

Risk-Conditioned Distributional Soft Actor-Critic for Risk-Sensitive Navigation

Jinyoung Choi¹, Christopher Dance², Jung-eun Kim¹, Seulbin Hwang¹, Kyung-sik Park¹

Abstract—Modern navigation algorithms based on deep reinforcement learning (RL) show promising efficiency and robustness. However, most deep RL algorithms operate in a risk-neutral manner, making no special attempt to shield users from relatively rare but serious outcomes, even if such shielding might cause little loss of performance. Furthermore, such algorithms typically make no provisions to ensure safety in the presence of inaccuracies in the models on which they were trained, beyond adding a cost-of-collision and some domain randomization while training, in spite of the formidable complexity of the environments in which they operate. In this paper, we present a novel distributional RL algorithm that not only learns an uncertainty-aware policy, but can also change its risk measure without expensive fine-tuning or retraining. Our method shows superior performance and safety over baselines in partially-observed navigation tasks. We also demonstrate that agents trained using our method can adapt their policies to a wide range of risk measures at run-time.

I. INTRODUCTION

Deep reinforcement learning (RL) is attracting considerable interest in the field of mobile-robot navigation, due to its promise of superior performance and robustness compared with classical planning-based algorithms [1], [2]. Despite this interest, few existing works on deep-RL-based navigation attempt to design risk-averse policies. This is surprising for several reasons. First, a navigating robot might cause harm to humans, to other robots, to itself or to its surroundings, and risk-averse policies may be safer than risk-neutral policies, while avoiding the over-conservative behaviour typical of policies based on worst-case analyses [3]. Second, in environments with such complex structure and dynamics that it is impractical to provide accurate models, policies optimizing certain risk measures are an appropriate choice, as they actually provide guarantees on robustness to modelling errors [4]. Third, the end-users, insurers and designers of navigation agents are risk-averse humans [5], so risk-averse policies seem to be the natural choice.

To address the issue of risk in RL, recent works [6], [7] have introduced the concept of *distributional RL*. Distributional RL learns the distribution of accumulated rewards,

¹NAVER LABS, Gyeonggi-do, 13494, South Korea

²NAVER LABS Europe, 6 chemin de Maupertuis, Meylan, 38240, France. Website: europe.naverlabs.com

¹jy-choi@naverlabs.com

²chris.dance@naverlabs.com

© 2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

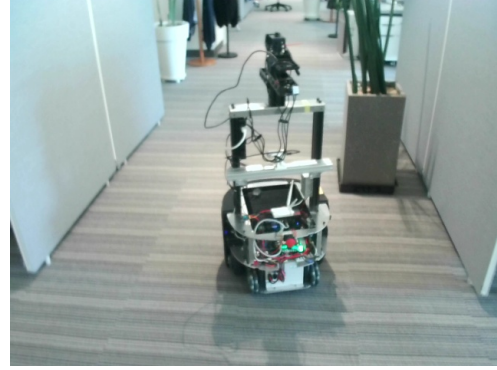


Fig. 1: The robot and environment used in the real-world experiments of Section IV.

rather than just the mean of that distribution. By applying an appropriate *risk measure*, which is simply a mapping from this distribution of rewards to a real number, distributional RL algorithms can infer risk-averse or risk-seeking policies. Distributional RL has shown superior sample efficiency and performance on arcade games [8], simulated robotics benchmarks [9], [10], and real-world grasping tasks [11]. However, the theoretical basis for these excellent results is not understood [12], and it is not clear that these advantages would necessarily extend to navigation tasks. Moreover, one might prefer risk-averse policies in one environment, for instance to avoid scaring pedestrians, but such policies might be too risk-averse to pass through a narrow passage. So, one might need to train policies with different risk measures, suited to each environment, which would be computationally expensive and time-consuming.

In this paper, to efficiently train an agent that can adapt to multiple risk measures, we present the *risk-conditioned distributional soft actor-critic* (RC-DSAC) algorithm which learns a wide range of risk-sensitive policies concurrently. In our experiments, RC-DSAC showed superior performance and safety compared with both non-distributional and distributional baselines. It could also adapt its policy to different risk measures without retraining.

In summary, our main contributions are:

- A novel navigation algorithm based on distributional RL, that can learn a variety of risk-sensitive policies concurrently;
- Improved performance over baselines in multiple simulation environments;
- Generalization to a wide range of risk measures at run-time.

The next section discusses related work. Subsequent sections, explain our method and present experiments demonstrating its effectiveness.

II. RELATED WORK

A. Risk in Mobile-Robot Navigation

Although this paper takes a deep RL approach to safe and low-risk robot navigation, there is a vast literature on classical model-predictive-control (MPC) and graph-search approaches. This literature considers diverse sources of risk, ranging from simple sensor noise and occlusion [13], [14], to uncertainty about the traversability of the edges (e.g. doors) of a navigation graph [15], and the unpredictability of pedestrian movements [16].

This literature has explored a wide variety of risk measures, ranging from collision probabilities [17] used as chance constraints [18], to entropic risk [19]. Interestingly, [19] took a hybrid approach, coupling deep learning for pedestrian motion prediction with nonlinear MPC, arguing that only such a hybrid approach allows a robot’s risk-metric parameters to be changed at run-time, unlike approaches relying on RL. To the contrary, the results of our paper demonstrate that such run-time parameter-tuning is straightforward for deep RL.

While [3] recently argued that *coherent risk measures* are well-suited to robotics, and we use coherent risk measures in this paper, the only work that we are aware of that uses a coherent risk-metric coupled with MPC for robot navigation is [20], which extends its authors’ previous work [21], in which risk measures were applied to inventory control.

B. Deep RL for Mobile-Robot Navigation

Deep RL has received much attention in the field of mobile-robot navigation due to its success in many game [22], [23] and robotics [11], [24]–[26] domains. Compared to classic approaches such as MPC, RL methods are known to be able to infer optimal actions without expensive trajectory predictions, and to perform more robustly when the cost or reward has local optima [2], [27], [28].

