Fast Adaptation of Deep Reinforcement Learning-Based Navigation Skills to Human Preference

Jinyoung Choi¹, Christopher Dance², Jung-eun Kim¹, Kyung-sik Park¹, Jaehun Han¹, Joonho Seo¹, Minsu Kim¹

Abstract—Deep reinforcement learning (RL) is being actively studied for robot navigation due to its promise of superior performance and robustness. However, most existing deep RL navigation agents are trained using fixed parameters, such as maximum velocities and weightings of reward components. Since the optimal choice of parameters depends on the use-case, it can be difficult to deploy such existing methods in a variety of real-world service scenarios. In this paper, we propose a novel deep RL navigation method that can adapt its policy to a wide range of parameters and reward functions without expensive retraining. Additionally, we explore a Bayesian deep learning method to optimize these parameters that requires only a small amount of preference data. We empirically show that our method can learn diverse navigation skills and quickly adapt its policy to a given performance metric or to human preference. We also demonstrate our method in real-world scenarios.

I. INTRODUCTION

Recently, deep reinforcement learning (RL) approaches to autonomous navigation have been actively studied. Such studies have reported higher performance compared with traditional methods [1] and better robustness to changes in the environment [2]. However, most existing deep RL methods are trained using fixed values for parameters such as the robot’s maximum velocity and the weightings expressing the trade-off between reward components (for instance, following a short path to the goal versus keeping a large safety distance). These parameters are often hand-engineered [3] or optimized based on simple criteria such as the number of waypoints reached [1].

This can be problematic in many real-world scenarios since the desirable robot behavior varies with the use-case. For example, robots deployed in hospital wards must be cautious to avoid collisions with delicate equipment and in order to not scare patients, whereas the top priority for a warehouse robot can be to arrive at a goal as soon as possible. Existing methods, which are trained with fixed parameters, cannot meet such diverse requirements and would need retraining to fine-tune them to each scenario. Furthermore, the desirable behavior of robots that interact with humans often depends on human preferences, which can be expensive to collect. Therefore, one needs not only agents that can adapt to diverse parameters but also a way to accurately predict near-optimal parameters quickly from small amounts of human preference data.

This paper presents a novel solution to these problems with the following main contributions:

• We propose a deep RL method that learns navigation policies that adapt to a wide range of reward weightings and other navigation parameters;
• We propose a method for navigation-parameter optimization, based on Bayesian deep learning, that requires only small amounts of preference data; and
• We provide empirical evidence that these methods together enable quick adaptation of navigation skills resulting in superior user satisfaction.

The next sections discuss related work and introduce our methods. Experimental results and conclusions follow.

II. RELATED WORK

A. Autonomous Navigation Using Deep RL

Deep RL-based robot navigation is being actively studied as it promises greater performance and robustness than traditional approaches. [1] reported that deep RL-based navigation can outperform traditional planning methods such as artificial potential fields [4] and the dynamic window approach [5]. [6] and [7] proposed deep RL-based multi-robot collision avoidance and showed superior performance compared to the widely-used ORCA algorithm [8]. [2] used a deep RL policy as a local planner within the probabilistic road map [9] approach to solving long-range navigation problems, finding that the RL-based local planner could
successfully generalize to environments different from its training environment.


Although these works show promising results, they train with fixed navigation parameters such as maximum velocity and reward weightings. The resulting agents are thus only capable of navigating with fixed behavioral characteristics such as cautiousness or speed. Since fine-tuning the policies of deep RL agents often takes millions of training steps, whereas traditional navigation algorithms can be tuned without expensive training, this lack of flexibility limits the real-world deployment of deep RL approaches.

In contrast, we propose a method that adapts to a wide range of navigation parameters, overcoming this inflexibility. Our adaptable policies differ from the policies resulting from universal value functions [15] which only generalize over goal states. Our policies also differ from those in so-called reward-variable settings [16], [17] which are limited to parameters that linearly weight reward components.

B. Hyperparameter Optimization in Deep RL

Navigation parameters such as maximum accelerations and reward weightings have a significant effect on robot behavior. Most previous work treats these parameters as hyperparameters, which are hand-tuned [3], [6]. However, such hand-tuning often requires a lot of trial and error.

