Simulated Autonomous Vehicle Parking: Examining Model-Based and Model-Free Reinforcement Learning Methods

Jean-Rafael Ereyi
Computer Science
jrereyi@stanford.edu

Andrew Hojel
Computer Science
ahojel@stanford.edu

Oscar O'Rahilly
Computer Science
oscarfco@stanford.edu

Christina Knight
Symbolic Systems
cqknight@stanford.edu

Abstract

We present various approaches for autonomous vehicle parking using model-based and model-free reinforcement learning methods. The vehicle autonomously explores a parking lot through turning maneuvers. The car observes an x-coordinate, y-coordinate, x-velocity, y-velocity, cos (heading angle) and sin (heading angle) at every step. Our reward dynamics model then guides learning by calculating weighted L1 p-norm between the state and the goal. We use a random baseline and Lookahead with Rollout, Forward Search, Monte Carlo Tree Search, and Soft Actor-Critic (SAC) as our other methods. We then used four metrics to compare these methods: the maximum reward received, the sum of the reward received, the agent's success ratio, and the mean number of steps taken to park in successful scenarios. Altogether, this paper successfully applies reinforcement learning techniques to train a car in a simulated environment.

1. Introduction

Autonomous driving has the potential to transform the travel industry. An important component of autonomous driving is parking. Parking is a challenging task that requires a driver to maneuver the car into a spot both accurately and efficiently. Many automobile manufacturers have already installed this technology for an automatic parking option. However, this technology still needs to be improved, especially in a large space with other parked cars. This paper trains a simulated car to park in a parking space. The simulated environment is a continuous space with a target parking spot. We also experimented with an environment that included stationary obstacles (other parked cars). To model the state dynamics of the environment, we developed a deep neural network to approximate a continuous linear time invariant system. This paper compares various model-based and model-free reinforcement learning methods to solve this problem.

2. Related Work

Previous research papers on autonomous parking and driving have explored classical reinforcement learning techniques and reinforcement learning in conjunction with other methods. For instance, Zhang et al. used model-based reinforcement learning to park by iteratively executing data generation, data evaluation, and training networks. [9] This paper combines Monte Carlo Tree Search (MCTS) with longitudinal and lateral policies and then learns the parking strategy using a neural network. Other papers have attempted model-free reinforcement learning methods, such as deep Q-Learning and several actor-critic methods. Haarnoja et al. experimented with a Soft Actor-Critic (SAC) method, an off-policy actor-critic algorithm based on the maximum entropy reinforcement learning framework in 2019. [4] In the autonomous parking setting, Folkers et al. base their training on the proximal policy optimization (PPO) algorithm, which considers an infinite MDP. Others, such as Maravall et al., uses artificial neural networks (ANN) to autonomously park. [7] [2] Previous papers also create state dynamics models from a Linear Time Invariant (LTI) systems for this type of problem. For instance, Schucker et al. [8] and Keen et al. [5] used LTI systems to model vehicle trajectory and driver steering skills, respectively. Altogether, an extensive literature exists on autonomous driving for modelling state dynamics, and for learning from model-free and model-based methods.

3. Environment

3.1. State Space

For this problem, parking cars takes place in an environment containing the target car to park, and a goal parking spot. In this environment, state measurements are given by observations, where a given observation reports the x and y location of the car, along with the car's x and y velocity, and the sin and cos of the heading angle: `[x,y,vx,vy,cos_h, sin_h]` The target car and the goal report their own observations. At each time-step t, the current observation of the target car is reported as well as an observation of the goal. For a given environment, the x and y locations are relative to the starting location of the car, as the starting location of the car is always at $x = 0$, $y = 0$. The starting position of the car (and also the origin) is at the center of the parking lot. An image of a car in its starting position and the corresponding goal is shown in the figure below.

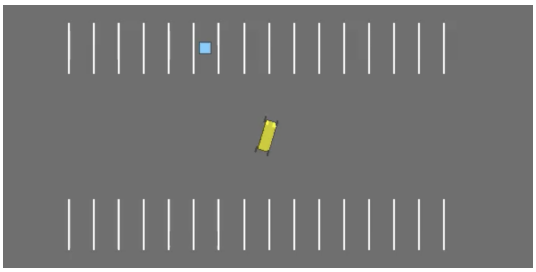


Figure 1: Starting position of environment, where yellow car is at the origin and the blue square is the goal state.

The duration of the environment is modeled as 100 time steps (t_{end}), such that the terminal state is when either the goal parking location has been reached, the car hits any of the obstacle cars, or 100 time steps have passed.

