

Dependability Analysis of Deep Reinforcement Learning based Robotics and Autonomous Systems through Probabilistic Model Checking

Yi Dong¹, Xingyu Zhao¹ and Xiaowei Huang¹

Abstract—While Deep Reinforcement Learning (DRL) provides transformational capabilities to the control of Robotics and Autonomous Systems (RAS), the black-box nature of DRL and uncertain deployment environments of RAS pose new challenges on its dependability. Although existing works impose constraints on the DRL policy to ensure successful completion of the mission, it is far from adequate to assess the DRL-driven RAS in a holistic way considering all dependability properties. In this paper, we formally define a set of dependability properties in temporal logic and construct a Discrete-Time Markov Chain (DTMC) to model the dynamics of risk/failures of a DRL-driven RAS interacting with the stochastic environment. We then conduct Probabilistic Model Checking (PMC) on the designed DTMC to verify those properties. Our experimental results show that the proposed method is effective as a holistic assessment framework while uncovering conflicts between the properties that may need trade-offs in training. Moreover, we find that the standard DRL training cannot improve dependability properties, thus requiring bespoke optimisation objectives. Finally, our method offers sensitivity analysis of dependability properties to disturbance levels from environments, providing insights for the assurance of real RAS.

I. INTRODUCTION

The major obstacle to reaping the benefits of Robotics and Autonomous Systems (RAS) is the assurance of their dependability [1]—an umbrella concept holistically covering aspects of a system’s quality, including reliability, safety, availability, and performance [2]. Since first proposed in 2013 [3], Deep Reinforcement Learning (DRL) has received significant attention in many applications [4]–[6]. DRL, yielding a control policy for RAS, is replacing the traditional control algorithms, thanks to its ability to deal with complex and nonlinear problems [7]–[9]. However, the existing research on safe DRL mainly focuses on eliminating unsafe “state-action” pairs that may cause failures [10]–[13], without considering more involved dependability properties, including robustness, resilience and safe requirements [14]. For example, a policy that consistently leads the vehicle to go in circles is safe but unacceptable; meanwhile, a policy that recovers quickly from an unsafe state is better than a slower strategy. There is an urgent need to (1) consider a set of dependability properties that concerns not only the completion of missions but also the quality of the completions, and (2) develop methods to evaluate and certify the dependable use of DRL-driven RAS in critical applications [15]. This paper addresses this need through Probabilistic Model Checking (PMC). First, a risk-aware Discrete Time Markov Chain (DTMC) is constructed to model the interactions of the DRL

agent with stochastic environments during the execution. Second, an off-the-shelf PMC tool is applied to the DTMC to verify a set of *quantitative dependability properties* expressed in the temporal logic Probabilistic Computational Tree Logic (PCTL).

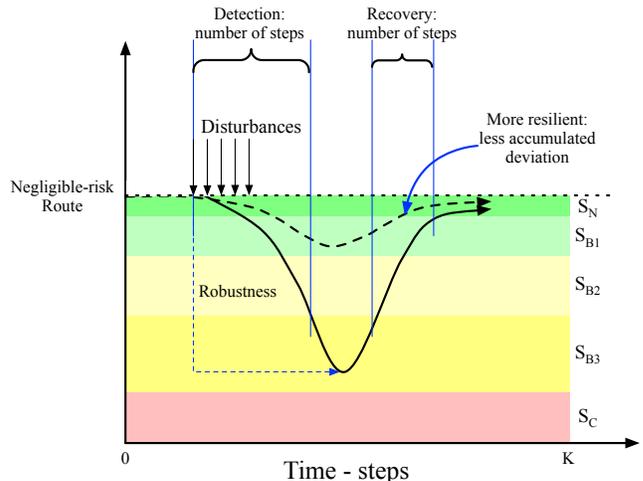


Fig. 1: Conceptualised illustration of dependability properties: safety, resilience, robustness, detection and recovery.

As illustrated in Fig. 1, we consider, in addition to the safety (i.e., completion of mission without failures), other dependability properties, including robustness, resilience, detection, and recovery. Simply speaking, in an environment where the robot’s sensory input may be subject to disturbances, *robustness* expresses the ability to complete the mission regardless of the disturbance, *resilience* evaluates the accumulated deviation from a negligible-risk trajectory, *detection* concerns how fast the robot’s risky situation may deteriorate (and therefore be detected), and *recovery* concerns how soon the robot can recover from a risky situation. Different from the safe RL research, [16] which mainly concerns the reachability of error states or bad events, the above properties require explicit consideration of a sequence of states and the states on another path (i.e., the negligible-risk trajectory as in Fig. 1). Arguably, such a *holistic* evaluation of a set of quantitative properties is needed to have an in-depth understanding of the robot’s (and the DRL’s) dependability.

Given the black-box nature of DRL and that we usually do not have a formal model for the environment, it is unlikely that we can have a probabilistic model that captures all the executions. To facilitate a formal analysis, we construct a DTMC from a set of sampled trajectories that can be

¹Department of Computer Science, University of Liverpool, UK {yi.dong, xingyu.zhao, xiaowei}@liverpool.ac.uk

augmented with domain knowledge [17] and other Verification and Validation (V&V) evidence [18]. The DTMC is dedicated to risk analysis to include only states that represent different levels of risks.

In addition to evaluating the dependability of RAS, we apply our method to study the real RAS. In particular, we do sensitivity analysis on dependability properties with respect to the disturbance levels of the environment and utilise the results to provide insights on whether, in a given natural environment, a certain level of dependability of the RAS can be achieved after deployment.

