and vectors  $\vec{f}, \vec{b}$  pointing from the car's front and back central points, respectively, to their counterparts in the park place (relative features):

$$\begin{aligned} \vec{f} &= \vec{x}_p + 0.5l\vec{d}_p - (\vec{x} + 0.5l\vec{d}), \\ \vec{b} &= \vec{x}_p - 0.5l\vec{d}_p - (\vec{x} - 0.5l\vec{d}). \end{aligned}$$

Therefore, the minimalistic representation of a state  $s$  can be:

$$(\Psi, \pm\|\vec{v}\|, \vec{f}, \vec{b})_s. \quad (17)$$

The above compact notation should be understood as a concatenation of two scalars and two vectors, thereby yielding a total of 6 real-numbered features. An equivalent representation but expressed only in terms of vectors could be:

$$(\vec{d}, \vec{v}, \vec{f}, \vec{b})_s. \quad (18)$$

It has 8 features with some redundancy, since  $\vec{d} = \pm\vec{v}/\|\vec{v}\|$ .

Instead of  $\vec{f}, \vec{b}$ , one can introduce four vectors spanning between the corner points of the car and their ideal target positions (also with redundancy), namely:

$$\begin{aligned} \vec{f}_L &= \vec{x}_{p,FL} - \vec{x} - 0.5(l\vec{d} - w\vec{d}_R), \\ \vec{f}_R &= \vec{x}_{p,FR} - \vec{x} - 0.5(l\vec{d} + w\vec{d}_R), \\ \vec{b}_L &= \vec{x}_{p,BL} - \vec{x} + 0.5(l\vec{d} + w\vec{d}_R), \\ \vec{b}_R &= \vec{x}_{p,BR} - \vec{x} + 0.5(l\vec{d} - w\vec{d}_R). \end{aligned} \quad (19)$$

Alternatively, the following variation is also possible:  
 $\vec{f}_{L,2} = \vec{x}_{p,FL} - (\vec{x} + 0.5l\vec{d})$ ,  $\vec{f}_{R,2} = \vec{x}_{p,FR} - (\vec{x} + 0.5l\vec{d})$ ,  
 $\vec{b}_{L,2} = \vec{x}_{p,BL} - (\vec{x} - 0.5l\vec{d})$ ,  $\vec{b}_{R,2} = \vec{x}_{p,BR} - (\vec{x} - 0.5l\vec{d})$ ,  
 (20)

where the vectors point to the corners of the target position but from the central front and central back points on the car. We refer to them as ‘2nd-type’ corner vectors. These vectors provide a useful information when the car is in the last phase of parking — covering final meters along the right track either forwards or backwards. One should realize that in such cases the angles  $\angle(\vec{f}_{L,2}, \vec{f}_{R,2})$  and  $\angle(\vec{b}_{L,2}, \vec{b}_{R,2})$  tend to  $\pi$  i.e.  $180^\circ$ . Note that it is not the case with the first type of corner vectors. Fig. 5 illustrates different types of discussed vectors.

Finally, the vector of features may also be extended with pieces of information that explicitly take part in the reward function (16), namely:  $\|\vec{x} - \vec{x}_p\|, \phi(\vec{d}), g(\vec{x})$ . Their presence potentially “relieves” the neural model from computing them indirectly (in intermediate layers). Yet again, such a claim requires experimental confirmation. Table 1 lists all representations used in experiments.

Table 1: Variants of state representations.

name	state representation (pieces of information in state $s$ )	# of features
avms_fb	$(\psi, \pm\ \vec{v}\ , \vec{f}, \vec{b})_s$	6
dv_fb	$(\vec{d}, \vec{v}, \vec{f}, \vec{b})_s$	8
dv_ffrlblr	$(\vec{d}, \vec{v}, \vec{f}_L, \vec{f}_R, \vec{b}_L, \vec{b}_R)_s$	12
dv_ffrlblr2s	$(\vec{d}, \vec{v}, \vec{f}_{L_2}, \vec{f}_{R_2}, \vec{b}_{L_2}, \vec{b}_{R_2})_s$	12
dv_fb_d	$(\vec{d}, \vec{v}, \vec{f}, \vec{b}, \ \vec{x} - \vec{x}_p\ )_s$	9
dv_ffrlblr_d	$(\vec{d}, \vec{v}, \vec{f}_L, \vec{f}_R, \vec{b}_L, \vec{b}_R, \ \vec{x} - \vec{x}_p\ )_s$	13
dv_ffrlblr2s_d	$(\vec{d}, \vec{v}, \vec{f}_{L_2}, \vec{f}_{R_2}, \vec{b}_{L_2}, \vec{b}_{R_2}, \ \vec{x} - \vec{x}_p\ )_s$	13
dv_fb_da	$(\vec{d}, \vec{v}, \vec{f}, \vec{b}, \ \vec{x} - \vec{x}_p\ , \phi(\vec{d}))_s$	10
dv_ffrlblr_da	$(\vec{d}, \vec{v}, \vec{f}_L, \vec{f}_R, \vec{b}_L, \vec{b}_R, \ \vec{x} - \vec{x}_p\ , \phi(\vec{d}))_s$	14
dv_ffrlblr2s_da	$(\vec{d}, \vec{v}, \vec{f}_{L_2}, \vec{f}_{R_2}, \vec{b}_{L_2}, \vec{b}_{R_2}, \ \vec{x} - \vec{x}_p\ , \phi(\vec{d}))_s$	14
dv_fb_dag	$(\vec{d}, \vec{v}, \vec{f}, \vec{b}, \ \vec{x} - \vec{x}_p\ , \phi(\vec{d}), g(\vec{x}))_s$	11
dv_ffrlblr_dag	$(\vec{d}, \vec{v}, \vec{f}_L, \vec{f}_R, \vec{b}_L, \vec{b}_R, \ \vec{x} - \vec{x}_p\ , \phi(\vec{d}), g(\vec{x}))_s$	15
dv_ffrlblr2s_dag	$(\vec{d}, \vec{v}, \vec{f}_{L_2}, \vec{f}_{R_2}, \vec{b}_{L_2}, \vec{b}_{R_2}, \ \vec{x} - \vec{x}_p\ , \phi(\vec{d}), g(\vec{x}))_s$	15

## 7 MAIN EXPERIMENTS AND RESULTS

**Experimental Setup.** To ensure a fair comparison of different combinations of reward functions and state representation, in all experiments we performed a constant number of 10k episodes (each with the time limit of 25s to park the car) and we applied the same structure for neural models. Training scenes involved a park place with fixed settings of  $\vec{x}_p = (-10.0, 0.0)$ ,  $\vec{d}_p = (-1.0, 0.0)$  and random initial positions of the car drawn from uniform distributions:

$\vec{x} \sim U([5.0, 15.0] \times [-5.0, 5.0])$ ,  $\psi \sim U(\frac{3}{4}\pi, \frac{5}{4}\pi)$ . For testing, we used 1k scenes generated by the same distributions but with different randomization seeds to verify the generalization ability of obtained models. Also, for several best models we performed additional tests where the initial angle was drawn from a broader distribution:  $\psi \sim U(\frac{1}{2}\pi, \frac{3}{2}\pi)$ , hence a range of  $180^\circ$  instead of  $90^\circ$ . Such tests verify the extrapolative generalization ability of models.