3.2. Action Space

To model the driving capabilities of the target car, the car varies both its steering angle and acceleration. The valid range of steering angles is between $-45°$ to $45°$, and valid acceleration values are between -5 and 5.

3.3. Reward Model

The reward model used was a weighted p-norm consisting of the L1 norm between the car's current state and the goal state (ie. the parking spot). Since the parking spot is always stationary, we ignore the difference between the speed of the goal and the car's current state. Therefore, it is a weighted p-norm consisting of the difference in position (x, y) values and heading of the car (the restriction on the heading puts pressure for the car to properly pull into the space and not dive in a straight line to the goal's location). The value of p selected was 0.5 and the reward weights focused on the x and y position primarily with lighter weight on the steering angle. The reward function can be written as follows where w is the weight vector and s is the state vector:

$$\textit{reward} = (w^T|s_{\text{curr}} - s_{\text{goal}}|)^p \tag{1}$$

4. Agent Methods

4.1. Model-Based Reinforcement Learning

In model based methods, explicit representations of the transition and reward models are built in order to inform decisions about the optimal actions to choose.

4.1.1 Discretizing the Action Space

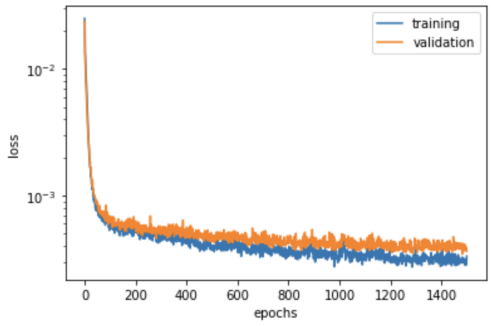
For model-based methods, like Lookahead with Rollouts, Forward Search, and Monte Carlo Tree Search, discrete actions are required. Every action needs to be considered to get all possible next states for the transition model from our current state to any depth d. As the action space is continuous, with given valid ranges for acceleration and steering angles, it needs to be discretized to be able to run any of the previously mentioned models. Discretization of our action space was performed two separate ways. Discretization was done by choosing a number of actions to be generated, n, and then randomly generating n valid accelerations and steering angles to result in n discrete actions. Due to computational limitations running various models, n was treated as a hyperparameter to be tuned by each method.

4.1.2 State Dynamics Model

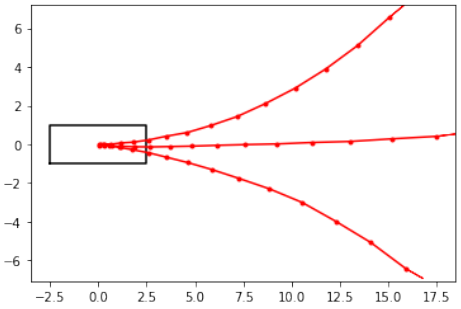
When working with model-based reinforcement learning, it was necessary to create a state dynamics model (ie. a transition model). This model is used to predict the next state as a result of taking a certain action. To approximate the dynamics of the `highway-env` environment, we used a Linear Time Invariant (LTI) system. Many works utilize LTI systems as a basis for modeling vehicle trajectory such as Schucker et al. [8], and our inspiration for a applying a neural approximation was from Avalos et al. [1]. We focused on the continuous time-invariant state space model, where the derivative of the state vector $\frac{d}{dt}s(t)$ can be calculated by multiplying the dynamics matrix A and input matrix B with the current state $s(t)$ and current action $a(t)$ respectively. The equation is shown below:

$$\frac{d}{dt}s(t) = As(t) + Ba(t) \tag{2}$$

To model this change in state space, we used a Deep Neural Network (DNN) to approximate the dynamics and



(a) State Dynamics Model Train/Val Loss



(b) State Dynamics Model Predicted Trajectories

Figure 2: Training data and visualized results of the State Dynamics Model

input matrix, as follows:

$$\frac{d}{dt}s(t) \approx f_\theta(s,a) = A_\theta s + B_\theta a \tag{3}$$

To train this neural network we samples 7,500 state, action, new state triples randomly from the environment. Each of the learned matrices was approximated by a 4-layer neural network that projected the state-action vector (d=8) to a 256-dimensional hidden layer, followed by a projection to a 128-dimensional hidden layer, finishing with a projection to the 6x6 matrix (the dynamics matrix, A_θ) or to the 6x2 matrix (the input matrix, B_θ). The model was trained using Mean Squared Error loss, and to prevent overfitting, we added a dropout layer with `p=0.2` before the projection from the 128-dimension hidden layer to the parametrized matrices.