Recently, several deep-RL-based navigation methods have been proposed that explicitly account for risks arising from uncertainty about the environment. As individual deep networks may make overconfident predictions on far-from-distribution samples, [29] applied MC-dropout [30] and bootstrapping [31] to predict collision probabilities. An uncertainty-aware RL method was proposed in [9], which has an additional observation-prediction model, and uses the prediction variance to adjust the variance of the actions taken by the policy. Meanwhile, [32] designed ‘risk rewards’ that encourage the safe behaviour of autonomous driving policies at lane intersections, and [33] proposed switching between two RL-based driving policies, based on the estimated uncertainty about future pedestrian motions. Although these works show promising performance and improved safety in uncertain environments, they either require an additional prediction model, carefully shaped reward functions, or expensive Monte Carlo sampling at run-time.

In contrast to existing works on RL-based navigation, we use *distributional RL* to learn computationally-efficient risk-sensitive policies, without using an additional prediction model or a specifically-tuned reward function.

C. Distributional RL and Risk-Sensitive Policies

Distributional RL models the distribution of the accumulated reward, rather than just its mean. While distributional RL was proposed over a decade ago [6], it received only scant mention in a comprehensive review of safe RL [34] from 2015, and it has only been widely studied since its integration with deep learning [8], [35]. Existing distributional RL algorithms rely on the recursion

$$Z^\pi(s, a) \stackrel{D}{=} r(s, a) + \gamma Z^\pi(S', A'), \quad (1)$$

where the *random return* $Z^\pi(s, a)$ is defined as the discounted sum of rewards when starting in state s and taking action a under policy π , the notation $A \stackrel{D}{=} B$ indicates that the random variables A and B have identical distributions, $r(s, a)$ is the random reward given the state-action pair, $\gamma \in [0, 1)$ is the discount factor, the random state S' follows the transition distribution given (s, a) , and the random action A' is drawn from policy π in state S' .

Empirically, distributional RL algorithms have shown superior performance and sample efficiency in many game domains [7], [36]. It has been argued that this is because predicting quantiles serves as an auxiliary task that enhances representation learning, but as yet there is little supporting evidence for this conjecture [12].

Of central importance to this paper, is the fact that distributional RL facilitates the learning of risk-sensitive policies. To extract a risk-sensitive policy, [8], [37] learned to predict arbitrary quantiles of the distribution of the random return, and select risk-sensitive actions by estimating various ‘distortion risk measures’ by sampling quantiles. As such sampling must be performed for each potential action, this approach is not possible for continuous action spaces. So instead, [38] recently combined the soft actor-critic (SAC) framework [24] with distributional RL, achieving a new state-of-the-art in risk-sensitive control tasks. In robotics, [39] considered a sample-based distributional policy gradient algorithm and demonstrated improved robustness to actuation noise on OpenAI Gym tasks when using coherent risk measures. Meanwhile, [11] proposed the use of distributional RL to learn risk-sensitive policies for grasping tasks, showing superior performance over non-distributional baselines on real-world grasping data.

Despite the impressive performance demonstrated in these existing works, they are all limited to learning a policy for a *single* risk measure at a time. This can be problematic since the desired risk measure can vary with the environment and situation. So, in this work, we train a single policy that can adapt to a wide variety of risk measures.

III. APPROACH

In this section, we discuss the problem formulation and proposed method in detail.

A. Problem Formulation

We consider a differential-wheeled robot navigating in two dimensions. The robot's shape is an octagon (Figure 3a), and its objective is to pass a sequence of waypoints without colliding with obstacles.

We formalize this problem as a partially-observed Markov decision process (POMDP) [40], with sets of states \mathcal{S}^{PO} , observations Ω and actions \mathcal{A} , a reward function $r : \mathcal{S}^{\text{PO}} \times \mathcal{A} \rightarrow \mathbb{R}$, and distributions for the initial state, for state $s_{t+1} \in \mathcal{S}^{\text{PO}}$ given state-action $(s_t, a_t) \in \mathcal{S}^{\text{PO}} \times \mathcal{A}$ and for observation $o_t \in \Omega$ given (s_t, a_t) . As is typical when applying RL, we treat this POMDP as a Markov decision process (MDP) with set of states \mathcal{S} given by the episode-histories of the POMDP:

$$\mathcal{S} = \{(o_0, a_0, o_1, a_1, \dots, o_T) : o_t \in \Omega, a_t \in \mathcal{A}, T \in \mathbb{Z}_{\geq 0}\}.$$

The MDP has the same action space \mathcal{A} as the POMDP, and its reward, initial-state and transition distributions are those implicitly defined by the POMDP. Note that the reward is random variable for the MDP, even though we define it as a function for the POMDP.

1) *States and Observations*: The full state, which is a member of the set \mathcal{S}^{PO} , is the location of all waypoints, coupled with the locations, velocities and accelerations of all obstacles, but real-world agents only sense a fraction of this state. So in this work, an observation

$$(o_{\text{rng}}, o_{\text{waypoint}}, o_{\text{velocity}}) \in \mathbb{R}^{180} \times \mathbb{R}^6 \times \mathbb{R}^4 =: \Omega$$

consists of range-sensor measurements describing the location of nearby obstacles, measurements of the robot's location relative of the next two waypoints, and information about the robot's velocity. Specifically, we define

$$o_{\text{rng},i} = \mathbb{I}\{d_i \in (0.01, 3) \text{ m}\} (2.5 + \log_{10} d_i),$$

where $\mathbb{I}\{\cdot\}$ is the indicator function, d_i is the distance in meters to the nearest obstacle in the angular range $[2i - 2, 2i)$ degrees, relative to the x -axis of the robot's coordinate frame, and we set $o_{\text{rng},i} = 0$ if there are no obstacles in the given direction. The waypoint observation is of the form

$$o_{\text{waypoint}} = [\log_{10} \delta_1, \cos \theta_1, \sin \theta_1, \log_{10} \delta_2, \cos \theta_2, \sin \theta_2],$$

where δ_1, δ_2 are the distances to the next waypoint and the waypoint after that, clipped to $[0.01, 100]$ m, and θ_1, θ_2 are the angles of those waypoints relative to the robot's x -axis. Lastly, the velocity observation $o_{\text{velocity}} = [v_c, \omega_c, v_u, \omega_u]$ consists of the robot's current linear and angular velocities v_c, ω_c , and the desired linear and angular velocities v_u, ω_u calculated from the agent's previous action.

