Automatic parking with Q-Learning

Leonardo Lai[1]

***Abstract*— Automatic parking is one of the fundamental challenges on the path towards self-driving vehicles. In this work I show how a reinforcement learning method, namely Q-learning, can be used to deal with the problem of automatic parking. The algorithm is then implemented in a simulated environment to prove its effectiveness.**

I. INTRODUCTION

Parking is one of the most common tasks when driving a car, yet it is quite difficult for many people, especially new drivers; mistaken estimation of the space between the car and the surroundings is often a cause of accidents. For this reason, many automobile manufacturers equip their vehicles with parking assistance systems (e.g. distance sensors), to make the procedure easier; nonetheless, these solutions still require a strong degree of human interaction. In the last years, more and more producers are showing a growing interest in self-driving cars, seeing them as a viable possibility for the future of transportation. In this perspective, they are investing a lot of resources to research and develop autonomous systems, including fully automated parking solutions. The advantages are several: improved safety, faster parking and, ultimately, less burden for the driver.

Solutions based on machine learning look very promising in this direction, thanks to their ability to learn from the environment and work in various scenarios. Moreover, the recent developments in hardware technologies enable ML algorithms to be deployed in embedded environments, including vehicles. Within the ecosystem of ML approaches, reinforcement learning is particularly suitable to include a model of the safe and harmful behaviours of an agent, in this case a car (e.g. mapping collisions to negative rewards). Researchers have started adopting this paradigm, obtaining interesting results. Shalev-Shwartz et al. applied reinforcement learning to the problem of forming long-term driving strategies, accounting for safety and potential unpredictable agents [3]. Sallab et al. proposed a deep RL framework including RNN for car lane keeping that works even in partially observable scenarios [4].
A survey on this topic has been published by Kiran et al [5].

II. PROBLEM

A. Model

A generic parking scenario is modeled by considering a bounded *region* containing elements of three types: a vehicle, a parking slot (i.e. the target final configuration for the car) and some obstacles (i.e. objects that reduce the set of legal configurations). Both the car and the obstacles can be represented by arbitrarily complex polygons. A model example of a typical parking situation is depicted in Fig. 1. A *collision* event is geometrically determined by a partial overlapping of the car polygon with any obstacles, as a result of a maneuver. Going out the region boundaries can be treated equal to a collision, as long as the allowed region is reasonably large for parking. The model considers only fixed obstacles (like other parked cars, or structures), not moving ones (like people); anyway, a simple and practical solution to deal with them would be to halt and wait until they eventually go away.

In order to fit the model in a reinforcement learning framework, it is necessary to map the model to the concepts of *state*, *actions* and *rewards*. In this case, the current state is fully determined by the position and orientation of the car. To have a finite number of states, the space is discretized with a tunable granularity: e.g. $25\ cm$ for the coordinates and $4\ deg$ for the orientation angle. The actions are represented by a finite set of maneuvers, which are a combination of forward/backward displacement and left/right turning. As shown in Fig. 2, I consider elementary shifts of $\pm 50/100/150\ cm$ and turns with a radius of $5/10\ m$. This representation is clearly not exhaustive, but any sequence of complex maneuvers can be approximated quite well by a composition of these actions. Note that shifts of 100 and 150 cm are actually redundant, but shorten the length of these sequences, making the algorithm learn faster. The rewarding scheme is very simple: a large reward $(+1000)$ is given when an action leads to the final target position, a large punishment (-200) is issued on collisions and a small penalty (-5) for each maneuvering step. The latter prevents stalls and loops, encouraging the car to reach the end as quickly as possible.

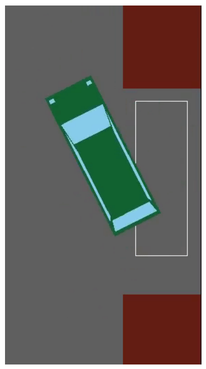

Fig. 1: Environment model: car, parking slot, obstacles

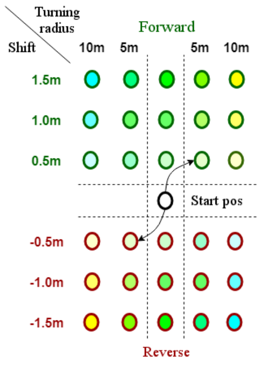

Fig. 2: Action space: possible vehicle maneuvers

[1]Leonardo Lai, Sant'Anna School of Advanced Studies, Pisa, Italy leonardo.lai@santannapisa.it

B. *Q-learning*

Q-learning is an off-policy reinforcement learning algorithm to determine the optimal action-selection policy for any state [1]. A $Q(s,a)$ matrix stores a quality score associated to every state-action pair. Q is not known a priori, but it is built iteratively, attempting to maximize the expected cumulative reward over the next steps from the current state s; Q is updated after every interaction with the environment, combining the current knowledge about the quality of the successive state s', and the reward r associated with the selected action. The update rule is formulated as follows:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a')\right]$$

where α and γ are two hyperparameters called *learning rate* and *discount factor*, respectively. At the start, Q is initialized with all values equal to 0. The training phase consists in simulating different episodes many times, updating the Q matrix according to the aforementioned rule. At each step, the next action is selected with a ϵ-greedy policy.
The set of possible states and actions for the car agent corresponds to that described in the previous section.
Since the actions are discrete, a sufficiently long and varied training ensures the convergence of Q-learning [1].
In practice, I defined a custom loss-like metric δ_k to monitor the convergence of the model at iteration k: starting from an initial value Δ, δ_k is computed from δ_{k-1} (previous value) and the variation of $Q(s,a)$ if it was updated at time k:

$$\delta_k = \begin{cases} \Delta & \text{for } k=0 \\ \delta_{k-1} & \text{for } k>0 \wedge s' \text{ illegal} \\ (1-\zeta)\delta_{k-1} + \zeta|Q_k(s,a) - Q_{k-1}(s,a)| & \text{for } k>0 \wedge s' \text{ legal} \end{cases}$$