In each episode we collected experience quadruplets  $(s, a, r, s')$  separated by time gaps of  $\Delta t = 48t$ . At most 250 experiences could have been collected per episode. Starting from episode 200 onwards, fitting of the online model  $\hat{Q}$  was conducted every 20 episodes. In accordance with experience replay, each fit was based on a bootstrap sample of 65536 experiences drawn from the buffer, with target values for regression computed in compliance with formula (9) using  $\gamma = 0.99$ . A single fit consisted of 1 epoch of Adam SGD corrections (Kingma and Ba, 2014) using minibatches of size 128. Starting from episode 1000 onwards, every 500 episodes we were switching the target network  $\tilde{Q}$  (to become a copy of  $\hat{Q}$ ).

**Modular Structure of Neural Models (Approximators).** To remind, a  $\hat{Q}$  or  $\tilde{Q}$  model takes as its input a vector of features representing a state and returns as the output a vector of predicted (approximated) action values, one value per each action in  $\mathcal{A}$ . Our Python implementation for such models — class named `qapproximations.QMLPRegressor` — works as a wrapper around a collection of standard MLP networks<sup>8</sup>. This means that in our experiments a single  $\hat{Q}$  (or  $\tilde{Q}$ ) model consisted, in fact, of **9 independent substructures**, or equivalently 9 subnetworks, that did *not* share weights; see Fig. 6. Each substructure corresponded to one action in  $\mathcal{A}$  and was fit based on experiences related to that action only. Hence, within calls of `fit(X, y, actions_taken)` method, the mentioned bootstrap samples containing 65536 examples were being partitioned into 9 disjoint subsets (based on the information from the last argument `actions_taken`), and those subsets were still further partitioned into minibatches to perform an epoch of Adam algorithm.

We remark that in RL applications this kind of modular structure for function approximators, sometimes called *multi-head* structure, is not new (Chen et al., 2021; Mankowitz et al., 2018; Goyal et al., 2019), but definitely less popular than the one with shared hidden layers and weights (most of DQNs and DDQNs (Mnih et al., 2013; Mnih et al., 2015; Hasselt et al., 2016)). However, in our opinion, the popu-

<sup>8</sup>sklearn.neural\_network.MLPRegressor (Pedregosa et al., 2011)



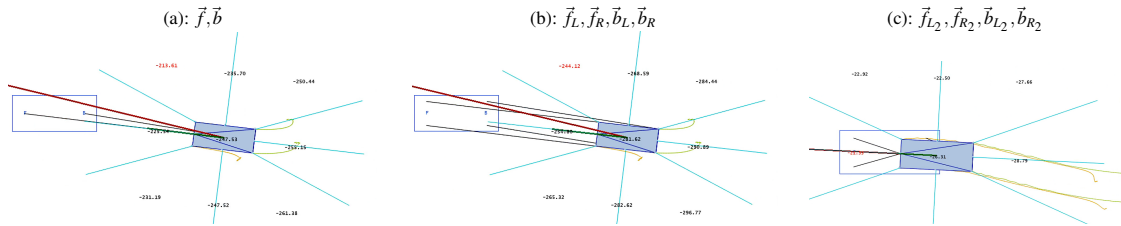


Figure 5: Vectors (black segments) between key points on the car and the park place.

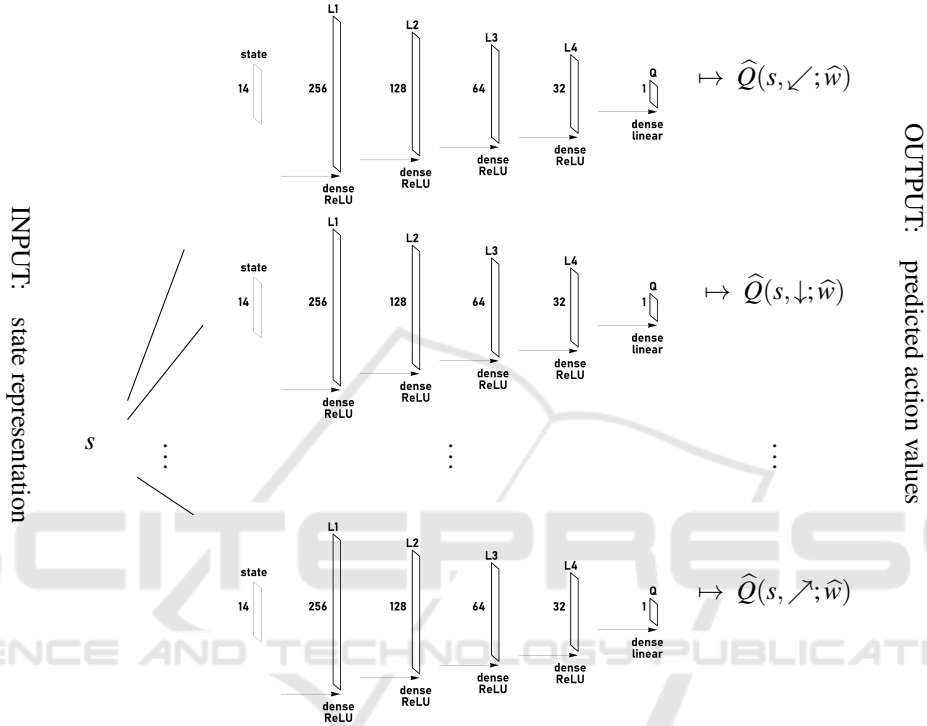


Figure 6: Modular structure of the neural network representing the  $\hat{Q}$  model with 14 inputs (state representation) and consisting of 9 independent substructures, one per each action.

lar structure has certain drawbacks related both to the principles of machine learning and to convergence. The most important one is the ambiguity on how to prepare those entries of target vectors that are associated with *non-taken* actions. Note that only the taken action entry is well-defined by the Bellman equation, e.g. (9).