2) *Actions*: We use normalized two-dimensional vectors $u = (u_0, u_1) \in [-1, 1]^2 =: \mathcal{A}$ as actions, in terms of which the desired linear and angular velocity of the robot are

$$\begin{aligned} v_u &= w_{\text{minv}}(1 - u_0)/2 + w_{\text{maxv}}(1 + u_0)/2, \\ \omega_u &= \mathbb{I}\{|w_{\text{max}\omega} u_1| \geq 15 \text{ deg/s}\} w_{\text{max}\omega} u_1, \end{aligned}$$

where $w_{\text{minv}} = -0.2 \text{ m/s}$, $w_{\text{maxv}} = 0.6 \text{ m/s}$, $w_{\text{max}\omega} = 90 \text{ deg/s}$.

These desired velocities are sent to the robot's motor controller, which clips them to the ranges $[v_c - w_{\text{accv}} \Delta t, v_c +$

$\omega_{\text{accv}} \Delta t]$ and $[\omega_c - w_{\text{acc}\omega} \Delta t, \omega_c + w_{\text{acc}\omega} \Delta t]$ for maximum accelerations $w_{\text{accv}} = 1.5 \text{ m/s}^2$ and $w_{\text{acc}\omega} = 120 \text{ deg/s}^2$, where $\Delta t = 0.02 \text{ s}$ is the control period of the motor controller. The agent's control period is larger than Δt , being uniformly sampled from $\{0.12, 0.14, 0.16\} \text{ s}$ when an episode begins in the simulation, and 0.15 s in real-world experiments.

3) *Reward*: Our reward function encourages the agent to follow the waypoints efficiently, while avoiding collisions. Omitting dependence on the state and action for brevity, the reward has the form

$$r = r_{\text{base}} + r_{\text{goal}} + r_{\text{waypoint}} \cdot r_{\text{angular}} + r_{\text{coll}}.$$

The base reward $r_{\text{base}} = -0.02$ is given at every step, to penalize the agent for the time taken to reach the goal (the last waypoint), and $r_{\text{goal}} = 10$ is given when the distance between the agent and the goal is less than 0.15 m .

The waypoint reward is

$$r_{\text{waypoint}} = \max\{-0.1, \max\{0, v_c\} \cos \theta_1\},$$

where θ_1 is the angle of the next waypoint relative to the robot's x -axis and v_c is the current linear velocity. We set r_{waypoint} to zero when the agent is in contact with an obstacle.

Reward r_{angular} encourages navigation in straight lines,

$$r_{\text{angular}} = \begin{cases} 1.2 & \text{if } |\omega_u| < 15 \text{ deg/s} \\ \max\left\{0.5, 1 - \frac{|\omega_u|}{(120 \text{ deg/s})}\right\} & \text{otherwise,} \end{cases}$$

and $r_{\text{coll}} = -10$ is given if the agent collides with an obstacle.

4) *Risk-Sensitive Objective*: As in (1), let $Z^\pi(s, a)$ be the *random return* given by

$$Z^\pi(s, a) = \sum_{t=0}^{\infty} \gamma^t r(S_t, A_t), \quad (2)$$

where $(S_t, A_t)_{t \in \mathbb{Z}_{\geq 0}}$ is the random state-action sequence with $(S_0, A_0) = (s, a)$, given by the MDP's transition distribution and policy π , and $\gamma \in [0, 1)$ is the discount factor.

There are two main approaches to defining risk-sensitive decisions. Either one defines a *utility function* $U : \mathbb{R} \rightarrow \mathbb{R}$, as in [41] and selects an action a that maximizes $\mathbb{E}U(Z^\pi(s, a))$ when in state s . Alternatively, as in [8], [42] one considers the *quantile function* of Z^π , defined by $Z^\pi_\tau(s, a) := \inf\{z \in \mathbb{R} : \mathbb{P}(Z^\pi(s, a) \leq z) \geq \tau\}$ for *quantile fraction* $\tau \in [0, 1]$. Then one defines a *distortion function*, which is a mapping $\psi : [0, 1] \rightarrow [0, 1]$ from quantile fractions to quantile fractions, and selects an action a that maximizes the *distortion risk measure* $\mathbb{E}_{\tau \sim U([0, 1])} Z^\pi_{\psi(\tau)}(s, a)$ when in state s .

In this work, we focus on two distortion risk measures, each with a scalar parameter β that we call the *risk-measure parameter*. The first is the widely-used [4], [8], [38] conditional value-at-risk (CVaR), which is the expectation of the fraction β of least-favourable random returns, and corresponds to the distortion function

$$\psi^{\text{CVaR}}(\tau; \beta) := \beta \tau \quad \text{for } \beta \in (0, 1].$$

Lower β results in a more risk-averse policy and $\beta = 1$ gives a risk-neutral policy. The second is the power-law risk measure, given by the distortion function

$$\psi^{\text{pow}}(\tau; \beta) := 1 - (1 - \tau)^{1/(1-\beta)} \quad \text{for } \beta < 0,$$

motivated by its good performance in grasping experiments [11]. For the given parameter ranges, both risk measures are coherent in the sense of [43].

B. Risk-Conditioned Distributional Soft Actor-Critic

To efficiently learn a wide range of risk-sensitive policies, we propose the *risk-conditioned distributional soft actor-critic* (RC-DSAC) algorithm.