In summary, the key contributions of this paper include:

- 1) A set of formally defined dependability properties that need to be evaluated before deploying the DRL-driven RAS in critical applications.
- 2) An initial framework on constructing failure process DTMCs that model the dynamics of risky situations in executing a DRL-driven RAS.
- 3) A publicly accessible repository of our proposed method with all source code, datasets and experimental results (including a real-world case study based on Turtlebot Waffle Pi).

II. RELATED WORKS

Most research in safe DRL focuses on enhancing safety and robustness by reducing potential unsafe actions, including methods for safe monitoring and adversarial training. Safe action sets are designed to avoid the unsafe states only based on the current state of the agent, including Shield [11], [12], Lyapunov method [19], [20], etc. For example, shielding methods prevent agents from making unsafe actions at each state. Although choosing an action from the bounded safe action set can return a safe action for that specific time, the correctness of the action at any specific time depends on the expected long-term accumulated rewards. For this reason, the verification of a DRL agent also needs to consider the current state and long-term rewards (and therefore, the future states). Mandlekar *et al.* [21] used actively chosen adversarial perturbations for robust policy training to improve robustness (resistance to changes) in complex environments. In [22] and [23], it is found that the DRL agent and adversarial agent can be trained in a cyclical way to avoid overfitting. Here, the mentioned manners do not consider the environment model. Therefore, to understand if a learned policy works well in an environment, we can conduct a PMC on their induced DTMC. This enables the analysis of various properties that can be expressed with PCTL.

In addition, these methods cannot ascertain whether a DRL model satisfies specific properties with provable guarantees. Verification techniques are required for this purpose, but unfortunately, due to the complexity of verification problems (NP-complete for robustness verification [24], [25] over Deep Neural Networks (DNN)), a direct verification is suffering from the scalability issue and can only work with miniature models. Compared to DNN verification, DRL verification is more complicated because it requires the consideration of not only the learned model but also the environment,

which in general does not have a formal model. An intuitive idea to make the DRL verification practical is to use an approximate model to replace the policy network [26]. For example, a decision tree based approximation model has been considered in [27]. Behzadan *et al.* proposed a new framework based on DRL to benchmark the behaviour of the collision avoidance mechanism in the worst case [10]. They verified the effectiveness of the framework by comparing the reliability of two collision avoidance mechanisms in dealing with deliberate collision attempts over, e.g., the number of collisions, return values, and the time from the start to the collision. Although these works are building equivalent models to replace the DRL models, not all dependability properties are covered like our method.

Based on traditional software systems, Zhu *et al.* developed the formal verification technology for reinforcement learning verification [28]. The proposed verification toolchain can ensure that the RL-based control policies are safe in terms of an infinite state transition system specification. In addition, to evaluate the robustness and resilience of the agent in the test phase against adversarial disturbances in a way independent of the attack type, Behzadan *et al.* proposed to measure the resilience and robustness of DRL strategies. Different from the above, we define a set of quantitative dependability properties and apply a PMC on a failure process DTMC.

III. PRELIMINARIES

A. Interaction of Robot with Environment

We use discounted infinite-horizon Markov Decision Process (MDP) to model the interaction of an agent with the environment E . An MDP is a 5-tuple $\mathcal{M}^E = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where \mathcal{S} is the state space, \mathcal{A} is the action space, $\mathcal{P}(s'|s, a)$ is a probabilistic transition, $\mathcal{R}(s, a) \in \mathbb{R}_{\geq 0}$ is a reward function, and $\gamma \in [0, 1)$ is a discount factor. A (deterministic¹) policy is $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maps from states to actions.

Based on \mathcal{M}^E , a policy π induces a trajectory distribution $\rho^{\pi, E}(\zeta)$ where $\zeta = (s_0, a_0, s_1, a_1, \dots)$ denotes a random trajectory. The state-action value function of π is defined as $Q^\pi(s, a) = \mathbb{E}_{\zeta \sim \rho^{\pi, E}}[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)]$ and the state value function of π is $V^\pi(s) = Q^\pi(s, \pi(s))$. In Section III, we will explain how to construct a DTMC to approximate $\rho^{\pi, E}(\zeta)$.

There is always some noise in the environment, which will affect the robot's perception of the environment. For each sensor signal $o_t^i \in s_t$, the existence of the disturbances suggests that the actual sensor reading may be deviated from its value in the current state. Formally, we assume that the actual state \hat{s}_t is within certain norm distance from s_t , i.e., $\|\hat{s}_t - s_t\|_p \leq d$ for some $d > 0$, where $\|\cdot\|_p$ denotes the p -norm.

B. Probabilistic Model Checking

PMC [31] has been used to analyse quantitative properties of systems across a variety of application domains, including

¹We consider Deep Deterministic Policy Gradient (DDPG) [3], [29], [30] for a reinforcement learning agent. DDPG returns a deterministic policy.

RAS [18], [32], [33]. It involves the construction of a probabilistic model, e.g., DTMC or MDP, that formally represents the behaviour of a system over time. The properties of interest are usually specified with, e.g., Linear Temporal Logic (LTL) or PCTL. Then, via model checkers, a systematic exploration and analysis are performed to check if a claimed property holds. In this paper, we adopt DTMC and PCTL whose definitions are as follows.

Definition 1 (DTMC). Let AP be a set of atomic propositions. A DTMC is a tuple (S, s_0, \mathbf{P}, L) , where S is a (finite) set of states, $s_0 \in S$ is an initial state, $\mathbf{P} : S \times S \rightarrow [0, 1]$ is a probabilistic transition matrix such that $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ for all $s \in S$, and $L : S \rightarrow 2^{AP}$ is a labelling function assigning each state with a set of atomic propositions.