For the main experiments presented in this section, we employed networks in which the mentioned substructures consisted of 4 hidden dense layers, each, with the following sizes — counts of neurons: (256, 128, 64, 32), and with ReLU activations (Nair and Hinton, 2010), as shown in Fig. 6. In the output layer, the single neurons returning the predicted action values worked as linear combinations of signals from the last hidden layer, i.e. without any non-linear activation. This overall architecture

comprised the total of approximately<sup>9</sup> 0.4 M network parameters (weights). In the case of additional experiments that we discuss in Section 8, we applied larger networks with doubled sizes of hidden layers ( $\approx 1.6$  M weights).

**Agent Behavior.** The agent was picking actions according to the  $\epsilon$ -greedy approach, with the exploration rate  $\epsilon$  (probability of random action) dropping linearly from 0.5 to 0.1 over all 10k episodes. After each 250 episodes, the subsequent episode was presented as a video (Shinners, 2011), in which the agent was taking only the greedy actions:

<sup>9</sup>Number of weights varied slightly due to different state representations, and hence different numbers of input signals.

$\arg \max_{a \in \mathcal{A}} \hat{Q}(s, a; \hat{w})$ . Apart from that, we implemented an additional mechanism in agent’s behavior named *anti-stuck nudge*. It plays a small but important role both while learning and testing, and makes the agent apply a forward or backward acceleration (random fifty-fifty choice) if within the last 3 seconds the car’s position stayed within a radius of 25 cm — car stucked (see constants in the footnote<sup>10</sup>). This mechanism counteracts a phenomenon of small local undulations (tiny waves) present in the approximation surface generated by the  $\hat{Q}$  model. Such undulations may lead to oscillations in agent’s policy.

**Main Results.** We remark that all experiments can be reproduced by the main script `./src/main.py`, available in the repository. In the script, the simulation and learning-related constants are prefixed by a `QL_` string (e.g.: `QL_RANDOM_SEED = 0`, `QL_DT = 0.25`, `QL_GAMMA = 0.99`, etc.). Based on those constants and the car-related settings in `./src/defs.py`, we have calculated 10-digit hashcodes to identify experiments. In folder `./models_zipped` one can find learning and testing logs, and the models themselves (saved in both binary or `.json` formats).

Table 2 gathers the main results. Columns 5 and 6 describe the quality of learning in terms of the final (at episode no. 10k) frequency of the ‘parked’ event and its exponential moving average (EMA<sup>11</sup>). Obviously, the most important measure is the success frequency at the *test* stage, provided in the right-most column. We have distinguished there experiments where that frequency achieved at least: 50% (green color), 80% (light red) and 95% (dark red).

We have experimented with different settings of penalty coefficients  $(\lambda_d, \lambda_\phi, \lambda_g)$  of the reward function. The initial setting of  $(1, 0, 0)$ , representing rewards based solely on the straight-line distance, led to the worst agents. The subsequent settings of  $(1, 1, 0)$  and  $(1, 1, 1)$  indicated some improvement. From then onwards, we experimented with various proportions by doubling and quadrupling the penalty coefficients ( $\lambda_d=1$  kept constant for reference). We should remark that a single experiment lasted about 5h on our computational environment<sup>12</sup>, hence, an exhaustive search was not possible. The proportions discov-

<sup>10</sup>`QL_ANTISTUCK_NUDGE = True, QL_ANTISTUCK_NUDGE_STEERING_STEPS = 2, defs.CAR_ANTISTUCK_CHECK_RADIUS = 0.25, defs.CAR_ANTISTUCK_CHECK_SECONDS_BACK = 3.0`

<sup>11</sup>informing about the recent tendency in the frequency of successful parking attempts

<sup>12</sup>Hardware: Intel(R) Xeon(R) CPU E3-1505M v5 @ 2.80GHz, 63.9 GB RAM, Quadro M4000M GPU. Software: Windows 10, Python 3.9.7 [MSC v.1916 64 bit (AMD64)], numpy 1.22.3, numba 0.57.0, sklearn 1.0.2.

ered in experiments 25–36, namely:  $(\lambda_d, \lambda_\phi, \lambda_g) = (1, 32, 8)$ , translated onto the best test results observed.

As for state representations, the minimalistic one `avms_fb` (involving the heading angle) turned out to work the worst. Vector-based representations with redundancy appeared to be better suited for neural approximations. It is difficult to point out a clear winner, but based on experiments 25–36, representations `dv_fb`, `dv_ffrblbr2s_da` and `dv_ffrblbr2s_dag` and seem to be the best candidates. Those three representations, combined with the  $(\lambda_d, \lambda_\phi, \lambda_g) = (1, 32, 8)$  coefficients, produced the best performing models, achieving high success rates at the test stage: 99.0%, 98.9% and 99.8%, respectively.

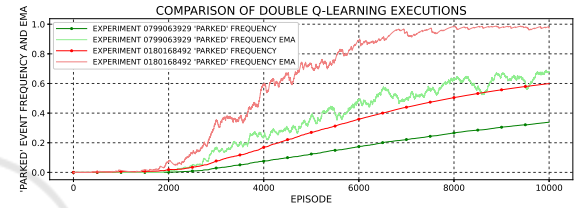


Figure 7: Comparison of two executions of double Q-learning — experiments: 0799063929 vs. 0180168492.

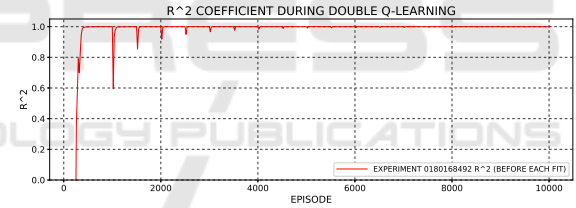


Figure 8: Regression score  $R^2$  (coefficient of determination) for the  $\hat{Q}$  model from experiment 0180168492, plotted along double Q-learning.

Figures 7, 8 provide example plots of: ‘parked’ event frequency, its EMA, and  $R^2$  regression score along the progress of double Q-learning for selected experiments. The characteristic peaks present in Fig. 8 indicate the moments where the target model  $\hat{Q}$  was switched.