1) *Soft Actor-Critic Algorithm*: Our algorithm is based on the *soft actor-critic* (SAC) algorithm [24], the term ‘soft’ indicating *entropy-regularized*. SAC maximizes the accumulated rewards and the entropy of the policy jointly:

$$J(\pi) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t [r(s_t, a_t) + \alpha H(\pi(\cdot|s_t))] \right], \quad (3)$$

where the expectation is over state-action sequences given by the policy π and transition distribution, $\alpha \in \mathbb{R}_{\geq 0}$ is a temperature parameter which trades-off the optimization of reward and entropy, and $H(p(\cdot)) := -\mathbb{E}_{a \sim p(a)} \log p(a)$ denotes the entropy of a distribution over actions which is assumed to have a probability density $p(\cdot)$.

SAC has a *critic* network that learns a soft state-action value function $Q^{\pi} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, using the soft Bellman operator

$$T^{\pi} Q^{\pi}(s_t, a_t) := \mathbb{E}_{\pi} \left[r(s_t, a_t) + \gamma (Q^{\pi}(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}|s_{t+1})) \mid s_t, a_t \right],$$

and an *actor* network that minimizes the Kullback-Leibler divergence between the policy and a distribution given by the exponential of the soft value function,

$$\pi_{\text{new}} = \arg \min_{\pi' \in \Pi} \mathbb{E}_{s \sim \mathcal{D}^{\pi_{\text{old}}}} \left[D_{\text{KL}} \left(\pi'(\cdot|s) \parallel \frac{e^{\frac{Q^{\pi_{\text{old}}}(s, \cdot)}}{\alpha}}}{Z_{\text{part.}}^{\pi_{\text{old}}}(s)} \right) \right],$$

where Π is the set of policies that can be represented by the actor network, \mathcal{D}^{π} is the distribution over states induced by policy π and the transition distribution, which is approximated in practice by experience replay, and $Z_{\text{part.}}^{\pi_{\text{old}}}(s_t)$ is the partition function normalizing the distribution.

In practice, the reparameterization trick is often used. In that case, SAC samples actions as $a_t = f(s_t, \epsilon_t)$ where $f(\cdot, \cdot)$ is the mapping implemented by the actor network, and ϵ_t is a sample from a fixed distribution like a spherical Gaussian \mathcal{N} . The policy objective then has the form

$$J(\pi) = \mathbb{E}_{s \sim \mathcal{D}^{\pi}, \epsilon \sim \mathcal{N}} [Q(s, f(s, \epsilon)) - \alpha \log \pi(f(s, \epsilon)|s)]. \quad (4)$$

For more details about SAC, we refer the reader to [24], [44].

2) *Distributional SAC and Risk-Sensitive Policies*: To capture the full distribution of accumulated rewards, rather than just its mean, [38] recently proposed *distributional SAC* (DSAC). As in previous work on distributional RL [7], [8], [37], DSAC uses quantile regression to learn this distribution. Such previous work is limited to finite action spaces, and DSAC overcomes this limitation using ideas from SAC.

Rather than using the random return Z^{π} from equation (2), DSAC works with the *soft random return* appearing in (3), given by

$$Z^{\alpha, \pi}(s, a) := \sum_{t=0}^{\infty} \gamma^t [r(S_t, A_t) - \alpha \log \pi(A_t|S_t)],$$

where $(S_t, A_t)_{t \in \mathbb{Z}_{\geq 0}}$ is as in (2).

Like SAC, the DSAC algorithm has an actor and a critic. To train the critic, some quantile fractions τ_1, \dots, τ_N and $\tau'_1, \dots, \tau'_{N'}$ are sampled independently, and the critic minimizes the loss

$$L(s_t, a_t, r_t, s_{t+1}) = \frac{1}{N'} \sum_{i=1}^N \sum_{j=1}^{N'} \rho_{\tau_i}(\delta_t^{\tau_i, \tau'_j}), \quad (5)$$

where for $x \in \mathbb{R}$, the quantile regression loss is

$$\rho_{\tau}(x) = |\tau - \mathbb{I}\{x < 0\}| \min\{x^2, 2|x| - 1\}/2, \quad (6)$$

and the temporal difference is

$$\delta_t^{\tau, \tau'} = r_t + \gamma [\hat{Z}'_{\tau'}(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}|s_{t+1})] - \hat{Z}_{\tau}(s_t, a_t),$$

where (s_t, a_t, r_t, s_{t+1}) is a transition from the replay buffer, $\hat{Z}_{\tau}(s, a)$ is the output of the critic, which is an estimate of the τ -quantile of $Z^{\alpha, \pi}(s, a)$, and $\hat{Z}'_{\tau'}(s, a)$ is the output of a delayed version of the critic known as the *target critic* [24].

To train a risk-sensitive actor network, DSAC works with a distortion function ψ . Rather than directly maximizing the corresponding distortion risk measure, DSAC substitutes $Q(s, a) = \hat{\mathbb{E}}_{\tau \sim U([0,1])} Z_{\psi(\tau)}(s, a)$ in equation (4), where $\hat{\mathbb{E}}$ denotes the average of a sample.

3) *Risk-Conditioned DSAC*: Although risk-sensitive policies learnt by DSAC show promising results [38] in multiple simulation environments, DSAC can learn only one type of risk-sensitive policy at a time. This may be problematic for mobile-robot navigation if the appropriate risk-measure parameter differs with the environment, and users wish to tune it at run-time.