The trained model was then used to determine the next state by multiplying the neurally approximated derivative of the state vector at the current state by the change in time (policy frequency) of the environment and adding the current state, as follows:

$$s_{\text{next}} = s + \Delta t \cdot f_\theta(s,a) \tag{4}$$

4.1.3 Lookahead With Rollouts (LwR)

Lookahead with Rollouts is the most rudimentary model-based approach we implemented. This method estimates the value of each action through a simulation up to depth d. We randomly sample 100 actions. Then, the simulation uses a stochastic roll-out policy from the generative model above. We found that a depth of 5 and a discount factor of 0.9 yielded optimal results. Increasing the depth further did not seem to improve the action taken.

4.1.4 Forward Search

Forward search, a model-based method, predicts the best action to take at a certain state by exploring every action in the search tree up until a prescribed depth, d. When the algorithm reaches the specified depth, it calculates the expected utility by using the Bellman equation (current reward + a discounted previous reward). We implemented forward search in a recursive manner and returned the action that yielded the maximum utility.

4.1.5 Monte Carlo Tree Search

Monte Carlo Tree Search predicts the optimal action by simulating a specified number of actions from the current state. This method has a lower complexity than the two above methods because it only runs a certain number of simulations. This algorithm then updates the action-value function and the number of times a state-action pair has been explored. We run the simulation 100 times and then choose the action that maximizes our action-value function. The exploration strategy we used for the MCTS was UCB1 Exploration Heuristic with the exploration parameter set to $c = 1$, which is shown below (where $Q(s,a)$ is the action-value function and $N(s,a)$ is the number of times a particular state-action pair has been selected):

$$Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \tag{5}$$

4.2. Model-Free Reinforcement Learning

Model-Free Reinforcement Learning does not require us to build explicit representations of transitions or reward models.

4.3. Soft Actor-Critic

Soft Actor-Critic (SAC) is a model-free deep reinforcement learning algorithm. The objective of SAC is:

$$J(\pi) = \mathbb{E}_\pi\Big[\sum_t r((s_t,a_t) - \alpha\log(\pi(a_t|s_t))\Big] \tag{6}$$

In this equation above, s_t and a_t are the respective state and action the agent can take at time step t. The optimal policy

maximizes the expected return and expected entropy. The SAC algorithm maximizes this objective by parameterizing a Gaussian policy and a Q-function with a neural network. For our hyperparameters, we used a $\gamma = 0.9$, a batch size of 256, $5e^4$ training steps, a learning rate of $1e^{-3}$, and a buffer size of $1e^6$. It is important to note that SAC is very insensitive to mild changes in hyperparamters which meant we did not experiment too much with these values.

5. Experiments and Results

5.1. Reward Shaping

Given that our reward function is a p-norm of the weighted sum of the difference between each feature `(x,y,vx,vy,cos_h, sin_h)` at the current state and the goal state, there was significant opportunity for experimenting with different reward function. We experimented with many variations of weights, and make the following discoveries:

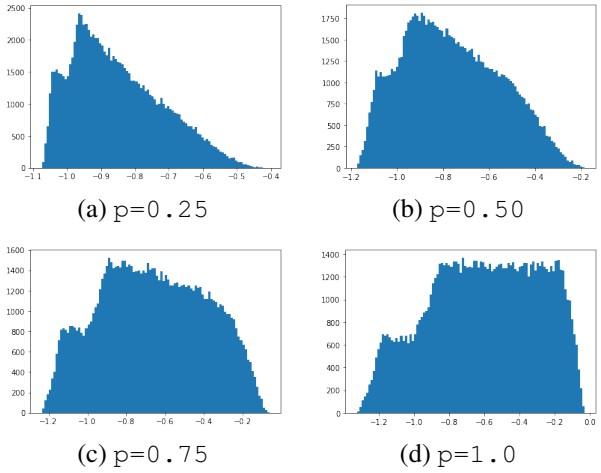
- Weighting x more than y was necessary because the vehicle had to travel a greater distance in the x direction than the y direction.
- Weighting the velocity caused the vehicle to stay stationary, which is intuitive given that the vehicle was rewarded for maintaining as low a velocity as possible.
- Weighting the heading angle (`cos_h, sin_h`) helped ensure that the vehicle pulled into the lane properly and did not drive to the goal parking spot in a straight line.