To ease this problem of hyperparameter search, AutoRL [1] uses an evolutionary algorithm to tune both the reward weightings and the neural network architecture so as to maximize simple criteria such as the probability of reaching the goal and the number of waypoints passed. Although AutoRL greatly improves performance according to such criteria, it runs large numbers of experiments using massive amounts of computing power. Furthermore, as we show in our experiments section, optimal parameters can vary with the environment and the performance metric.

To solve this problem, we propose to first train an adaptable agent and then use Bayesian deep learning to quickly optimize the parameters at test time.

C. Reinforcement Learning from Human Preferences

In complex tasks, manually-designed reward functions often fail due to reward hacking [18], poorly specified reward values and other difficulties [19]. Therefore, many works use human preference to guide reward-function design. [20] proposed to learn a reward function by regression with a Bradley-Terry model [21] for preferences over pairs of trajectories. They successfully trained agents without explicitly specifying reward functions in Atari games. However, they required thousands of human preference queries to ensure an adequate coverage of the state space when fitting the reward function, which would be prohibitive in a robotics setting.

[22] combined expert demonstrations with active generation of preference queries to reduce the required number of queries. They successfully trained a robot arm to move to a goal position while avoiding an obstacle with only 15 preference queries.

In this work, we reduce the number of preference queries required by optimizing only a small number of navigation parameters, and by actively generating the queries by combining upper-confidence bounds (UCB) [23], [24] with Bayesian deep learning [25].

III. APPROACH

In this section, we introduce our problem formulation. Then we discuss our novel methods to train an agent that can adapt to diverse navigation parameters and to optimize these parameters quickly using preference data.

A. Problem Setting

We consider a path-following navigation task, which is similar to the one used in [1]. In this task, the agent follows a path to a goal. The path is represented as sequence of waypoints. When the agent reaches the last waypoint (goal), a new goal and waypoints are given. We model the task as a partially observed Markov decision process (S,A,Ω,r, ptrans, pobs), with sets of states S, actions A and observations Ω, with reward function r, and with conditional state-transition and observation probabilities ptrans and pobs.

We use a differential two-wheeled mobile platform model for the robot dynamics and work in a discounted episodic setting with discount factor γ = 0.99. The process is considered to be parameterised by a navigation parameter drawn from a set of navigation parameters W, as we now describe.

1) Navigation Parameters: Many parameters affect the behavior of RL navigation agents. Here, we focus on seven such parameters and consider a navigation parameter w ∈ W ⊆ R7 to have the following components:

\[ w = (w_{\text{stop}}, w_{\text{socialLimit}}, w_{\text{social}}, w_{\text{maxV}}, w_{\text{accV}}, w_{\text{maxW}}, w_{\text{accW}}), \]

where wstop is the reward in the event of a collision or emergency stop, wsociallimit is the minimum acceptable estimated time to collision with other agents, wsocial is the reward for violating this minimum time, and wmaxV, waccV, wmaxW, waccW are the maximum linear speed, linear acceleration, angular speed and angular acceleration respectively. Our goal is to train agents that can adapt to diverse w and to efficiently find a w that is suitable for a given use-case.

2) Observations: The agent’s observations are of the form o = (o_{scan}, o_{velocity}, o_{odometry}, o_{path}) ∈ Ω ⊆ R^{26}. o_{scan} ∈ R^{18} consists of scan data from a range sensor such as a lidar. We bin the data from −180° to 180° in 20° intervals and take the minimal value from each bin. The maximum distance that the agent can perceive is 3 m. o_{velocity} ∈ R^2 consists of the current linear and angular velocity. The change in the robot’s position relative to the position at the previous timestep is

\[ o_{odometry} = (\Delta x/\Delta t, \Delta y/\Delta t, \cos(\Delta \theta/\Delta t), \sin(\Delta \theta/\Delta t)), \]
where $\Delta x$, $\Delta y$ and $\Delta \theta$ are changes in the robot’s $xy$-position and heading relative to the previous timestep and $\Delta t$ is the duration of one timestep. Lastly, $o_{\text{path}} = (\cos(\phi), \sin(\phi))$ where $\phi$ is the relative angle to the next waypoint in the robot’s coordinate system.