To address this issue, we propose the *risk-conditioned distributional SAC* (RC-DSAC) algorithm, which extends DSAC to learn a wide range of risk-sensitive policies concurrently and can change its risk-measure parameter without retraining. RC-DSAC learns risk-adaptable policies for a distortion function $\psi(\cdot; \beta)$ with parameter β , by supplying β as an input to the policy $\pi(\cdot|s, \beta)$, the critic $\hat{Z}_{\tau}(s, a; \beta)$, and the target critic $\hat{Z}'_{\tau'}(s, a; \beta)$. Specifically, the critic’s objective (5) becomes

$$L(s_t, a_t, r_t, s_{t+1}, \beta) = \frac{1}{N'} \sum_{i=1}^N \sum_{j=1}^{N'} \rho_{\tau_i}(\delta_t^{\tau_i, \tau'_j, \beta}), \quad (7)$$

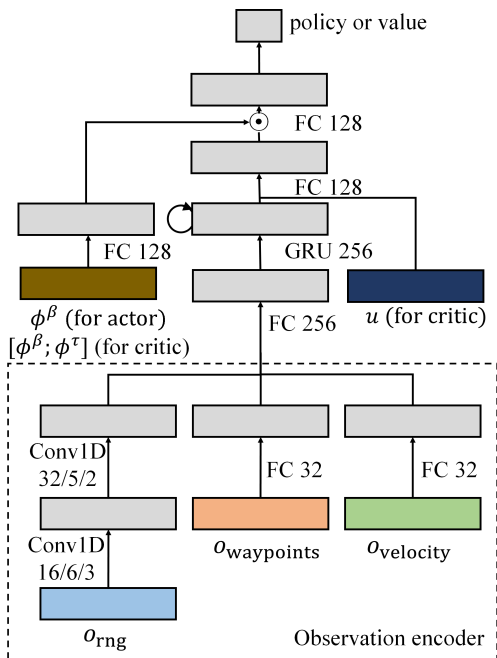


Fig. 2: Architecture of the networks used in RC-DSAC. FC denotes a fully-connected layer, Conv1D denotes a one-dimensional convolutional layer with the given number of channels/kernel_size/stride, and GRU denotes a gated recurrent unit [45]. Multiple arrows pointing to a single block indicate concatenation and \odot denotes element-wise multiplication.

where $\rho_\tau(\cdot)$ is as in (6) and the temporal difference is

$$\delta_t^{\tau, \tau', \beta} = r_t + \gamma [\hat{Z}'_{\tau'}(s_{t+1}, a_{t+1}; \beta) - \alpha \log \pi(a_{t+1} | s_{t+1}, \beta)] - \hat{Z}_\tau(s_t, a_t; \beta),$$

and the actor’s objective (4) becomes

$$J(\pi) = \mathbb{E}_{s \sim \mathcal{D}^\pi, \epsilon \sim \mathcal{N}, \beta \sim \mathcal{B}} [Q(s, f(s, \epsilon, \beta); \beta) - \alpha \log \pi(f(s, \epsilon, \beta) | s)], \quad (8)$$

where $Q(s, a; \beta) = \hat{\mathbb{E}}_{\tau \sim U([0, 1])} \hat{Z}_{\psi(\tau; \beta)}(s, a; \beta)$ and \mathcal{B} is a distribution for sampling β as we now explain.

During training, the risk-measure parameter β is uniformly sampled from $\mathcal{B} = U([0, 1])$ for ψ^{CVaR} , and $U([-2, 0])$ for ψ^{POW} . As in other RL algorithms, each iteration has a *data-collection* phase and a *model-update* phase. In the data-collection phase, we sample β at the start of each episode and fix it until the episode’s end. For the model-update phase, we explore the following two alternatives. The first alternative, called *stored*, stores the β used in data-collection in the experience-replay buffer, and only uses that stored β for updates. The second alternative, called *resampling*, samples a new β for each experience in a mini batch at every iteration.

4) *Network Architectures*: We represent τ and β using a cosine embedding, and use element-wise multiplication to fuse these with information about the observation and quantile fraction (Figure 2).

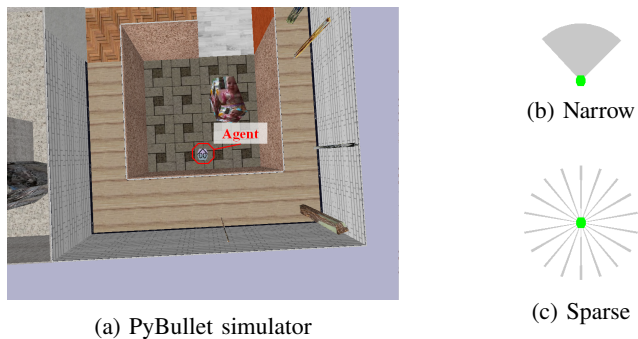


Fig. 3: (a) An example of our procedurally-generated environment. (b,c) Sensor settings: green octagons denote the robot and grey areas are the sensors’ fields of view.

As in DSAC [38], only the critic network of RC-DSAC depends on τ . However, both the actor and critic networks of RC-DSAC depend on β . So, we calculate embeddings $\phi^\beta \in \mathbb{R}^{64}$, $\phi^\tau \in \mathbb{R}^{64}$, with elements $\phi_i^\beta = \cos(\pi i \beta)$ and $\phi_i^\tau = \cos(\pi i \tau)$. Then, we apply element-wise multiplication

$$g^{\text{actor}}(o_{0:t}) \odot g^{\text{actorRisk}}(\phi^\beta)$$

to the actor network and

$$g^{\text{critic}}(o_{0:t}, u_t) \odot g^{\text{criticRisk}}([\phi^\beta; \phi^\tau])$$

to the critic network, where $g^{\text{actor}}(o_{0:t}), g^{\text{critic}}(o_{0:t}, u_t) \in \mathbb{R}^{128}$ are the embeddings of the observation history (and the current action for the critic) calculated using a gated recurrent unit (GRU) [45] and a fully-connected layer, $g^{\text{actorRisk}}: \mathbb{R}^{64} \rightarrow \mathbb{R}^{128}$ and $g^{\text{criticRisk}}: \mathbb{R}^{128} \rightarrow \mathbb{R}^{128}$ are fully-connected layers, and $[\phi^\beta; \phi^\tau]$ is the concatenation of vectors ϕ^β and ϕ^τ .

IV. EXPERIMENTS

In this section, we describe the simulation environment used for the training. Then we compare the performance of our method against baselines and demonstrate the trained policy using a real-world robot.