Definition 2 (DTMC Reward Structure). A reward structure for DTMC $D = (S, s_0, \mathbf{P}, L)$ is a tuple $r = (r_S, r_T)$ where $r_S : S \rightarrow \mathbb{R}_{\geq 0}$ is a state reward function and $r_T : S \times S \rightarrow \mathbb{R}_{\geq 0}$ is a transition reward function.

Definition 3 (PCTL). The syntax of PCTL is defined by state formulae ϕ , path formulae ψ and reward formulae μ .

$$\begin{aligned} \phi &::= \text{true} \mid ap \mid \phi \wedge \phi \mid \neg \phi \mid P_{\bowtie p}(\psi) \mid R_{\bowtie q}^r(\mu) \\ \psi &::= \bigcirc \phi \mid \phi U \phi \\ \mu &::= C^{\leq t} \mid \diamond \phi \end{aligned}$$

where $ap \in AP, p \in [0, 1], q \in \mathbb{R}_{\geq 0}, t \in \mathbb{N}, \bowtie \in \{<, \leq, >, \geq\}$ and r is a reward structure.

The temporal operator \bigcirc is called “next”, and U is called “until”. We write $\diamond \phi$ for $\text{true} U \phi$, and call it “eventually”. Operator $C^{\leq t}$ is “bounded cumulative reward”, expressing the reward accumulated over t steps. Formula $R_{\bowtie q}^r(\diamond \phi)$ expresses “reachability reward”, the reward accumulated up until the first time a state satisfying ϕ .

Given $D = (S, s_0, \mathbf{P}, L)$ and $r = (r_S, r_T)$, the satisfaction of state formula ϕ on a state $s \in S$ is defined as:

$$\begin{aligned} s \models \text{true}; \quad s \models ap &\Leftrightarrow ap \in L(s); \quad s \models \neg \phi \Leftrightarrow s \not\models \phi; \\ s \models \phi_1 \wedge \phi_2 &\Leftrightarrow s \models \phi_1 \text{ and } s \models \phi_2; \\ s \models P_{\bowtie p}(\psi) &\Leftrightarrow Pr(s \models \psi) \bowtie p; \\ s \models R_{\bowtie q}^r(\mu) &\Leftrightarrow \mathbb{E}[\text{rew}^r(\mu)] \bowtie q, \end{aligned}$$

where $Pr(s \models \psi) \bowtie p$ concerns the probability of the set of paths satisfying ψ starting in s . Given a path η , if write $\eta[i]$ for its i -th state and $\eta[0]$ the initial state, then

$$\begin{aligned} \text{rew}^r(C^{\leq t})(\eta) &= \sum_{j=0}^{t-1} (r_S(\eta[j]) + r_T(\eta[j], \eta[j+1])) \\ \text{rew}^r(\diamond \phi)(\eta) &= \begin{cases} \infty & \forall j \in \mathbb{N}(\eta[j] \not\models \phi) \\ \text{rew}^r(C^{\leq \text{ind}(\eta, \phi)})(\eta) & \text{otherwise} \end{cases} \end{aligned}$$

where $\text{ind}(\eta, \phi) = \min\{j \mid \eta[j] \models \phi\}$ denotes the index of the first occurrence of ϕ on path η . Moreover, the satisfaction relations for a path formula ψ on a path η is defined as:

$$\begin{aligned} \eta \models \bigcirc \phi &\Leftrightarrow \eta[1] \models \phi \\ \eta \models \phi_1 U \phi_2 &\Leftrightarrow \exists j \geq 0 (\eta[j] \models \phi_2 \wedge \forall k < j (\eta[k] \models \phi_1)) \end{aligned}$$

Very often, it is of interest to know the actual probability that a path formula is satisfied, rather than just whether or not the probability meets a required threshold since this can provide a notion of margins as well as benchmarks for comparisons following later updates. Subsequently, the PCTL definition can be extended to allow *numerical queries* of the form $\mathcal{P}_{=?}(\psi)$ or $\mathcal{R}_{=?}^r(\psi)$ [31]. After formalising the system behaviours and properties in DTMC and PCTL, respectively, automated tools have been developed to solve the verification problem, e.g., PRISM [34] and STORM [35].

IV. PROBLEM FORMULATION

A. Running Example

We consider a DRL-driven robot that navigates, and avoids collisions, in a complex environment where there are static and dynamic objects/obstacles. The model-free DDPG algorithm [29] is applied for the training of a DRL policy for the robot. Typically, the DRL policies are trained in a simulation environment before being applied to the real world [36]. That is because of the unbearable costs of having real-world (negative) examples for training in the real world [37].

As stated in Section III-A, the robot can be modelled as an MDP. At each time t , it has its observation of the laser sensors from the environment, namely state s_t , i.e.,

$$s_t = (o_t^1, o_t^2, \dots, o_t^n)^T \quad (1)$$

where $o_t^1, o_t^2, \dots, o_t^n$ are sensor signals at time t . As usual, the sensors can only scan the environment within a certain distance, for example, it is within 3.15 metres in Turtlebot Waffle Pi [38] for a distance sensor.

An action $a_t \in \mathcal{A}$ consists of several decision variables. With the Proportional-Integral-Derivative (PID) controller on the robot, we consider two action variables, representing line velocity and angle velocity, respectively, i.e., $a_t = (v_t^{\text{line}}, v_t^{\text{angle}})^T$. At each time t , the DRL actor network outputs an action pair $(v_t^{\text{line}}, v_t^{\text{angle}})$ from the action set \mathcal{A} .