Table 3 reports the aforementioned test results on extrapolative generalization. Based on the table, one can see how frequently the best models were able to park successfully starting from a broader range ( $180^\circ$ ) of random angles,  $\psi \sim U(\frac{1}{2}\pi, \frac{3}{2}\pi)$ .

Finally, in Fig. 9 we demonstrate several selected trajectories obtained by the best model (0180168492) during: the learning stage and the two testing stages. We also encourage the reader to see the related sample videos available in the repository by clicking on the presented images (README.md).

Table 2: Results of parking experiments (double Q-learning and testing).

no.	experiment hash code	reward parameters			state representation	final frequency of 'parked' event at learning stage	final EMA of 'parked' event frequency at learning stage	frequency of 'parked' event at test stage
		$\lambda_d$	$\lambda_\phi$	$\lambda_g$				
1	0378634304	1.0	0.0	0.0	avms_fb	2.04%	1.30%	0.8%
2	1228517047	1.0	0.0	0.0	dv_fb	6.03%	14.68%	10.7%
3	3783388807	1.0	0.0	0.0	dv_ffrlblr	5.46%	18.34%	6.6%
4	3223156582	1.0	0.0	0.0	dv_ffrlblr2s	4.63%	15.24%	9.5%
5	1712051871	1.0	1.0	0.0	avms_fb	3.52%	6.00%	7.9%
6	2561934614	1.0	1.0	0.0	dv_fb	7.20%	15.31%	3.4%
7	0821839078	1.0	1.0	0.0	dv_ffrlblr	6.87%	19.87%	14.6%
8	0261606853	1.0	1.0	0.0	dv_ffrlblr2s	7.50%	20.92%	25.9%
9	3085463456	1.0	1.0	1.0	avms_fb	3.49%	10.07%	5.6%
10	3935346199	1.0	1.0	1.0	dv_fb	15.55%	45.24%	34.5%
11	2195250663	1.0	1.0	1.0	dv_ffrlblr	10.11%	24.66%	13.5%
12	1635018438	1.0	1.0	1.0	dv_ffrlblr2s	12.63%	30.23%	33.1%
13	0799063929	1.0	2.0	4.0	dv_fb	33.86%	67.76%	65.1%
14	3353935689	1.0	2.0	4.0	dv_ffrlblr	18.20%	37.67%	61.7%
15	2793703464	1.0	2.0	4.0	dv_ffrlblr2s	21.00%	39.74%	25.3%
16	3390133950	1.0	8.0	16.0	dv_fb	25.44%	46.08%	52.7%
17	0986779886	1.0	8.0	16.0	dv_ffrlblr	19.20%	54.06%	71.1%
18	0799450095	1.0	8.0	16.0	dv_ffrlblr2s	24.39%	46.27%	57.2%
19	0719075893	1.0	4.0	2.0	dv_fb	25.64%	56.98%	43.0%
20	3273947653	1.0	4.0	2.0	dv_ffrlblr	24.21%	52.50%	57.8%
21	2713715428	1.0	4.0	2.0	dv_ffrlblr2s	21.55%	51.01%	46.6%
22	3468419818	1.0	16.0	8.0	dv_fb	52.55%	89.12%	87.3%
23	1065065754	1.0	16.0	8.0	dv_ffrlblr	60.95%	91.31%	93.6%
24	0877735963	1.0	16.0	8.0	dv_ffrlblr2s	43.07%	81.47%	92.6%
25	3497227376	1.0	32.0	8.0	dv_fb	60.64%	98.39%	99.0%
26	1093873312	1.0	32.0	8.0	dv_ffrlblr	55.95%	95.68%	97.8%
27	0906543521	1.0	32.0	8.0	dv_ffrlblr2s	52.75%	96.37%	96.6%
28	2686104021	1.0	32.0	8.0	dv_fb_d	49.05%	93.32%	80.2%
29	3755253765	1.0	32.0	8.0	dv_ffrlblr_d	58.14%	94.19%	88.6%
30	4119351048	1.0	32.0	8.0	dv_ffrlblr2s_d	57.66%	96.22%	94.8%
31	2849328398	1.0	32.0	8.0	dv_fb_da	43.98%	88.94%	94.6%
32	1633232094	1.0	32.0	8.0	dv_ffrlblr_da	56.02%	95.71%	94.2%
33	0053945917	1.0	32.0	8.0	dv_ffrlblr2s_da	61.35%	97.80%	98.9%
34	0937679483	1.0	32.0	8.0	dv_fb_dag	32.05%	72.47%	74.6%
35	1893399723	1.0	32.0	8.0	dv_ffrlblr_dag	59.87%	91.10%	96.7%
36	0180168492	1.0	32.0	8.0	dv_ffrlblr2s_dag	59.92%	97.39%	99.8%

Table 3: Extrapolative generalization: results of best models from Table 2 on a broader range ( $180^\circ$ ) of initial random angles, i.e.  $\psi \sim U(\frac{1}{2}\pi, \frac{3}{2}\pi)$ .

experiment hash code	state representation	frequency of 'parked' event at 2nd test stage
1893399723	dv_ffrlblr_dag	91.8%
1093873312	dv_ffrlblr	94.5%
0053945917	dv_ffrlblr2s_da	87.3%
3497227376	dv_fb	96.8%
0180168492	dv_ffrlblr2s_dag	89.9%

## 8 ADDITIONAL EXPERIMENTS

**Arbitrary Initial Positions and Angles.** Having discovered the best performing combination of: reward coefficients  $(\lambda_d, \lambda_\phi, \lambda_g) = (1, 32, 8)$  and state representation  $dv\_ffrlblr2s\_dag$  from the set of main experiments (previous section), we performed additional experiments with more general scenes.

In the first additional setup the park place was fixed in the middle of the scene, directed to the west:  $\vec{x}_p = (0.0, 0.0)$ ,  $\vec{d}_p = (-1.0, 0.0)$ ; but the car's initial position was entirely random within the region of  $20\text{ m} \times 20\text{ m}$  and with any heading angle within  $360^\circ$ , namely:

$$\vec{x} \sim U([-10.0, 10.0] \times [-10.0, 10.0]),$$

$$\psi \sim U(0, 2\pi).$$

For reference, we preserved 10k learning episodes, but since the scenes became more general we

employed neural networks of larger internal capacity. The sizes of hidden layers in each action-related substructure were doubled to become: (512, 256, 128, 64). This translated onto approximately four times more network weights than before, i.e.  $\approx 1.6\text{ M}$  (because of dense layers). Also, we have enlarged the batch samples in the experience replay to the size  $4 \cdot 65536 = 262144$ . In consequence, the time of one full experiment with 10k episodes also increased about four times, becoming  $\approx 22\text{ h}$ .