3) Actions: The action of the agent is a vector in $[-1, 1]^2$ representing the desired linear velocity of the robot normalised to the interval $[-0.2 \text{ m/s}, w_{\text{maxV}}]$ and its angular velocity normalised to $[-w_{\text{maxW}}, w_{\text{maxW}}]$. When the robot executes this action, an angular acceleration of $\pm w_{\text{accW}}$ is applied, and the linear acceleration is $w_{\text{accV}}$ when increasing the speed and $-2 \text{ m/s}^2$ when decreasing it.

4) Reward: The reward function $r:S \times A \times W \to \mathbb{R}$ is the sum of five components:

$$r = r_{\text{base}} + 0.1 r_{\text{waypointDist}} + r_{\text{waypoint}} + r_{\text{stop}} + r_{\text{social}}$$

in which every term has the arguments $(s, a, w)$.

Reward $r_{\text{base}} = -0.01$ is given in every timestep to encourage the agent to reach the waypoint in minimum time.

We set $r_{\text{waypointDist}} = -\text{sign}(\Delta d) \sqrt{|\Delta d|/\Delta t/w_{\text{maxV}}}$, where $\Delta d = d_{t} - d_{t-1}$ and $d_t$ is the Euclidean distance to the waypoint from timestep $t$ and $\Delta t$ is the duration of a timestep. We use a square root to reduce the penalty for small deviations from the shortest path that are necessary for collision avoidance. When the distance between the agent and the current waypoint is less than 1 m, there is a reward of $r_{\text{waypoint}} = 1$ and the waypoint is updated.

To force the robot to keep a minimum safety distance, in both simulation and the real world, we stop the robot by setting the linear velocity to 0 m/s if the estimated time to collide with an obstacle or other agent is less than 1 s or if a collision occurs, and a reward of $r_{\text{stop}} = w_{\text{stop}}$ is given. The estimated collision time is calculated using the desired velocity given by the current action, using obstacle points represented by $o_{\text{scan}}$, and modelling the robot as a square of sides 0.5 m.

Reward $r_{\text{social}} = w_{\text{social}}$ is given when the estimated collision time to any other agent is less than $w_{\text{socialLimit}}$ seconds. Estimated collision time is calculated as for $r_{\text{stop}}$, except that we use the position of other agents within a 3 m range instead of scan data. As we do not incorporate the other agents’ positions in the observations, the robot has to distinguish other agents from static obstacles using the sequence of scan data.

B. Learning Adaptable Navigation Policies

We propose a novel training method that learns a navigation policy that can adapt to wide range of navigation parameters without expensive re-training.

1) Multi-Agent Training with Randomized Parameters:

As summarized in Algorithm 1, we use decentralized multi-agent training, similarly to [5], [6]. In each episode, multiple agents are deployed in a shared environment. To make the policy adaptable to diverse navigation parameters, we sample the navigation parameter of each agent from a distribution at the start of each episode. For the RL algorithm, we chose soft actor-critic (SAC) [26] since several benchmarks [27].
Algorithm 2: Bayesian Optimization of the Navigation Parameter Using Preference Data

Given the trained adaptable agent $\pi_{\text{adaptable}}$;
Initialize the Bayesian neural network parameters $\theta_{\text{BN}}$;
Initialize empty datasets $D_{\text{params}}, D_{\text{preference}}$;
$W_{\text{new}} \leftarrow$ sample $N_{\text{query}}$ navigation parameters;
for iteration $i = 1, 2, \ldots, N_{\text{iter}}$ do
  $D_{\text{params}} \leftarrow D_{\text{params}} \cup W_{\text{new}}$;
  Calculate $\mu(f(w|\theta_{\text{BN}})), \sigma(f(w|\theta_{\text{BN}}))$,
  UCB$(w|\theta_{\text{BN}})$ for all $w \in D_{\text{params}}$;
  for query $j = 1, 2, \ldots, N_{\text{query}}$ do
    Sample $w_j^1$ from the $W_{\text{mean}} \subseteq D_{\text{params}}$ having
    $N_{\text{top}}$ highest $\mu(f(w|\theta_{\text{BN}}))$;
    Sample $w_j^2$ from the $W_{\text{UCB}} \subseteq D_{\text{params}}$ having
    $N_{\text{top}}$ highest UCB$(w|\theta_{\text{BN}})$;
    Collect preference $w_j^1 \succ w_j^2$ using
    trajectories from $\pi_{\text{adaptable}}$;
    $D_{\text{preference}} \leftarrow D_{\text{preference}} \cup \{w_j^1 \succ w_j^2\}$;
  end
  Train $\theta_{\text{BN}}$ using $D_{\text{preference}}$ for $N_{\text{update}}$ steps;
  $W_{\text{random}} \leftarrow$ sample $N_{\text{sample}}$ navigation parameters;
  Calculate UCB$(w|\theta_{\text{BN}})$ for all $w \in W_{\text{random}}$;
  $W_{\text{new}} \leftarrow$ set of $w \in W_{\text{random}}$ having $N_{\text{query}}$ highest
  UCB$(w|\theta_{\text{BN}})$;
end
return $w \in D_{\text{params}}$ having the highest $\mu(f(w|\theta_{\text{BN}}))$;