A. Training Environment

We use the same procedural environment-generation algorithm as in [25], and use PyBullet [46] to simulate the robot dynamics, as shown in Figure 3 (a). To increase the throughput of data collection, we run 10 simulations in parallel. Specifically, for each environment generated, we run 10 episodes in parallel, where the episodes involve agents with distinct start and goal positions, as well as distinct risk-metric parameters β . Each episode terminates after 1,000 steps, and a new goal is sampled when an agent reaches its goal.

To study the impact of partial observation on our method, we conduct experiments using two different sensor configurations, as shown in Figure 3 (b,c). The *narrow* setting zeros out $o_{\text{rng}, 22:158}$, and the *sparse* setting zeros out all $o_{\text{rng}, i}$ except those with $i \equiv 0 \pmod{10}$.

B. Training Agents

We compare the performance of RC-DSAC with SAC [24] and DSAC [38]. We also compare with Algorithm 1 in [25], applied to this paper’s reward function, calling this method *reward-component-weight randomization* (RCWR).

We train two RC-DSAC agents, one for each of the distortion functions ψ^{CVaR} and ψ^{pow} . Then RC-DSAC with ψ^{CVaR} is evaluated for $\beta \in \{0.25, 0.5, 0.75, 1\}$, and RC-DSAC with ψ^{pow} is evaluated for $\beta \in \{-2, -1.5, -1, -0.5\}$.

For DSAC, we use ψ^{CVaR} with $\beta \in \{0.25, 0.75\}$, and ψ^{pow} with $\beta \in \{-2, -1\}$, each DSAC agent being trained and evaluated for a single β .

For RCWR, we use only one *navigation parameter* [25] $w_{\text{coll}} \sim U([0.1, 2])$. The reward r_{coll} is replaced by $w_{\text{coll}}r_{\text{coll}}$, when calculating the reward r , with higher values of w_{coll} making an agent more collision-averse while still remaining risk-neutral. We use $w_{\text{coll}} \in \{1, 1.5, 2\}$ for evaluation.

All baselines use the same network architecture as RC-DSAC, with the following exceptions. DSAC does not use $g^{\text{actorRisk}}$, and $g^{\text{criticRisk}}$ depends only on ϕ^τ . RCWR has an extra 32-dimensional fully-connected layer in its observation encoder for w_{coll} . Lastly, RCWR and SAC use neither $g^{\text{actorRisk}}$ nor $g^{\text{criticRisk}}$.

We use the hyperparameters in Table I for all algorithms.

TABLE I: Hyperparameters

Parameter	Value	Parameter	Value
Learning rate	3×10^{-4}	Quantile fraction samples (N, N')	16
Discount factor (γ)	0.99	Experience replay buffer size	5×10^6
Target network update coefficient	0.001	Mini-batch size	100
Entropy target [24]	-2	GRU unroll	64

We train each algorithm for 100,000 weight updates (5,000 episodes in 500 environments). Then we evaluate the algorithms on 50 environments not seen in training. We evaluate for 10 episodes per environment, with agents having distinct start and goal positions, but having a common value for β or w_{coll} .

To ensure fairness and reproducibility, we use fixed random seeds for training and evaluation, so different algorithms are trained and evaluated on exactly the same sequences of environments, and starting/goal positions.

C. Performance Comparison

Table II presents the mean and standard deviation of the number of collisions and reward of each method, averaged over the 500 episodes across the 50 evaluation environments.

RC-DSAC with ψ^{pow} and $\beta = -1$ had the highest rewards in the narrow setting, and RC-DSAC with ψ^{pow} and $\beta = -1.5$ had the fewest collisions in the both settings.

The risk-sensitive algorithms (DSAC, RC-DSAC) all had fewer collisions than SAC, and some of them could achieve this while attaining a higher reward. Also, the results for RCWR suggest that distributional risk-aware approaches can

TABLE II: Performance evaluation against baselines.

Agent	ψ	β	Narrow		Sparse	
			Collisions	Rewards	Collisions	Rewards
RC-DSAC (resample)	CVaR	0.25	0.67 \pm 2.06	403.9 \pm 186.2	0.19 \pm 0.48	487.8 \pm 88.2
		0.5	0.59 \pm 1.03	451.3 \pm 125.4	0.29 \pm 0.62	512.0 \pm 54.8
		0.75	0.81 \pm 1.75	452.0 \pm 145.9	0.42 \pm 0.93	507.6 \pm 65.1
		1	1.15 \pm 2.48	458.8 \pm 140.3	0.55 \pm 1.03	505.2 \pm 60.1
	pow	-2	0.50 \pm 0.84	509.4 \pm 99.2	0.21 \pm 0.68	473.4 \pm 113.9
		-1.5	0.48 \pm 0.89	511.7 \pm 98.8	0.17 \pm 0.53	479.0 \pm 107.4
		-1	0.58 \pm 1.36	514.7 \pm 96.4	0.21 \pm 0.58	482.2 \pm 101.9
		-0.5	0.68 \pm 1.18	506.7 \pm 113.3	0.23 \pm 0.75	488.3 \pm 104.2
RC-DSAC (stored)	CVaR	0.25	0.68 \pm 3.47	443.5 \pm 168.3	0.37 \pm 0.68	494.7 \pm 89.3
		0.5	1.00 \pm 5.14	397.7 \pm 173.2	0.38 \pm 0.80	499.4 \pm 87.7
		0.75	1.10 \pm 2.27	431.0 \pm 152.3	0.39 \pm 0.77	501.0 \pm 86.0
		1	1.59 \pm 8.09	298.4 \pm 246.9	1.00 \pm 1.63	477.7 \pm 97.6
	pow	-2	0.87 \pm 3.90	465.0 \pm 151.6	0.42 \pm 0.72	492.3 \pm 84.5
		-1.5	0.73 \pm 2.11	471.4 \pm 130.0	0.68 \pm 1.32	468.4 \pm 335.8
		-1	1.13 \pm 3.40	460.1 \pm 122.2	0.58 \pm 0.96	504.5 \pm 80.6
		-0.5	0.95 \pm 3.30	499.4 \pm 112.9	1.12 \pm 1.52	496.7 \pm 84.0
DSAC	CVaR	0.25	1.05 \pm 1.75	431.9 \pm 127.6	0.76 \pm 1.18	417.2 \pm 117.8
		0.75	0.72 \pm 3.00	299.6 \pm 199.2	0.63 \pm 1.03	515.4 \pm 74.1
	pow	-2	1.14 \pm 4.02	469.2 \pm 212.6	0.54 \pm 1.29	525.5 \pm 76.8
		-1	0.73 \pm 2.57	499.4 \pm 115.7	1.80 \pm 1.80	513.3 \pm 84.5
RCWR	$w_{\text{coll}} = 2$	1.58 \pm 2.68	488.2 \pm 122.5	0.81 \pm 1.08	506.1 \pm 81.1	
	$w_{\text{coll}} = 1.5$	1.50 \pm 2.39	491.7 \pm 108.8	1.17 \pm 1.71	491.9 \pm 101.2	
	$w_{\text{coll}} = 1$	1.60 \pm 2.55	493.7 \pm 116.7	1.23 \pm 1.59	490.8 \pm 93.5	
SAC	-	1.76 \pm 2.02	476.7 \pm 105.4	1.62 \pm 2.48	491.8 \pm 103.5	