The resulting model (0623865367) that we obtained was able to perform complicated and very interesting maneuvers, including e.g.: hairpin turns, rosette-shaped turns, and zigzag patterns. Some of its trajectories are illustrated in Fig. 10, for videos we again address the reader to the repository. The success rate for this model at test stage was 97.8% (over 1k attempts with 25s time limit).

**Model Transfer to Rotation-Invariant State Representation.** Our second additional experiment aimed to check if model 0623865367, described in the previous paragraph, can be *transferred* to a rotation-invariant state representation and perform well *without training* on new testing scenes where not only the car's but also the park place's position and angle are arbitrary (within  $20\text{ m} \times 20\text{ m}$ ). More pre-

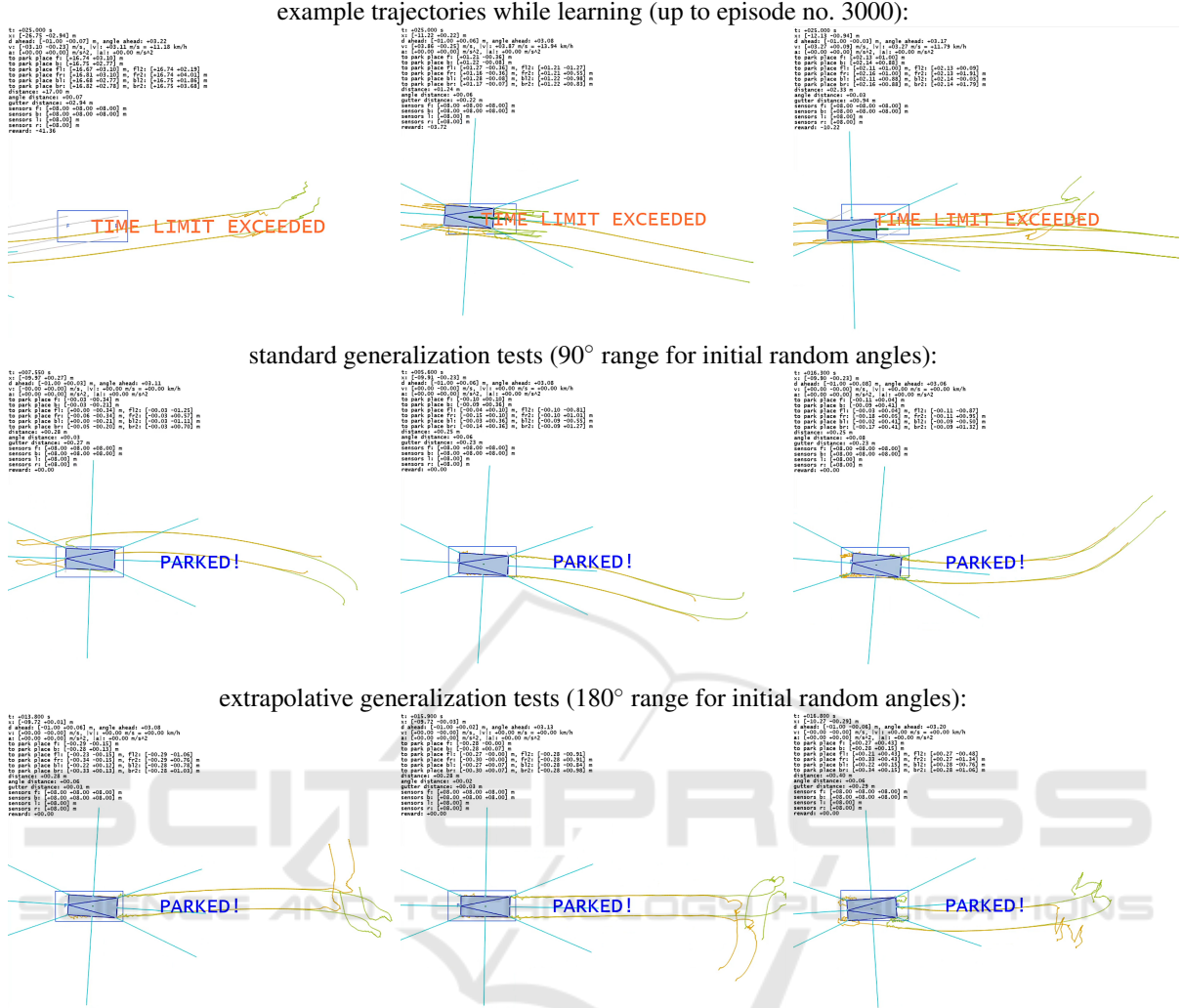


Figure 9: Example trajectories at learning and testing stages (model: 0180168492).

cisely, we used the following distributions:

$$\begin{aligned} \vec{x} &\sim U([-10.0, 10.0] \times [-10.0, 10.0]), \\ \psi &\sim U(0, 2\pi), \\ \vec{x}_p &\sim U([-10.0, 10.0] \times [-10.0, 10.0]), \\ \psi_p &\sim U(0, 2\pi). \end{aligned} \quad (21)$$

The rotation-invariant state representation, called `dv_ffrblbr2s_dag_invariant`, can be introduced by finding the park place rotation angle as

$$\beta = \arctan2(-\vec{d}_p) \quad (21)$$

and then multiplying the vector components of `dv_ffrblbr2s_dag` representation by a suitable rotation matrix at every simulation step, as shown below.

state representation  
`dv_ffrblbr2s_dag_invariant`:

$$\left( M \cdot \vec{d}, M \cdot \vec{v}, M \cdot \vec{f}_{L2}, M \cdot \vec{f}_{R2}, M \cdot \vec{b}_{L2}, M \cdot \vec{b}_{R2}, \|\vec{x} - \vec{x}_p\|, \phi(\vec{d}), g(\vec{x}) \right)_s, \quad (22)$$

where

$$M = \begin{pmatrix} \cos \beta & -\sin \beta \\ \sin \beta & \cos \beta \end{pmatrix} \quad (23)$$

is the rotation matrix.