The objective of the robot is to avoid obstacles and reach a goal area. On every state s_t , the sensory input o_t^i can be utilised to, e.g., predict the distance to the obstacles and the goal area when they are close enough (within 3.15 metres). The environment imposes a reward function r on both the states (w.r.t. the distance to obstacles) and the actions (w.r.t. the acceleration in linear or angular speed).

We leave out the details of training a DDPG agent, and only refer to the trained policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$.

Remark 1 (Risk). In different application contexts, “risky situations” may vary case by case and typically are defined based on safety analysis like hazard identification for the given application. While two examples are shown in Fig. 2, we define risk concerning the distance between the robot and the closest obstacle in this paper.

Remark 2 (Disturbance). In real-world RAS applications, the environments of the robots are subject to different levels of disturbances due to, e.g., different wind speeds, weather conditions, and ground surfaces [39], [40]. In this paper,

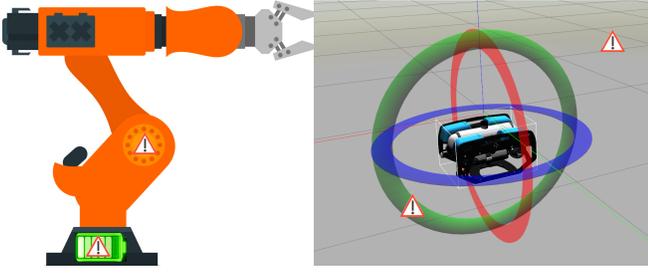


Fig. 2: Definitions on risk in different applications. LHS: mechanical failures and power supply shortage are risky situations for industrial robots. RHS: over-limited rotation angle and high liquid flow rates are risky situations for underwater vehicles.

the sensor noise is deemed to be the disturbances of the unmanned ground vehicle.

B. Construction of a DTMC Describing the Failure Process

We consider the execution of the policy π in an environment. For simplicity, we only differentiate the environments with a disturbance level that the robot’s sensory input may be subject to, and assume that the disturbance level follows a distribution $\mathcal{N}(0, \sigma)$. Now, as stated in Section III-A, given an MDP \mathcal{M}^σ (based on a disturbance $\mathcal{N}(0, \sigma)$) and a DRL policy π , there is a trajectory distribution $\rho^{\pi, \sigma}(\zeta)$. Based on the dynamics of risk levels in $\rho^{\pi, \sigma}(\zeta)$, we can define a DTMC, as shown in Fig. 3. It consists of a “negligible-risk” state s_N , a catastrophic failure state s_C , and several states s_{B_i} representing different levels of “benign failures”.

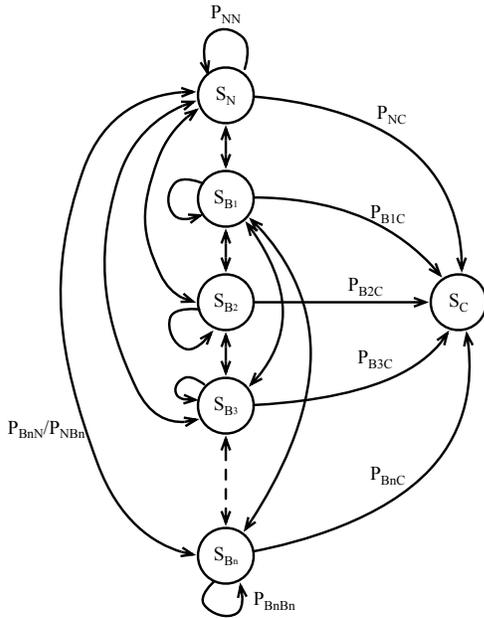


Fig. 3: The failure process DTMC based on risk levels.

Each trajectory is a sequence of successive states from the initial state to the end state of a DRL episode. First, we map each state in the trajectories to one of the states describing

the failure process (i.e., s_N , s_C , and s_{B_i}). Second, we may conduct statistical analysis on the frequency of transitions between s_N , s_C , and s_{B_i} , based on which we estimate their corresponding transition probabilities. Finally, we construct the failure process DTMC with the defined structure and the estimated transition probabilities. To be exact, we describe the 3 main steps above as what follows.

1) *Mapping MDP States onto DTMC States*: First of all, every state in the DTMC (cf. Fig. 3) is associated with a risk level. Specifically, s_N is the negligible-risk state, s_C is the catastrophic failure state, and s_{B_i} are benign failure states such that the risk on s_{B_i} is higher than on s_{B_j} if $i > j$.

Now, to map \mathcal{S} (the states on the trajectories) onto \mathcal{S} (the states on the DTMC), we define a measure of risk based on the distance of the robot to obstacles. For instance, s_N suggests that the robot is 3+ metres away from the obstacle, s_{B_1} suggests 2-3 metres away, s_{B_2} suggests 1-2 metres away, etc. Moreover, catastrophic failure s_C is defined as the robot terminated unexpectedly by a non-recoverable failure. The determination of the risk levels for states in \mathcal{S} can be done by evaluating the sensory input.

Definition 4 (Negligible-Risk Route). Given an MDP \mathcal{M}^σ and a DRL policy π , a negligible-risk route is defined as a mission trajectory in $\rho^{\pi, \sigma}(\zeta)$ that contains only s_N states.

We remark that, the negligible-risk route is not necessarily the optimal route achieving the highest reward, rather it only depends on the risk levels during the RAS mission.