be more effective than simply increasing the penalty for collisions.

We compare DSAC with the two alternative implementations of RC-DSAC by averaging over both risk measures, but only for the two values of β on which DSAC was evaluated. In the narrow setting, RC-DSAC (stored) had a comparable number of collisions (0.95 vs. 0.91) but higher rewards (449.9 vs. 425.0) than DSAC, whereas in the sparse setting RC-DSAC (stored) had fewer collisions (0.44 vs. 0.68) but comparable rewards (498.1 vs. 492.9). Overall, RC-DSAC (resampling) had the fewest collisions (0.64 in the narrow setting and 0.26 in the sparse setting), and attained the highest rewards in the narrow setting (470.0). This shows the algorithm’s ability to adapt to a wide range of risk-measure parameters, without the retraining required by DSAC.

In addition, the number of collisions made by RC-DSAC shows a clear positive correlation with β , for the CVaR risk measure. One would expect this, as low β corresponds to risk aversion.

D. Real-World Experiments

To demonstrate the proposed method in the real world, we build a mobile-robot platform and test the agents trained in simulation (as described in Section IV-B), in a challenging office environment (Figure 1). The robot has four depth cameras on its front, and point cloud data from these sensors is mapped into the observation o_{rng} corresponding to the narrow setting. Then we deploy RC-DSAC (resampling) and baseline agents. For each agent, we ran two experiments in a course of length 53.7m, making a run forward and another in the reverse direction.

Table III presents the number of collisions and required time to reach the goal for each agent. As can be seen, SAC had more collisions than distributional risk-averse agents. DSAC had no collisions throughout the experiments but

TABLE III: Results of real-world experiment.

Agent	ψ	β	Forward		Reverse	
			Collision	Required Time (s)	Collision	Required Time (s)
RC-DSAC	CVaR	0.25	0	107	0	114
		0.75	0	112	1	109
	pow	-2	0	110	0	116
		-1	0	107	1	107
DSAC	CVaR	0.25	0	141	0	128
		0.75	0	104	0	114
	pow	-2	0	109	0	104
		-1	0	111	0	104
SAC	-	-	3	115	2	111

showed over-conservative behaviour and took the longest time to reach the goal with ψ^{CVaR} and $\beta = 0.25$. RC-DSAC performed competitively with DSAC except minor collisions in less risk-averse modes, and could adapt its behaviour according to β .

We refer the supplementary video for the detailed trajectories and characteristics of each agent.

V. CONCLUSION

This paper proposed a novel distributional-RL method that produces agents that can adapt to a wide range of risk measures at run-time. In our experiments, this method showed superior performance over the baselines, as well as adjustable risk-sensitivity. We also demonstrated the method using a real-world robot.

ACKNOWLEDGEMENT

We thank Tomi Silander and the NAVER LABS Robotics Group for their help with proofreading and the experiments.

REFERENCES

- [1] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, “PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 5113–5120.
- [2] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, “Learning navigation behaviors end-to-end with AutoRL,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.
- [3] A. Majumdar and M. Pavone, “How should a robot assess risk? Towards an axiomatic theory of risk in robotics,” in *Robotics Research*, Springer, 2020, pp. 75–84.
- [4] Y. Chow, A. Tamar, S. Mannor, and M. Pavone, “Risk-sensitive and robust decision-making: A CVaR optimization approach,” in *Advances in Neural Information Processing Systems*, 2015, pp. 1522–1530.
- [5] D. Kahneman and A. Tversky, “Prospect theory: An analysis of decision under risk,” in *Handbook of the Fundamentals of Financial Decision Making: Part I*, World Scientific, 2013, pp. 99–127.
- [6] T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, and T. Tanaka, “Nonparametric return distribution approximation for reinforcement learning,” in *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010, pp. 799–806.
- [7] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, “Distributional reinforcement learning with quantile regression,” *arXiv preprint arXiv:1710.10044*, 2017.
- [8] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, “Implicit quantile networks for distributional reinforcement learning,” *arXiv preprint arXiv:1806.06923*, 2018.
- [9] T. Fan, P. Long, W. Liu, J. Pan, R. Yang, and D. Manocha, “Learning resilient behaviors for navigation under uncertainty,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 5299–5305.
- [10] Y. Yue, Z. Wang, and M. Zhou, “Implicit distributional reinforcement learning,” in *Advances in Neural Information Processing Systems*, 2020.
- [11] C. Bodnar, A. Li, K. Hausman, P. Pastor, and M. Kalakrishnan, “Quantile QT-Opt for risk-aware vision-based robotic grasping,” in *Robotics: Science and Systems*, 2020.
- [12] C. Lyle, M. G. Bellemare, and P. S. Castro, “A comparative analysis of expected and distributional reinforcement learning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4504–4511.
- [13] J. Lunenburg, R. van de Molengraft, and M. Steinbuch, “A representation method based on the probability of collision for safe robot navigation in domestic environments,” *Autonomous Robots*, vol. 42, no. 3, pp. 601–614, 2018.
- [14] S.-K. Kim, R. Thakker, and A.-A. Agha-Mohammadi, “Bi-directional value learning for risk-aware planning under uncertainty,” *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2493–2500, 2019.
- [15] L. Nardi and C. Stachniss, “Long-term robot navigation in indoor environments estimating patterns in traversability changes,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 300–306.
- [16] D. Mehta, G. Ferrer, and E. Olson, “Backprop-PPDM: Faster risk-aware policy evaluation through efficient gradient optimization,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 1740–1746.
- [17] B. Axelrod, L. P. Kaelbling, and T. Lozano-Pérez, “Provably safe robot navigation with obstacle uncertainty,” *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1760–1774, 2018.
- [18] L. Blackmore, M. Ono, and B. C. Williams, “Chance-constrained optimal path planning with obstacles,” *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1080–1094, 2011.
- [19] H. Nishimura, B. Ivanovic, A. Gaidon, M. Pavone, and M. Schwager, “Risk-sensitive sequential action control with multi-modal human trajectory forecasting for safe crowd-robot interaction,” *arXiv preprint arXiv:2009.05702*, 2020.