Tests carried out for the transferred model on 1k scenes (log file 1168277942\_t.log in the repository) revealed the correct and again interesting behavior of the agent. Selected examples of maneuvering trajectories are illustrated in Fig. 12. The success rate we obtained was 94.5% — slightly lower than in the previous case, but we believe still satisfactory taking into

park place fixed in middle,  
arbitrary car position within 20 m×20 m  
and arbitrary angle within 360°:

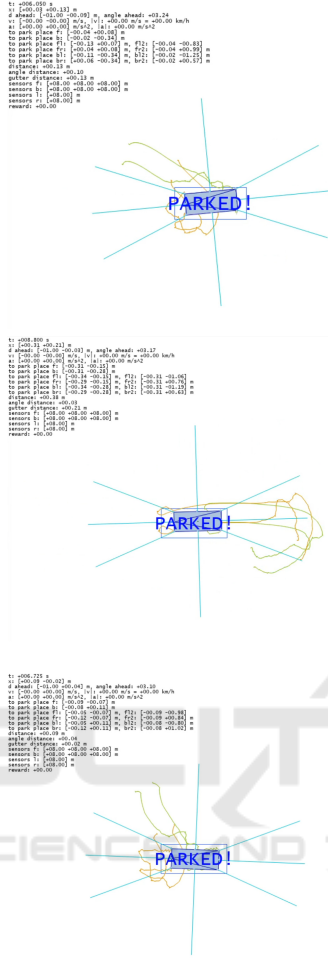


Figure 10: Example trajectories at test stage obtained by the model trained on scenes with arbitrary initial car position within 20 m×20 m and angle within 360° (experiment no. 0623865367).

account the facts that the model was not trained (just transferred) and that it was submitted to new scenes, not experienced before, with arbitrary park place rotations and positions.

### Obstacles and Sensors — Preliminary Attempts.

It is natural to ask whether the proposed reward function (16) could be applicable for scenes with obstacles present. A complete answer to such a question requires a thorough analysis and more experiments, and we plan those to be our future research direction. Nevertheless, in this paragraph we report our very first, preliminary attempts on this problem.

We are going to consider a scene with two obstacles located sideways with respect to the target park place — a common real-life situation where other cars

are parked in the adjacent spots. We are aware that other arrangements of obstacles (e.g. front and rear obstacles) might affect the results, especially having the gutter distance in mind.

In the experiments to follow, the rotation-invariant state representation was extended with 8 or 12 readings from virtual sensors, represented graphically by the blue beams around the car in figures, comprising the total of 23 or 27 features (representation: `dv_ffrlblr2s_dag_invariant_sensors`). Each sensor indicates the distance to the closest obstacle along its direction. The maximum reading of 8.0 m indicates no obstacle seen within that distance.

To have a referential view on the applicability of reward function (16), we have preserved  $(\lambda_d, \lambda_\phi, \lambda_g) = (1, 32, 8)$  — the best coefficients observed in former experiments. Yet, in the presence of obstacles there is now one new element that must be considered, i.e. a reward for *collision* (in fact, a penalty). Let us denote it by  $r_c$  constant. Therefore, the new reward function is of the following form with three cases:

$$r(s') = \begin{cases} 0; & \text{if car successfully parked in } s', \\ r_c; & \text{if car collided in } s', \\ -(\Delta t + \lambda_d \|\vec{x}_{|s'} - \vec{x}_p\| + \lambda_\phi \phi(\vec{d}_{|s'}) / \pi + \lambda_g g(\vec{x}_{|s'})). & \end{cases} \quad (24)$$

It is difficult to say in advance what is a good choice for the value of  $r_c$ . For the purpose of preliminary experiments described here, we picked  $r_c$  to be equal to  $-10^2$ . This choice can be treated as a guess. In fact, it is plausible that  $r_c$  should somehow depend on  $(\lambda_d, \lambda_\phi, \lambda_g)$ . Note also that for any fixed choice of  $r_c$  there shall always exist some states  $s'$ , suitably deviated from the ideal target position, such that

$$-(\Delta t + \lambda_d \|\vec{x}_{|s'} - \vec{x}_p\| + \lambda_\phi \phi(\vec{d}_{|s'}) / \pi + \lambda_g g(\vec{x}_{|s'})) < r_c,$$

which implies collisions not to be the worst possible states. Obviously, in real life one prefers not to park than to collide.

One more mathematical detail that needs to be discussed. It pertains to collisions and the context of episodic tasks. Suppose  $r_1, \dots, r_T$  is a sequence of rewards — realizations of  $R_1, \dots, R_T$  random variables, and  $T$  stands for the last time index. In our case  $T = 25 \text{ s} / \Delta t = 25 \text{ s} / (4\delta t) = 250$ . Suppose also, a collision takes place at time index  $T_c$ . Then the discounted return the agent collects ought to be calculated as:

$$\sum_{t=1}^{T_0-1} \gamma^{-1} r_t + \sum_{t=T_0}^T \gamma^{-1} r_c = \sum_{t=1}^{T_0-1} \gamma^{-1} r_t + r_c \frac{\gamma^{T_0-1} - \gamma^T}{1 - \gamma}. \quad (25)$$

model transfer using rotation-invariant state representation,  
arbitrary position and angle for both car and park place (within  $20\text{m} \times 20\text{m}$ ):