2) *Estimating Transition Probabilities*: We can collect a set of mission trajectories by conducting statistical testing (the simple Monte Carlo sampling in our case) on $\rho^{\pi, \sigma}(\zeta)$. Then, all mission trajectories collectively can be transformed into a set of transitions, based on which we build a transition matrix to record the statistical data as follows:

	s_N	s_{B_1}	...	s_{B_m}	s_C
s_N	$n_{1,1}$	$n_{1,2}$...	$n_{1,m+1}$	$n_{1,m+2}$
s_{B_1}	$n_{2,1}$	$n_{2,2}$
...
s_{B_m}	$n_{m+1,1}$	$n_{m+1,m+1}$...
s_C	$n_{m+2,1}$	$n_{m+2,m+2}$

where $n_{1,1}$ records the number of transitions from s_N to s_N , and so on. m is the number of levels of benign failures (that varies case by case depending on the application-specific context, e.g., we choose $m = 3$ in our experiments).

Let the transition probability matrix of the failure process DTMC be $\mathbf{P}_1 = (p_{ij}) \in [0, 1]^{(m+2) \times (m+2)}$. In a DTMC, given a current state i , the transition to the next state follows a *categorical distribution*. Due to the Markov property, the categorical distributions of each state are *independent*. Hence, as we observe repeated outgoing transitions from state i , the repeated categorical process follows a *multinomial distribution*. For the i -th row of \mathbf{P}_1 , the likelihood function \mathcal{L} is (by omitting the combinatorial factor):

$$\mathcal{L}(p_{i,1}, \dots, p_{i,m+2} \mid n_{1,1}, \dots, n_{1,m+2}) = \prod_{j=1}^{m+2} p_{i,j}^{n_{i,j}} \quad (2)$$

Upon establishing the likelihood function, many existing estimators can be invoked for our purpose—from the basic Maximum Likelihood Estimation (MLE), Bayesian estimators [41] and estimator with (frequentist/Bayesian) bounds [18], [42]. While more advanced estimators can be easily integrated in our proposed framework, we only present the use of MLE in this paper for brevity:

$$\hat{p}_{i,j} = \frac{n_{i,j}}{\sum_{j=1}^{m+2} n_{i,j}} \quad (3)$$

It is known that MLE is an unbiased estimator [41], while the uncertainty in the estimates is captured by the variance that depends on the number of samples. The propositions in [42] provide the means for calculating $(1 - \alpha)$ confidence intervals of the verification results, given the observations on the frequencies between states (exactly as our statistical data $n_{i,j}$). Such result may in turn determine the required number of samples $n_{i,j}$ given a required say 95% confidence level for the final verification results. Although we did not calculate the confidence interval to determine the sample size in this paper, we instead chose a sample size in our later experiments that is sufficiently large to show a converging trend of the verification results (cf. Section VI-B).

3) *Construction of Failure Process DTMC*: The failure process DTMC is the product of two DTMCs, M_1 and M_2 , via the synchronisation of the transition actions.

Let $AP_1 = \{crash, neg_risk, risk_B_1, \dots, risk_B_n\}$, and we construct the first DTMC $M_1 = (S, s_N, \mathbf{P}_1, L_1)$ where $S = \{s_N, s_{B_1}, \dots, s_{B_n}, s_C\}$, $neg_risk \in L_1(s_N)$, $risk_B_i \in L_1(s_{B_i})$ for $i \in \{1..n\}$, and $crash \in L_1(s_C)$. Each entry $p_{i,j}$ of \mathbf{P}_1 is defined as Eqn. (3). We also define a reward structure “deviation” = (r_S, r_T) with $r_S(s_N) = 0$, $r_S(s_C) = 0$, $r_S(s_{B_i}) = d_i$ (where d_i is the deviation from s_N to s_{B_i}) and $r_T(s_1, s_2) = 0$ for all $s_1, s_2 \in S$.

Moreover, we need a “mission stage DTMC” (for simplicity, we only consider two stages—mission terminated or not). Let $AP = \{progressing, terminated\}$, we construct $M_2 = (K, k_0, \mathbf{P}_2, L_2)$ with $K = \{k_0, k_1\}$, $progressing \in L_2(k_0)$ and $terminated \in L_2(k_1)$. The transition probabilities \mathbf{P}_2 are $p_{k_0, k_1} = \frac{1}{l_{mis}}$, $p_{k_0, k_0} = 1 - \frac{1}{l_{mis}}$, $p_{k_1, k_1} = 1$ and $p_{k_1, k_0} = 0$, where l_{mis} is a constant representing the expected mission length (number of transitions) obtained from the testing data. We also define a reward structure for this DTMC: “step” = (r_S, r_T) with $r_T(k_0, k_0) = 1$, $r_T(k_0, k_1) = 1$ and $r_S(k) = 0$ for all $k \in K$.

Finally, we encode the failure process DTMC with PRISM model checker [34]. Although we only present how to leverage sampling to construct the DTMCs as an initial framework, other works have shown how to incorporate diverse evidence for which we have the following remark.

Remark 3 (Constructing DTMCs from Disparate Evidence). *In addition to statistical sampling, the DTMCs constructed in our method may contain knowledge sourced from disparate evidence, e.g., domain knowledge, V&V and traditional safety/hazard analysis (say from FTA/FMEA to DTMCs) [17], [18]. Such extra knowledge is formally incorporated*

via our failure process DTMC to accurately model the dynamics of risk in the executions of RAS.

V. FORMAL PROPERTIES

We define a set of quantitative dependability properties, and use PMC to check if they hold on the DTMC. In what follows, we go through each property with both informal description (referring to Fig. 1) and PCTL definition.

Before proceeding, we define

$$\begin{aligned} miss_comp &:= \neg crash \wedge terminated \\ crit_situ &:= risk_B_{max} \wedge progressing \\ ncrit_situ &:= neg_risk \wedge progressing \end{aligned} \quad (4)$$

where $miss_comp$ denotes a successful completion of the mission (i.e., *terminated* without *crash*), $crit_situ$ denotes the robot is in critical situation (i.e., at the most serious benign failure level $risk_B_{max}$ but still *progressing*), and $ncrit_situ$ denotes the robot is in a sufficiently safe situation (i.e., negligible risk and *progressing*).

a): Safety requires that “something bad will never happen”. Thus, we quantify the RAS safety as the probability of never reaching the catastrophic failure state (representing, e.g., crashes) before the mission is terminated normally.