Our final weighting (corresponding the feature order listed above) was `[1., 0.3, 0., 0., 0.02, 0.02]`.

In addition to determining the individual feature weights, we wanted to investigated the effect of varying the p value in the weighted p-norm used to calculate the reward. To visualize the effect of different p values we ran 100,000 bootstrapping simulations randomizing realistic values for the goal state vector and current state vector and then calculating the reward given these values. We found that as the p value was decreased, the kurtosis of the reward function increased (ie. the reward values were more highly concentrated in a smaller range). This can be seen in the histograms produced by the bootstrapping in Figure 3. As a result of this analysis, we decided to choose $p = 0.5$ to choose slightly higher kurtosis while remaining well distributed in *reward* $\in [0, 1]$.

5.2. Multiple Parked Cars

As methods such as Forward Search, Lookahead with Rollout and Soft Actor-Critic were able to perform so well on an open parking lot, we wanted to increase the difficulty of our environment. We extended our environment parking lot to include multiple parked cars (an example image is shown below). In order for our car to learn not to collide with the park cars, we also introduced a collision penalty of -5.



(a) `p=0.25` (b) `p=0.50` (c) `p=0.75` (d) `p=1.0`

Figure 3: Results of bootstrapping 100,000 reward values varying p value

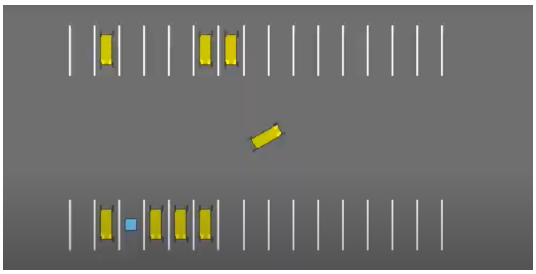


Figure 4: Example environment with parked car obstacles

Due to the increased complexity of our environment, models like forward search and Soft Actor-Critic took far too long to run a simulation and so we tested this new environment on our lightest weight well-performing algorithm: Lookahead with Rollout. For this implementation of LwR, we initially sampled 100 actions and ran it to a depth of 3.

The results are shown in the Metrics section below.

5.3. Metrics

To evaluate the quality of our different methods we decided to use four metrics: the max reward received, the sum of the reward received, the success ratio of our agent, and the mean number of steps it took to park in successful scenarios.

We run 100 simulations for each model and report the mean of all the statistics discussed above. For each method we compare it against our baseline (Random), and our best performing model (Soft Actor-Critic). The full metrics table

can be found in the Appendix. The results can be seen in the section below.

5.4. Results

5.4.1 Lookahead with Rollouts

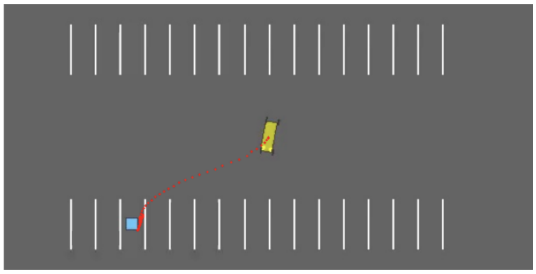


Figure 5: Visualization of car parking with LwR. Dots indicate subsequent time steps

Lookahead with Rollouts (1) was run with 100 sampled actions, $\gamma = 0.9$, and with a depth of 3. As it executed much faster than our other models, in order to compare its effectiveness to the other models, Lookahead with Rollouts (2) was run again, this time with 25 sampled actions, $\gamma = 0.9$, and with a depth of 2.

Algorithm	Max Reward	Sum Reward	Success Ratio	# Steps
LwR (1)	-0.116	-16.9	0.85	42.1
Random	-0.396	-53.086	0.01	45
SAC	**-0.113**	**-8.959**	**0.96**	**26.2**
LwR (2)	-0.139	-24.633	0.58	37.7

Table 1: Metrics for Lookahead with Rollouts

5.4.2 Forward Search

In the model simulations, Forward search was run with 25 sampled actions, $\gamma = 0.9$, and with a depth of 2. In order to compare the effectiveness of Lookahead with Rollouts to Forward Search, Lookahead with Rollouts was run with the same hyperparameters.