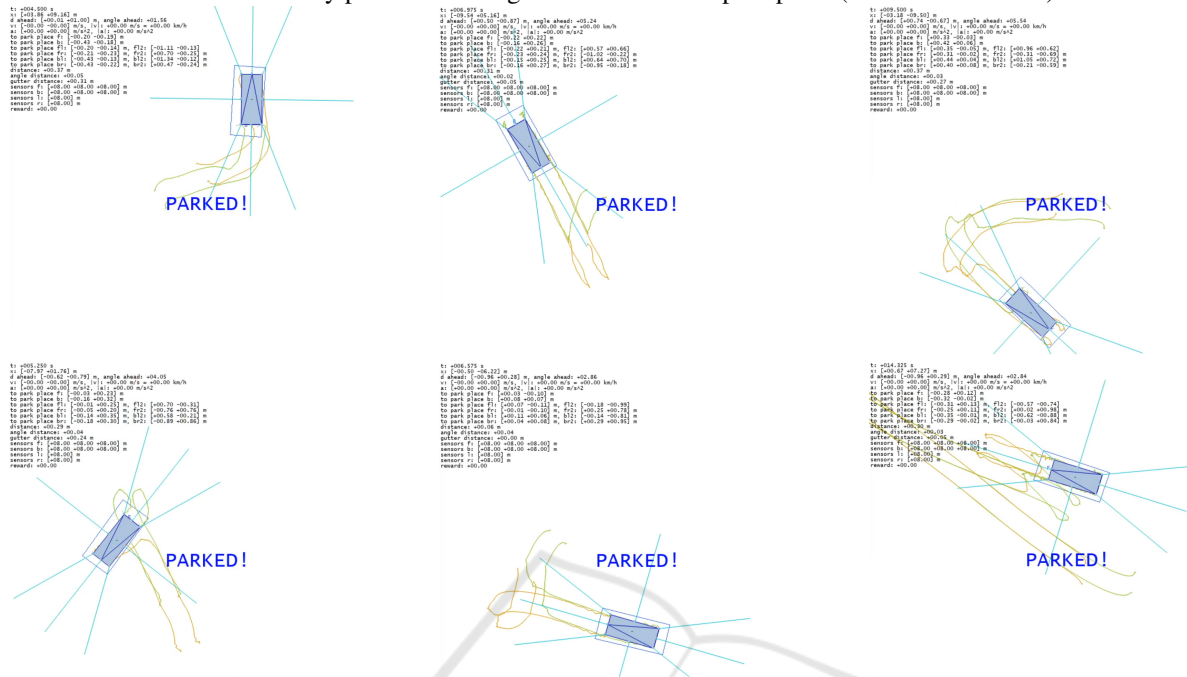


Figure 11: Example trajectories obtained at tests by model 0623865367 transferred to new scenes using rotation-invariant state representation `dv_ffrblbr2s_dag_invariant`.

park place fixed in middle with 2 obstacles 1 m sideways (model 2914586007)

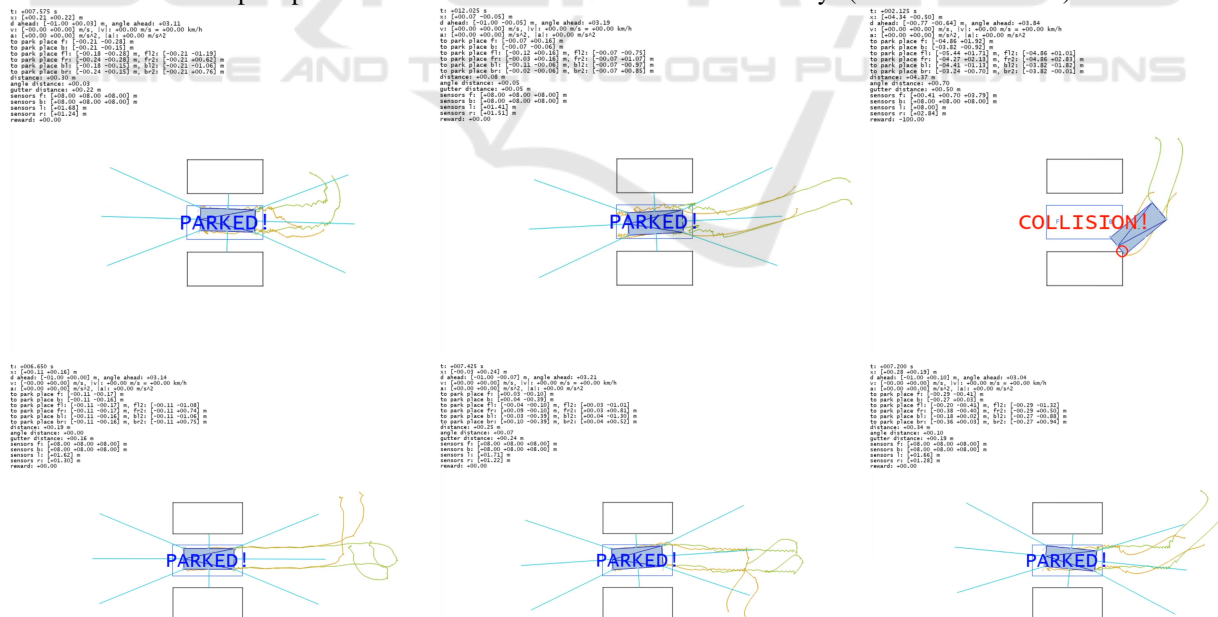


Figure 12: Example trajectories obtained at tests by model 2914586007 for a scene involving 2 obstacles located 1 m to the sides of the park place.

It means that although a simulation as such stops at the collision moment, the agent keeps on receiving constant negative rewards for collision until the end of episode. Note that this is consistent with the zero-valued reward for successful parking which cancels the tail of discounted rewards in a similar manner. In the context of double Q-learning computations and the formula (9) for the target regression values:

$$y_i^* := r_i + \gamma \tilde{Q}(s'_i, \arg \max_{a'} \widehat{Q}(s'_i, a'; \widehat{w}); \widehat{w}),$$

the proper handling of the future returns for *terminal* states can be achieved simply by replacing the  $\tilde{Q}(\cdot)$  response as follows

$$y_i^* := r_i + \gamma \cdot 0, \quad (26)$$

$$y_i^* := r_i + \gamma \cdot r_c, \quad (27)$$

for the successfully parked car and the collided car, respectively.

We now move on to the details of this additional experiment. The park place was located at  $\vec{x}_p = (0.0, 0.0)$  and directed along  $\vec{d}_p = (-1.0, 0.0)$ . Two obstacles were adjacent sideways to it, each 1 m away from the park place border. Random distributions for the initial car position and angle were:  $\vec{x} \sim U([5.0, 15.0] \times [-5.0, 5.0])$ ,  $\psi \sim U(\frac{1}{2}\pi, \frac{3}{2}\pi)$  — i.e. a range of 180°. In experiments involving 8 sensors, their layout was (3, 3, 1): 3 in front, 3 in the back, 1 at each side. In experiments involving 12 sensors, their layout was (3, 3, 3).