show it is more sample-efficient, more stable and produces better-performing policies than alternatives like PPO [28] and TD3 [29].

2) Neural Network Architecture: As shown in Figure 2 (left), we give the navigation parameter of the agent as additional input to the network. To model the temporal dynamics of an agent and its environment, we use gated recurrent units (GRU) [30], which provide competitive performance to the LSTM used in [3] while requiring less computation.

C. Optimizing Navigation Parameters for Human Preference

Even if we have an agent that can adapt to a wide range of navigation parameters, the problem of finding near-optimal navigation parameters for a given use-case remains. Therefore, we propose a novel Bayesian approach to optimizing navigation parameters using preference data. In this paper we assess preference by pairwise comparisons, as such comparisons are simple to elicit and they overcome many limitations of ratings [31].

1) Preference Model: We use the Bradley-Terry model [21] to model preferences, as in [18], [20]. The probability that navigation parameter $w_1 \in W$ is preferred to $w_2 \in W$ is modeled as

$$P(w_1 \succ w_2) = P(t_1 \succ t_2) = 1/(1 + \exp(f(w_2) - f(w_1))),$$

where $t_1$ and $t_2$ are robot trajectories collected using $w_1$ and $w_2$, $w_1 \succ w_2$ indicates that $w_1$ is preferred than $w_2$, and $f : W \rightarrow \mathbb{R}$ is called the utility function. To facilitate accurate preference evaluation, we collect trajectories $t_1$ and $t_2$ using the same environment and waypoints. We fit utility function $f(w)$ to preference data and use it to predict preferences for new navigation parameters.

2) Active Learning of the Preference Model: We learn utility function $f(w|\theta_{\text{BN}})$ with a Bayesian neural network [25] with parameters $\theta_{\text{BN}}$. We use its estimates of the uncertainty of its predictions to actively generate queries, thus minimising the number of such queries. As shown in Algorithm 2 we train the network (Figure 2 right) to minimize

$$\text{loss}(\theta_{\text{BN}}) = \log(1 + \exp(f(w_{\text{lose}}|\theta_{\text{BN}}) - f(w_{\text{win}}|\theta_{\text{BN}}))),$$

which is the negative log-likelihood of the preference model. In each iteration, we train the network for $N_{\text{update}}$ steps, starting with the parameters $\theta_{\text{BN}}$ from the previous iteration.

We use a variant of upper-confidence bounds (UCB) [23], [24], [32] to actively sample new queries, setting

$$\text{UCB}(w|\theta_{\text{BN}}) = \mu(f(w|\theta_{\text{BN}})) + \sigma(f(w|\theta_{\text{BN}})),$$

where $\mu(f(w|\theta_{\text{BN}}))$ and $\sigma(f(w|\theta_{\text{BN}}))$ are the mean and standard deviation of $f(w|\theta_{\text{BN}})$ computed with $N_{\text{forward}}$ forward passes of the network. We omit the coefficient of $\sqrt{\log(t)}$ that usually appears before the $\sigma(f(w|\theta_{\text{BN}}))$ as it had no clear benefit in our simulated experiments for the small number of queries used.