Algorithm	Max Reward	Sum Reward	Success Ratio	# Steps
Random	-0.396	-53.086	0.01	45
SAC	**-0.113**	**-8.959**	**0.96**	**26.2**
Fwd Search	-0.129	-22.948	0.75	47.5
LwR	-0.139	-24.633	0.58	37.7

Table 2: Metrics for Forward Search

5.4.3 Monte Carlo Tree Search

Due to limitations in computational power, attempting to run Monte Carlo Tree Search with 25 actions and a depth of 3 was infeasible. Monte Carlo Tree Search was run with 10 sampled actions, $\gamma = 0.9$, and a depth of 3.

Algorithm	Max Reward	Sum Reward	Success Ratio	# Steps
Random	-0.396	-53.086	0.01	45
SAC	**-0.113**	**-8.959**	**0.96**	**26.2**
MCTS	-0.174	-29.332	0.44	57.5

Table 3: Metrics for Monte Carlo Tree Search

5.4.4 Soft Actor-Critic

In the model simulations, Soft Actor-Critic was run with $\gamma = 0.9$, a batch size of 256, 5000 training steps, a learning rate of 0.001, and a buffer size of 1,000,000.

Algorithm	Max Reward	Sum Reward	Success Ratio	# Steps
Random	-0.396	-53.086	0.01	45
SAC	**-0.113**	**-8.959**	**0.96**	**26.2**

Table 4: Metrics for Soft Actor-Critic

5.4.5 Parked Cars

As Lookahead with Rollouts was our most computationally efficient model to perform well on our original environment, it was run on an environment with parked cars placed as obstacles. It was run with 100 sampled actions, $\gamma = 0.9$, and with a depth of 3. The random baseline was also ran on the parked car environment for comparison.

Algorithm	Max Reward	Sum Reward	Success Ratio	# Steps
LwR	**-0.249**	**-15.258**	**0.97**	**26.9**
Random	-0.454	-50.859	0.0	0

Table 5: Metrics for Lookahead with Rollouts on Parked Car Environment

5.5. Video Results

To see the full video simulations of some of the results discussed above please follow the link below. In this video we demonstrate results from four of our experiments:

1. Random (baseline) method

2. Lookahead with Rollout
3. Soft Actor-Critic
4. Lookahead with Rollout on the environment with Parked Cars

Link: https://www.youtube.com/watch?v=4vKWjGuvFYs

6. Discussion

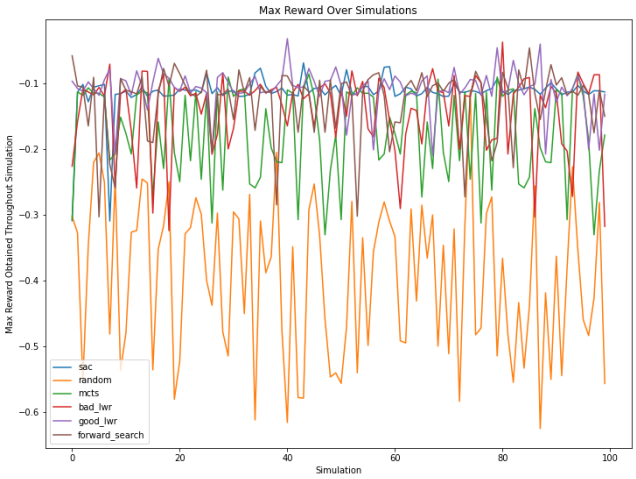


Figure 6: Graph Showing Max Reward Obtained During Each Simulation for Each Method Discussed Above

Out of all four methods discussed, the Soft Actor-Critic method performed the best, with a max reward of -0.113, a sum reward of -8.959, a success ratio of 0.96 (96%), and 26.2 average steps taken to park (out of successful rounds). This method can approximate the reward of each action from a state better than the other algorithms. The lookahead with rollout algorithm implemented with 100 actions (LwR(1)) sampled yielded the second best results. This method obtained a maximum reward of -0.139, a summed reward of -16.9, a success ratio of 0.85 (85%), and 42.1 average steps taken to the parking spot. However, when lookahead with rollout was performed with 50 actions (LwR(2)) (the same amount as forward search), it performed worse than forward search. Looking at Figure 6, we can see visually see the difference between Lwr(1) (good_lwr) and LwR(2) (bad_lwr). If we had the computational time to compute forward search with 100 actions, we predict that it would have performed better than lookahead with rollout (LwR (1)). It is also interesting to note the large difference in variance amongst LwR(1) and LwR(2). If we look at Figure 6, we see that LwR(2) has a far greater range in the maximum reward it received during its 100 simulations - which of course makes sense considering the shallower exploration depth and much smaller sampled action space. Somewhat surprisingly, Monte Carlo Tree Search performed badly in the task of parking cars. This could be attributed to the small action space (10) that our computational power allowed us to use.