Definition 5 (Safety). *The safety property $Prop_S$ measures the probability that the RAS, starting in the initial state², successfully completes the mission. Formally,*

$$Prop_S := P_{=?}[\diamond miss_comp] \quad (5)$$

b): Resilience, despite the existence of various definitions in the literature, is generally referred as the ability to respond to change and survive/prosper, e.g., the ability to deal with attacks or surprised disturbances [43]. For RAS, we define it as an ability of the DRL policy that can help the RAS recover from any deviations from the negligible-risk route.

Definition 6 (Resilience). *Given the reward structure “deviation” (cf. Section IV-B.3), resilience is defined as the expected total deviation from the negligible-risk route in a successful³ mission. Formally,*

$$Prop'_{Res} := R_{=?}^{“deviation”}[\diamond miss_comp] \quad (6)$$

To accord with the intuition, we normalise it by considering $max_dev := \max_i \{d_i\} \cdot \bar{R}_{=?}^{“step”}[\diamond miss_comp]$, the worst-case total deviation that could happen in a mission, which is the product of the largest deviation $\max_i \{d_i\}$ and the expected length of a mission that terminates safely, i.e.,

$$Prop_{Res} := 1 - \frac{Prop'_{Res}}{max_dev} \quad (7)$$

c): Robustness requires that the behaviour of the RAS is invariant against small disturbance on inputs. Formally,

²We assume the RAS always initialises in the negligible-risk state s_N .
³Presumably, there is no practical meaning to consider the resilience in a crashed mission.

Definition 7 (Robustness). *Given a disturbance level, the robustness $Prop_{Rob}$ quantifies the ability to resist the disturbance, i.e., recovering from the critical situation.*

$$Prop_{Rob} := \frac{P_{=?}[\diamond (crit_situ \wedge \diamond miss_comp)]}{P_{=?}[\diamond crit_situ]} \quad (8)$$

d): Detection intuitively refers to the ability of “realising” the existence of the disturbance and then starting to recover.

Definition 8 (Detection). *The detection property $Prop'_D$ is defined as the number of steps (i.e. transitions in the mission-stage DTMC) between the times when the disturbance is first applied⁴ and when the RAS first reaches the critical situation*

$$Prop'_D := R_{=?}^{“step”}[\diamond crit_situ] \quad (9)$$

where the reward structure “step” is defined in Section IV-B.3. Again, we normalise it by letting

$$Prop_D := 1 - \frac{Prop'_D}{R_{=?}^{“step”}[\diamond miss_comp]} \quad (10)$$

where the denominator is the expected length of a mission that terminated safely.

e): Recovery intuitively refers to the ability of “recovering” from a critical situation and eventually reaching the negligible-risk state, after the detection of the disturbance.

Definition 9 (Recovery). *The recovery property $Prop_{Rec}$ is defined as the number of steps between the times when the disturbance is first detected and when the RAS first reaches the s_N after the detection, i.e.,*

$$Prop'_{Rec} := R_{=?}^{“step”}[\diamond (crit_situ \wedge \diamond ncrit_situ)] - Prop'_D \quad (11)$$

It can be normalised in a similar way as in Definition 8 to get $Prop_{Rec}$, and we omit it for brevity.

Remark 4 (Generalisability). *The set of formally defined dependability properties is generic and can be applied to all RAS applications with problem specific definitions on the risk that maps onto failure process DTMC states.*

VI. EXPERIMENTAL RESULTS

This section presents experimental results regarding the following research questions:

RQ1: How sensitive are the verification results to the number of sampled trajectories (used in constructing the DTMC)?

RQ2: Does the DDPG training improve its dependability?

RQ3: How do the dependability properties react to different levels of disturbances?

In the following, we will first introduce our experimental environment, and then address the RQs individually⁵.

⁴For simplicity, we assume the disturbance level is applied at the very beginning of the mission and does not change thereafter.

⁵We omit the verification of the recovery property in the experiments, due to the limitation of state-of-the-art model checker (e.g., PRISM and STORM)—they cannot calculate rewards with nested LTL.

A. Experimental Environment

We have both simulation and physical environments for our experiments⁶. For the simulation environment, we use ROS [44] and Gazebo [45]. The training of DDPG algorithm is conducted on a maze space with obstacles.

For the physical environment, we assemble a Turtlebot3 Waffle Pi as the robot, together with a laboratory environment where a number of static and dynamic objects are randomly placed, as shown in Fig. 4. The Turtlebot3 is a compact and fully programmable mobile robot with a 360° LiDAR onboard. It is based on the standard ROS platform. The LiDAR information is taken as the input to the trained DRL policy. A Raspberry Pi and OpenCR are used to trigger two motors once received the output from the DRL policy.