We generate trajectories using the $N_{\text{query}}$ navigation parameters that have the highest UCB$(w|\theta_{\text{BN}})$ among the $N_{\text{sample}}$ uniformly-sampled navigation parameters. Then, we actively generate $N_{\text{query}}$ new preference queries. To do so, we calculate $\mu(f(w|\theta_{\text{BN}}))$ and UCB$(w|\theta_{\text{BN}})$ for all $w \in D_{\text{params}}$ where $D_{\text{params}}$ is the set of all navigation parameter we have collected trajectory from so far. Let $W_{\text{mean}}$ be the set of samples with the $N_{\text{top}}$ highest $\mu(f(w|\theta_{\text{BN}}))$ in $D_{\text{params}}$ and let $W_{\text{UCB}}$ be the set of samples with the $N_{\text{top}}$ highest UCB$(w|\theta_{\text{BN}})$ in $D_{\text{params}}$. Each preference query consists of a pair of navigation parameters $(w_1, w_2)$ where $w_1$ and $w_2$ are sampled uniformly from $W_{\text{mean}}$ and $W_{\text{UCB}}$.

IV. EXPERIMENTS

In this section, we describe our training environment. Then we evaluate the performance of the adaptable policy and preference-based navigation-parameter optimization algorithm using a synthetic preference model. Lastly, we report the results of navigation-parameter optimization using human
preferences and demonstrate the resulting agent in a real-world environment.

A. Training Environment

Our simulation environment (Figure 3) is based on Unity ML-Agents \cite{33}. We procedurally generate mazes and sample random starting points and goals for the navigation task. Dijkstra’s algorithm \cite{24} was used to generate waypoints. To reduce the gap between simulation and the real world, we apply simple sensor and actuator noises at each timestep. Specifically, we add $\mathcal{N}(0, 0.05)$ to $o_{\text{scan}}$, and multiply $o_{\text{velocity}}$ and $o_{\text{odometry}}$ by $\mathcal{N}(1, 0.05)$. We also add $\mathcal{N}(0, 0.1)$ to the angle $\phi$ when we calculate $o_{\text{path}}$. The duration of a timestep in the simulator is $\Delta t = 0.15$ s/step.

We train the adaptable agent using Algorithm 1. For the navigation-parameter distribution $p_W$, we used the uniform distribution with lower and upper bounds given in Table I. We train $N_{\text{agent}} = 10$ agents for 300,000 weight updates. Each episode ends at timestep 1000.

B. Performance of Adaptable Agent

Firstly we demonstrate that our adaptable agent performs competitively with agents trained in exactly the same way, but with the fixed navigation parameters given in Table I. We test the agents in the three environments shown in Figure 4 (top). We deployed five agents having the same navigation parameter and used same initial positions and waypoints in all experiments. The length of each episode was 500 timesteps. For each baseline, we measure the sum of all agents’ rewards over these 500 timesteps. We also measure the performance of the adaptable agent with the same navigation parameter. As shown in Table III, the adaptable agent performed competitively, or even outperformed the baselines with the same amount of training.

C. Navigation-Parameter Optimization with Oracle Preference

Before testing our navigation-parameter optimization method on human preference data, we demonstrate its performance when preferences are given by simple known metrics. As in \cite{18, 20}, we use an oracle preference, which always prefers the trajectory having higher total reward. We define the following three total reward metrics to measure the performance of trajectories of length $T$ collected from environments having $N_{\text{agents}}$ agents:

$$R_{\text{waypointDist}} = \sum_{t=0}^{T} \sum_{i=0}^{N_{\text{agents}}} r_{\text{waypointDist}}^{i}$$

$$R_{\text{social}} = \sum_{t=0}^{T} \sum_{i=0}^{N_{\text{agents}}} r_{\text{social}}^{i}$$

$$R_{\text{mixed}} = R_{\text{waypointDist}} + 10 R_{\text{social}}$$

which are calculated with $w_{\text{maxV}} = 1$, $w_{\text{socialLimit}} = 4$, $w_{\text{social}} = -1$, $T = 500$ and $N_{\text{agents}} = 5$. For each metric, we optimize the navigation parameter of trained adaptable agent using Algorithm 2 with the oracle preference. Since this is a stochastic algorithm, we ran three trials of navigation-parameter optimization to verify its consistency.

For the hyperparameters of Algorithm 2 we found that $N_{\text{query}} = N_{\text{top}} = 10$, $N_{\text{update}} = 10000$, $N_{\text{sample}} = 3500$ and $N_{\text{forward}} = 50$ were enough to provide consistent results across multiple trials of experiments. We used the uniform distribution with bounds given by Table I to sample $W_{\text{rand}}$ and initialize $W_{\text{new}}$. We use these values for later experiments as well, assuming that performance metrics used in this experiment are representative of our a priori beliefs as to what the preference function might look like.