In running Lookahead with Rollouts on the parked car environment and contrasting results with the obstacle-free environment, the model actually performed better on the parked car environment. This could be due to the added constraints the presence of obstacles provided to the environment. This penalty could have served to additionally constrain our action space, ensuring the actions returned by Lookahead with Rollouts were more optimal.

7. Conclusion

This paper explored various reinforcement learning methods to park a car in a continuous state space. Our environment consists of one target parking spot. We also experimented with building another environment with parked cars. However, with our computational power, we could only perform Lookahead with Rollout on this new environment. Our best performing method, as predicted, was the Soft Actor-Critic (SAC) method, which yielded a 96% accuracy rate. This makes a lot of sense for a few reasons. Firstly, we spent the most amount of time training SAC, allowing it ample time converge to an optimal policy function. Secondly, the SAC policy is trained to maximize a trade-off between expected return and randomness, meaning it can account well for stochasticity in environments. As we are randomly assigning the parking spot each simulation, there is a large amount of stochasticity within our environment which the SAC is able to perform very well in. For future works, we would want to try scenarios where we receive more noisy estimates for sensor data from the car. We would also like to try more model-free methods, such as Deep Q-learning and Deep Deterministic Policy Gradient.

8. Contributions

Work done amongst the team members happened almost entirely in pairs of two or as a group of four. In other words, we always worked either all together or in pairs, so there was no clear work division. In terms of each pair, the work breakdown is as follows:

- Oscar & JR
  - Lookahead With Rollouts, Monte Carlo Tree Search, Soft Actor-Critic, Experiments & Results Aggregation
- Andrew & Christina

– Forward Search, State Dynamic Model, Reward Shaping & Adding Parked Vehicles to Environment

9. Acknowledgements

We would like to acknowledge the fantastic parking environment developed by Edouard Leurent (DeepMind), which we used as a foundation to expand off [6]. In addition, we referenced Pradeep Gopal's implementation of Soft Actor-Critic when developing our own implementation [3]. Finally, we got inspiration for a neural approximation of a continuous linear time invariant state space model from the RLSS 2019 Github [1].

We would like to note that Lookahead With Rollouts, Forward Search, Monte Carlo Tree Search, the State Dynamics Model, the addition of parked vehicles in the environment, and all data visualizations were developed completely from scratch.

10. Appendices

Algorithm	Max Reward	Sum Reward	Success Ratio	# Steps
LwR (1)	-0.116	-16.9	0.85	42.1
Random	-0.396	-53.086	0.01	45
SAC	**-0.113**	**-8.959**	**0.96**	**26.2**
Fwd Search	-0.129	-22.948	0.75	47.5
MCTS	-0.174	-29.332	0.44	57.5
LwR (2)	-0.139	-24.633	0.58	37.7

Table 6: Metrics for Each Approach (No Parked Vehicles)

References

[1] R. Avalos, G. Cideron, O. Domingues, and Y. Flet-Berliac. Rlss 2019: Pratical sessions. https://github.com/yfletberliac/rlss-2019, 2019. 2, 7

[2] A. Folkers, M. Rick, and C. Büskens. Controlling an autonomous vehicle with deep reinforcement learning. *CoRR*, abs/1909.12153, 2019. 1

[3] P. Gopal, A. Virmani, D. Kothandaraman, and S. M. Makan. Implementation of decision making algorithms in openai parking environment, Nov 2020. 7

[4] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018. 1

[5] S. D. Keen and D. J. Cole. Application of time-variant predictive control to modelling driver steering skill. *Vehicle System Dynamics*, 49(4):527–559, 2011. 1

[6] E. Leurent. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env, 2018. 7

[7] D. Maravall, M. A. Patricio, and J. Asiaín. Automatic car parking: A reinforcement learning approach. volume 2686, pages 214–221, 06 2003. 1

[8] J. Schucker and U. Konigorski. Linear time-variant vehicle trajectory guidance with nonlinear inversion-based feedforward. *IFAC-PapersOnLine*, 52(5):378–384, 2019. 9th IFAC Symposium on Advances in Automotive Control AAC 2019. 1, 2

[9] P. Zhang, L. Xiong, Z. Yu, P. Fang, S. Yan, J. Yao, and Y. Zhou. Reinforcement learning-based end-to-end parking for automatic parking system. *Sensors*, 19(18), 2019. 1