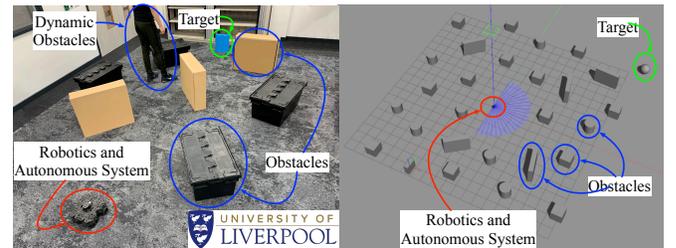


Fig. 4: Simulation of Turtlebot3 Waffle Pi

B. Dependability w.r.t. Sampled Trajectories

To answer **RQ1**, we evaluate the dependability properties against the number n of sampled trajectories. We consider the cases where the number n is gradually increased to 500. As given in Fig. 5a, the properties (safety, robustness, detection, and resilience) are all stable w.r.t. n in a typical experiment setting (a policy trained by some number of episodes and a certain disturbance level). We also repeat the experiment with other settings (used in later RQs), and the overall results suggest that our dependability analysis in the next two RQs are insensitive to the sample size especially when $n \geq 300$ which is the number of samples used in later experiments.

C. Dependability in Training

To answer **RQ2**, we apply the dependability analysis to monitor the changes of dependability properties during the training. We record the properties for the first 300 training episodes, as elucidated in Fig. 5b. During the training, the motion of the robot is terminated if and only if it crashes, reaches the goal, or gets stuck for 1000+ steps.

The values of the properties have significant fluctuations at the beginning of the training. This is due to the drastic weight update of DRL policies by the training process. As the number of training episodes increases, these properties gradually stabilise. Due to the existence of the exploration process, the properties retain a certain level of fluctuations, until reaching the final convergence (with minor oscillation).

⁶All source code, DRL models, datasets, PRISM files and experiment results are publicly available at our project website <https://github.com/YD-19/HAM4DRL.git>

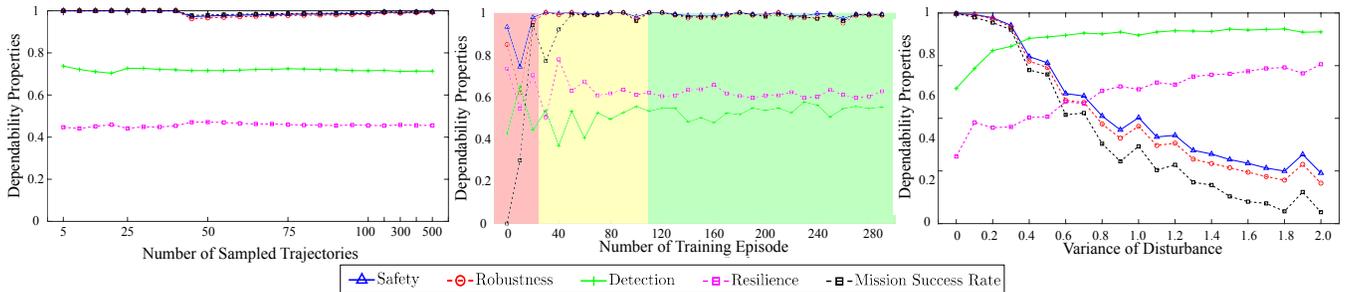


Fig. 5: Dependability analysis with (a.) different number of Samples, (b.) different training episodes, and (c.) different disturbance levels.

Importantly, the training process can be roughly split into three phases: *conservative phase* (red area in Fig. 5b), *optimising phase* (yellow area), and *optimised phase* (green area). Due to the high cost of crashes, the DRL policy explores the environment safely and carefully in the *conservative phase*, without reaching a goal. Therefore, the mission success rate at this stage is low, while the safety property remains high. Once the robot reaches its goal for the first time, it will receive a much higher reward, which in turn will influence all the state values in the path according to the training algorithm. In the *optimising phase*, the DRL algorithm optimises the policy through exploration. Failed missions still exist because the exploration may cause crashes. During this phase, the safety and robustness are kept high to ensure safe exploration, while the other properties remain fluctuating. In the *optimised phase*, the policies are already well-trained, and the properties are converged. The remaining small oscillation is due to environment disturbances.

Nevertheless, we notice that the training does not improve, only stabilise, properties such as resilience and detection. This is mainly because the training, as a stochastic optimisation process, does not incorporate optimisation objectives concerning these properties. This forms our future work.

D. Dependability w.r.t. Disturbance Levels

As suggested in Section IV-B, we use a half-Gaussian distribution $\mathcal{N}(0, \sigma)$ to model the disturbance level of the environment, and differentiate the environments with the parameter σ . This is without loss of generality, as we can extend this to more involved ways of modelling disturbances. In our experiments, σ ranges from 0.1 to 2.0, and we take a well-trained policy π for analysis.

As shown in Fig. 5c, we can compute the dependability properties of $\rho^{\pi, \sigma}$ by varying σ , so that the analysis is conducted over a set of environments with different levels of disturbances. We can see that, the RAS becomes less safe and robust (cf. the blue curve and the red curve, respectively) when the disturbance level increases. The trends are roughly aligned with the decrease of the mission success rate. Also, it is unsurprising that the detection becomes easier as the increases of disturbance level (cf. the green curve), because the robot may be easier, and faster, to reach a critical

situation. On the other hand, the resilience property shows a trend of getting better (cf. the purple curve). We believe this may be related to the fact that, with greater disturbances, there are more risky routes, 1) many of which may be easily recovered and thus lead to the improvement to the resilience property; 2) most routes with worse resilience are crashed and thus excluded from the calculation by our definition of resilience.

Notably, we observe two trends in Fig. 5c—curves of detection and resilience increase while the curves of other properties decrease. This reveals the conflicts between different properties, which may require trade-offs in training.

VII. CONCLUSION

A dependability analysis framework is proposed to evaluate a set of quantitative properties of a given DRL policy, such as safety, resilience, robustness, detection and recovery. Experimental results show the effectiveness of the framework in assessing DRL-driven RAS holistically.