As shown in Figure 5, we compared the performance metrics of adaptable agent with optimized navigation parameter to the baselines of Table I. The adaptable agent achieved higher or competitive performance with a small number of preference queries. This confirms that our approach can adapt quickly to diverse environments and metrics using only preference data, resulting in competitive performance.
D. Optimizing Navigation Parameters for Human Preference

We conducted a navigation-parameter optimization experiment with preference data from five human participants, taking a cafe as environment and drink-delivery as the task. There were three male and two female participants. Three of the participants were robotics researchers and two were user-experience designers. Before collecting any preference data, we asked participants to “prefer” agents that they judge to be better suited to that environment and task. Participants then discussed the desired properties of such a robot. They agreed that the robot should move at a similar speed to a walking human, should not change the direction too frequently, and should keep enough distance from humans.

We did 10 iterations of navigation-parameter optimization. In each iteration, each participant evaluated two pairs of trajectories, giving a total of 10 preference queries. Trajectories were collected from the simulated cafe environment (Figure 4 bottom). 10 agents with the same navigation parameter were deployed in the environment and the length of trajectory was 500. Since participants found it difficult to perceive the speed of the robot and its proximity to obstacles from the simulated trajectories, we additionally provided numerical values for $\omega_{\text{velocity}}$, $r_{\text{stop}}$, and the shortest estimated collision time to any other agent $r_{\text{social}}$. We also provided cumulative values of $r_{\text{waypointDist}}$, $r_{\text{stop}}$, and $r_{\text{socialTimeScore}} = -\max\{0, 1 - r_{\text{social}}/4\}$ up to the current timestep. $w_{\text{maxV}} = 1$, $w_{\text{stop}} = -1$ were used to compute $r_{\text{waypointDist}}$ and $r_{\text{stop}}$.

After 10 iterations, we selected the navigation parameter $w_{\text{best}}$ with the highest mean utility $\mu(f(w|\theta_{\text{BN}}))$ from the collected dataset. The numerical values of $w_{\text{best}}$ were $w_{\text{stop}} = -0.91$, $w_{\text{socialLimit}} = 3.53$ s, $w_{\text{social}} = -0.99$, $w_{\text{maxV}} = 1.04$ m/s, $w_{\text{accV}} = 0.45$ m/s$^2$, $w_{\text{maxW}} = 0.38$ rad/s and $w_{\text{accW}} = 0.97\pi$ rad/s$^2$. Then we did 30 additional preference evaluations (six per participant), pairing trajectories from the $w_{\text{best}}$ with uniformly-sampled navigation parameters. We resampled trajectories from $w_{\text{best}}$ for every evaluation to prevent participants from memorizing specific numbers or behaviors. In the additional evaluations, $w_{\text{best}}$ was preferred in 27 out of these 30 evaluations, showing that our method successfully optimizes the policy to the participants’ preferences.

E. Real-World Experiment

To test the optimized navigation parameter $w_{\text{best}}$ in the real-world, we demonstrated the optimized policy in a real cafe environment (Figure 1), whose layout is the same as in the preference experiment just described. We built a mobile robot platform with time-of-flight range sensors and an NVIDIA Jetson AGX Xavier as its main processor. We collected 10 trajectories using $w_{\text{best}}$ and random navigation parameters respectively. Then, we randomly matched the videos in pairs. Working with the same five participants as above, each participant evaluated six pairs. The additional information given in the simulation was not given in real-world preference evaluations. The navigation parameter $w_{\text{best}}$ was preferred in 28 out of the 30 evaluations. We refer the reader to the supplementary material for the videos used in this experiment.

V. Conclusion

In this paper, we propose a novel approach to training deep RL-based navigation policies that can adapt to a wide range of navigation parameters without expensive retraining, and a Bayesian deep learning method for tuning navigation parameters using a small number of preference evaluations over pairs of trajectories. We demonstrated that our method can optimize navigation parameters to maximize human preference using only small amounts of preference data. Our optimized agent was significantly more likely to be preferred by users in both simulated and real-world experiments.

Acknowledgement

We thank Tomi Silander for his help in writing the paper and Naver Labs’ PDX team for participating in experiments.
REFERENCES


[27] 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,” 