As shown in Fig. 3, this value rapidly decays during the first episodes, until the agent eventually reaches the final state for the first time; here δ_k bounces up, as an effect of the Q matrix updates, but resumes decreasing afterwards until convergence. The training phase stops when δ_k settles to a sufficiently stable value, or alternatively after a maximum number of episodes.

III. Implementation and Experimental Results

The proposed approach has been validated in simulation[1], in a scenario with 30 actions and 54000 states (positions and orientations) like the one in Fig. 1. Although the Q-learning theorem ensures convergence after every state/action has been considered *many* (i.e. infinitely many) times, a practical rule of thumb is to visit every state-action pair about 10 times; clearly, this value may vary with problem difficulty. From empirical observation, an episode typically explores a ten of different states before terminating in a successful parking or a crash. Hence, according to the rule above, one may expect the training to be approximately as long as $E_{ref} = 1.6M$ episodes to achieve reasonable results.

The *training factor* τ is defined normalizing the actual training length w.r.t. the such reference number: $\tau = \frac{E_{train}}{E_{ref}}$.

[1]The simulator was entirely developed in C++, without ML frameworks. Code publicly available at: `github.com/leoll2/Autoparking`

Experiments have been performed to measure the network effectiveness, that is the fraction of successful parking attempts, as a function of the training length, along with the average number $\bar{m}$ of maneuvers (actions) required to reach the end. The results are shown in Table I: it turns out that the network starts behaving occasionally well at $\tau = 0.3$, and rapidly improves its success rate as it keeps learning until $\tau = 1$; afterwards it slowly continues making progress, eventually peaking at $\tau = 10$ when it becomes fully reliable, that is crashes are no longer observed. Note the trend of $\bar{m}$: it is initially low because the network can only resolve simple cases, where the car is already near the target position, then increases as it deals with more complex cases, and finally returns decreasing thanks to path optimization.

Other experiments were executed to determine the optimal hyperparameters α, γ and ϵ, varying them while keeping constant $\tau = 1$ (Tab. IIa, IIb, IIc). Since the environment is deterministic, the optimal learning rate α is 1 as expected. Different discount factors γ, instead, do not appear to cause significant performance variations. As for the ϵ-greedy ϵ, any value is acceptable except for very low ones (no exploration).

IV. Conclusions and Future Works

This articles showed the feasibility and effectiveness of Q-learning to tackle the problem of automatic parking. The next step could be to exploit deep Q-learning and other generalization-oriented techniques to deal with maps (obstacle layouts) not previously practiced in simulation [6][7].

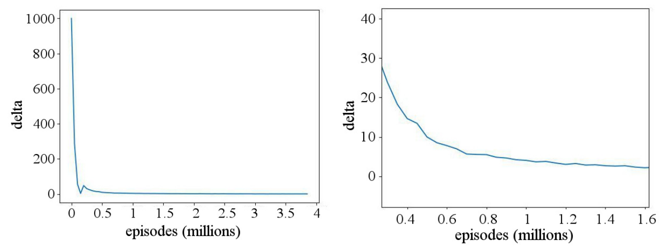

Fig. 3: Monitoring Q-matrix convergence over time with δ_k

τ	episodes	$\bar{m}$	success (%)
0.3	486 000	-	0
0.4	648 000	5.25	12.2
0.5	810 000	8.30	33.0
0.7	1 134 000	12.43	81.6
1	1 620 000	10.81	98.6
3	4 860 000	9.85	99.9
10	16 200 000	9.54	100

TABLE I: Relation between train episodes and success rate

α	$\bar{m}$
0.0	-
0.3	10.48
0.7	9.84
1.0	9.56

(a) Learning rate

γ	$\bar{m}$
0.3	9.61
0.6	9.82
0.9	9.67
1.0	9.70

(b) Discount factor

ϵ	$\bar{m}$
0.0	18.58
0.1	9.82
0.3	9.60
0.5	9.70
0.7	9.61
1.0	9.69

(c) ϵ-greedy epsilon

References

[1] Watkins, C. J. & Dayan, P. (1992). *Q-learning*. Machine learning, 8(3-4), 279-292.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

[3] Shalev-Shwartz, S., Shammah, S., Shashua, A. (2016). Safe, multi-agent, reinforcement learning for autonomous driving.

[4] Sallab, A. E., Abdou, M., Perot, E., Yogamani, S. (2017). Deep reinforcement learning framework for autonomous driving. Electronic Imaging, 2017(19), 70-76.

[5] Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., Pérez, P. (2020). Deep Reinforcement Learning for Autonomous Driving: A Survey

[6] Cobbe, K., Klimov, O., Hesse, C., Kim, T., Schulman, J. (2018). Quantifying generalization in reinforcement learning.

[7] Packer, C., Gao, K., Kos, J., Krähenbühl, P., Koltun, V., Song, D. (2018). Assessing generalization in deep reinforcement learning.