- [20] A. Hakobyan and I. Yang, “Wasserstein distributionally robust motion planning and control with safety constraints using conditional value-at-risk,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 490–496.
- [21] S. Samuelson and I. Yang, “Safety-aware optimal control of stochastic systems using conditional value-at-risk,” in *2018 Annual American Control Conference (ACC)*, IEEE, 2018, pp. 6285–6290.
- [22] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, *et al.*, “Impala: Scalable distributed deep-RL with importance weighted actor-learner architectures,” *arXiv preprint arXiv:1802.01561*, 2018.
- [23] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, “Mastering the game of Go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.
- [24] 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.
- [25] J. Choi, C. Dance, J. Kim, K. Park, J. Han, J. Seo, and M. Kim, “Fast adaptation of deep reinforcement learning-based navigation skills to human preference,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 3363–3370.
- [26] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, “Solving Rubik’s cube with a robot hand,” *arXiv preprint arXiv:1910.07113*, 2019.
- [27] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, “Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning,” *arXiv preprint arXiv:1709.10082*, 2017.
- [28] Y. F. Chen, M. Everett, M. Liu, and J. P. How, “Socially aware motion planning with deep reinforcement learning,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1343–1350.
- [29] B. Lötjens, M. Everett, and J. P. How, “Safe reinforcement learning with model uncertainty estimates,” in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 8662–8668.
- [30] Y. Gal and Z. Ghahramani, “Dropout as a Bayesian approximation: Representing model uncertainty in deep learning,” in *International Conference on Machine Learning (ICML)*, 2016, pp. 1050–1059.
- [31] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, “Deep exploration via bootstrapped DQN,” in *Advances in Neural Information Processing Systems*, 2016, pp. 4026–4034.
- [32] D. Kamran, C. F. Lopez, M. Lauer, and C. Stiller, “Risk-aware high-level decisions for automated driving at occluded intersections with reinforcement learning,” *arXiv preprint arXiv:2004.04450*, 2020.
- [33] K. D. Katyal, G. D. Hager, and C.-M. Huang, “Intent-aware pedestrian prediction for adaptive crowd navigation,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 3277–3283.
- [34] J. García and F. Fernández, “A comprehensive survey on safe reinforcement learning,” *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.
- [35] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tirumala, A. Muldal, N. Heess, and T. Lillicrap, “Distributed distributional deterministic policy gradients,” *arXiv preprint arXiv:1804.08617*, 2018.
- [36] M. G. Bellemare, W. Dabney, and R. Munos, “A distributional perspective on reinforcement learning,” *arXiv preprint arXiv:1707.06887*, 2017.
- [37] D. Yang, L. Zhao, Z. Lin, T. Qin, J. Bian, and T.-Y. Liu, “Fully parameterized quantile function for distributional reinforcement learning,” in *Advances in Neural Information Processing Systems*, 2019, pp. 6193–6202.
- [38] X. Ma, L. Xia, Z. Zhou, J. Yang, and Q. Zhao, “DSAC: Distributional soft actor critic for risk-sensitive reinforcement learning,” *arXiv preprint arXiv:1710.10044*, 2020.
- [39] R. Singh, Q. Zhang, and Y. Chen, “Improving robustness via risk averse distributional reinforcement learning,” in *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, A. M. Bayen, A. Jadbabaie, G. Pappas, P. A. Parrilo, B. Recht, C. Tomlin, and M. Zeilinger, Eds., 2020, pp. 958–968.
- [40] K. J. Åström, “Optimal control of Markov processes with incomplete state information,” *Journal of Mathematical Analysis and Applications*, vol. 10, no. 1, pp. 174–205, 1965.
- [41] O. Morgenstern and J. Von Neumann, *Theory of Games and Economic Behavior*. Princeton university press, 1953.
- [42] M. E. Yaari, “The dual theory of choice under risk,” *Econometrica: Journal of the Econometric Society*, pp. 95–115, 1987.
- [43] P. Artzner, F. Delbaen, J.-M. Eber, and D. Heath, “Coherent measures of risk,” *Mathematical Finance*, vol. 9, no. 3, pp. 203–228, 1999.
- [44] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” *arXiv preprint arXiv:1801.01290*, 2018.
- [45] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, “On the properties of neural machine translation: Encoder-decoder approaches,” *arXiv preprint arXiv:1409.1259*, 2014.
- [46] E. Coumans and Y. Bai, “PyBullet, a Python module for physics simulation for games, robotics and machine learning,” *GitHub repository*, 2016.