Table 4: Results of preliminary experiments on parking with obstacles. Reward function coefficients:  $(\lambda_d, \lambda_\phi, \lambda_g) = (1, 32, 8)$ , state representation: `dv_ffrlbr2s_dag_invariant_sensors`, batch size for experience replay: 262k.

no.	experiment hash code	sensors	final frequency of 'parked' event at learning stage	final EMA of 'parked' event frequency at learning stage	frequency of 'parked' event at test stage
episodes: 10k, NN: 9 × (256, 128, 64, 32)					
1	0809626551	(3, 3, 1)	32.29%	60.54%	42.3%
2	0501599417	(3, 3, 3)	39.60%	68.66%	57.8%
episodes: 20k, NN: 9 × (256, 128, 64, 32)					
3	2914586007	(3, 3, 1)	63.72%	73.59%	80.6%
4	2606558873	(3, 3, 3)	35.33%	40.71%	69.4%
episodes: 10k, NN: 9 × (512, 256, 128, 64)					
5	0726961302	(3, 3, 1)	21.70%	48.25%	56.4%
6	4063022036	(3, 3, 3)	16.18%	35.23%	41.0%
episodes: 20k, NN: 9 × (512, 256, 128, 64)					
7	2831920758	(3, 3, 1)	38.41%	56.85%	62.0%
8	1873014196	(3, 3, 3)	41.19%	59.48%	70.4%

Table 4 summarizes the results obtained in this preliminary experiment (example trajectories shown in Fig. 12). Overall, the results are not satisfactory but also not too pessimistic. Six out of 8 models managed

to perform more than 50% successful parking maneuvers at the test stage in the presence of obstacles. The best observed model (2914586007) achieved the success rate of 80.6%. The troubling aspect is that no clear tendencies can be seen in the results, which makes them difficult to understand. All the tested settings (smaller / larger NNs, fewer / more sensors, fewer / more training episodes) seem not to have a clear impact on final rates. Therefore, as mentioned before, the general problem setting — parking with obstacles — is planned as our future research direction.

## 9 CONCLUSIONS AND FUTURE RESEARCH

Within the framework of reinforcement learning, we have studied a simplified variant of the parking problem (no obstacles present, but time regime imposed). Learning agents were trained to park by means of the double Q-learning algorithm and neural networks serving as function approximators. In this context, our main points of attention pertained to: reward functions (parameterized) and state representations relevant for this problem.

We have demonstrated that suitable proportions of penalty terms in the reward function, coupled with informative state representations, can translate onto accurate neural approximations of long-term action values, and thereby onto an efficient double Q-learning procedure for a car parking agent. Using barely 10k training episodes we managed to obtain high success rates at the testing stage. In the main set of experiments (Section 7) that rate was exceeding 95% for several models, reaching 99% and 99.8% for two cases.

In the additional set of experiments (Section 8) we showed that using larger capacities of neural models the agent was able to learn performing well in more general scenes involving arbitrary initial positions and rotations of both the park place and the car. In particular, the agent learned to perform complicated and interesting maneuvers such as hairpin turns, rosette-shaped turns, or zigzag patterns without supervision — i.e. with being explicitly instructed about trajectories of such maneuvers.

Our future research shall pertain to a general parking problem with obstacles present in the scenes and sensor information included in state representation (with time regime preserved). Preliminary results for such a problem setting (Section 8) indicate the need for more experiments and analysis. Also, it seems appropriate to conduct in our future work a compar-

ative evaluation of parking agents using such frameworks as, for example, Farama<sup>13</sup> or a more general MuJoCo<sup>14</sup>.

## REFERENCES

- Aditya, M. et al. (2023). Automated Valet Parking using Double Deep Q Learning. In *ICAECIS 2023*, pages 259–264.
- Cai, L. et al. (2022). Multi-maneuver vertical parking path planning and control in a narrow space. *Robotics and Autonomous Systems*, 149:103964.
- Chai, R. et al. (2022). Deep Learning-Based Trajectory Planning and Control for Autonomous Ground Vehicle Parking Maneuver. *IEEE Transactions on Automation and Science Engineering*, 20:1633–1647.
- Chen, X. et al. (2021). Independent Q-learning for Robust Decision-Making under Uncertainty. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11598–11606.
- Fujita, Y. et al. (2021). ChainerRL: A Deep Reinforcement Learning Library. *Journal of Machine Learning Research*, 22(77):1–14.
- Goyal, A. et al. (2019). Modular deep reinforcement learning with functionally homogeneous modules. In *Advances in Neural Information Processing Systems*, volume 32, pages 12449–12458.
- Hasselt, H. (2010). Double Q-learning. In *Advances in Neural Information Processing Systems*, volume 23, Curran Associates, Inc.
- Hasselt, H., van Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. In *Proc. of 30th AAAI Conf. on AI*, pages 2094–2100.
- Kingma, D. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. In *ICLR (Poster)*. <https://arxiv.org/pdf/1412.6980.pdf>.
- Kobayashi, T. and Ilboudo, W. (2021). t-soft update of target network for deep reinforcement learning. *Neural Networks*, 136:63–71.
- Li, Y. et al. (2018). Neural Network Approximation Based Near-Optimal Motion Planning with Kinodynamic Constraints Using RRT. *IEEE Transactions on Industrial Electronics*, 65:8718–8729.
- Mankowitz, D. et al. (2018). Multi-head reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31, pages 960–971.
- Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2013). Playing Atari with Deep Reinforcement Learning. arXiv 1312.5602 cs.LG.
- Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–533.
- Nair, V. and Hinton, G. (2010). Rectified linear units improve restricted Boltzmann machines. In *ICML 2010*, pages 807–814.
- Pedregosa, F. et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.
- Sall, R., Wagner, R., and Feng, J. (2019). Drivers’ Perceptions of Events: Implications for Theory and Practice. In *Proceedings of the Human Factors and Ergonomics Society 2019 Annual Meeting*, pages 1996–2000. Sage Journals.
- Shinners, P. (2011). PyGame. <http://pygame.org/>.
- Sowerby, H., Zhou, Z., and Littman, M. (2022). Designing Rewards for Fast Learning. arXiv 2205.15400 cs.LG.
- Sprenger, F. (2022). Microdecisions and autonomy in self-driving cars: virtual probabilities. *AI & SOCIETY*, 37:1–16.
- Sutton, R. and Barto, A. (2018). *Reinforcement Learning: An Introduction*. Bradford Books, 2 edition.
- Thrun, S. and Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School*.
- Zeng, D. et al. (2019). A unified optimal planner for autonomous parking vehicle. *Control Theory and Technology*, 17:346–356.
- Zhang, P. et al. (2019). Reinforcement Learning-Based End-to-End Parking for Automatic Parking System. *Sensors*, 19:3996.

<sup>13</sup><https://github.com/Farama-Foundation/HighwayEnv>

<sup>14</sup><https://mujoco.org>