ACKNOWLEDGMENT & DISCLAIMER

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 956123. This work is supported by the UK Dstl (through the project of Safety Argument for Learning-enabled Autonomous Underwater robots) and the UK EPSRC (through the Offshore Robotics for Certification of Assets [EP/R026173/1] and End-to-End Conceptual Guarding of Neural Architectures [EP/T026995/1]). XZ’s contribution is partially supported through Fellowships at the Assuring Autonomy International Programme.

REFERENCES

- [1] D. Lane, D. Bisset, R. Buckingham, G. Pegman, and T. Prescott, “New foresight review on robotics and autonomous systems,” Lloyd’s Register Foundation, London, U.K., Tech. Rep. No. 2016.1, 2016.
- [2] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Tran. on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.
- [3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [4] Y. Dong, Z. Dong, T. Zhao, and Z. Ding, “A strategic day-ahead bidding strategy and operation for battery energy storage system by reinforcement learning,” *Electric Power Systems Research*, vol. 196, p. 107229, 2021.

- [5] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.
- [6] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI Conf. on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 3387–3395.
- [7] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, "Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 14 413–14 423, 2020.
- [8] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. A. Ojea, E. Solowjow, and S. Levine, "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5548–5555.
- [9] J. Li, T. Chai, F. L. Lewis, Z. Ding, and Y. Jiang, "Off-policy interleaved q -learning: Optimal control for affine nonlinear discrete-time systems," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 5, pp. 1308–1320, 2018.
- [10] V. Behzadan and A. Munir, "Adversarial reinforcement learning framework for benchmarking collision avoidance mechanisms in autonomous vehicles," *IEEE Intelligent Transportation Systems Magazine*, vol. 13, no. 2, pp. 236–241, 2021.
- [11] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proc. of the 32nd AAAI Conf. on Artificial Intelligence*, 2018.
- [12] N. Jansen, B. Könighofer, S. Junges, and R. Bloem, "Shielded decision-making in mdps," *arXiv preprint arXiv:1807.06096*, 2018.
- [13] H. Zhang, H. Chen, C. Xiao, B. Li, M. Liu, D. Boning, and C.-J. Hsieh, "Robust deep reinforcement learning against adversarial perturbations on state observations," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 21 024–21 037.
- [14] R. Bloomfield, G. Fletcher, H. Khlaaf, P. Ryan, S. Kinoshita, Y. Kinoshita, M. Takeyama, Y. Matsubara, P. Popov, K. Imai *et al.*, "Towards identifying and closing gaps in assurance of autonomous road vehicles—a collection of technical notes part 1," *arXiv preprint arXiv:2003.00789*, 2020.
- [15] V. Robu, D. Flynn, and D. Lane, "Train robots to self-certify as safe," *Nature*, vol. 553, no. 7688, pp. 281–281, 2018.
- [16] J. García, Fern, and o Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 42, pp. 1437–1480, 2015.
- [17] R. Calinescu, C. Paterson, and K. Johnson, "Efficient Parametric Model Checking Using Domain Knowledge," *IEEE Transactions on Software Engineering*, vol. 47, no. 6, pp. 1114–1133, 2021.
- [18] X. Zhao, V. Robu, D. Flynn, F. Dinmohammadi, M. Fisher, and M. Webster, "Probabilistic model checking of robots deployed in extreme environments," in *Proc. of the AAAI Conf. on Artificial Intelligence*, vol. 33, Honolulu, Hawaii, USA, 2019, pp. 8076–8084.
- [19] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," *arXiv preprint arXiv:1705.08551*, 2017.
- [20] S. Huh and I. Yang, "Safe reinforcement learning for probabilistic reachability and safety specifications: A lyapunov-based approach," *arXiv preprint arXiv:2002.10126*, 2020.
- [21] A. Mandelkar, Y. Zhu, A. Garg, L. Fei-Fei, and S. Savarese, "Adversarially robust policy learning: Active construction of physically-plausible perturbations," in *2017 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3932–3939.
- [22] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," in *ICLR '18*, 2018.
- [23] K. Kurach, A. Raichuk, P. Stańczyk, M. Zajac, O. Bachem, L. Espeholt, C. Riquelme, D. Vincent, M. Michalski, O. Bousquet *et al.*, "Google research football: A novel reinforcement learning environment," in *Proceedings of the AAAI Conf. on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 4501–4510.
- [24] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Re-luplex: An efficient SMT solver for verifying deep neural networks," in *CAV*, 2017, pp. 97–117.
- [25] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," in *IJCAI*, 2018, pp. 2651–2659.
- [26] N. Fulton, "Verifiably safe autonomy for cyber-physical systems," Ph.D. dissertation, Ph.D thesis, Computer Science Department, Carnegie Mellon University, 2018.
- [27] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," in *Proc. of the 32nd Int. Conf. on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 2499–2509.
- [28] H. Zhu, Z. Xiong, S. Magill, and S. Jagannathan, "An inductive synthesis framework for verifiable reinforcement learning," in *Proceedings of the 40th ACM SIGPLAN Conf. on Programming Language Design and Implementation*, 2019, pp. 686–701.
- [29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *ICLR'16*, 2016.
- [30] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [31] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic Model Checking: Advances and Applications," in *Formal System Verification: State-of-the-Art and Future Trends*, R. Drechsler, Ed. Cham: Springer, 2018, pp. 73–121.
- [32] X. Zhao, M. Osborne, J. Lantair, V. Robu, D. Flynn, X. Huang, M. Fisher, F. Papacchini, and A. Ferrando, "Towards integrating formal verification of autonomous robots with battery prognostics and health management," in *Software Engineering and Formal Methods*, ser. LNCS, P. C. Ölveczky and G. Salaün, Eds., vol. 11724. Cham: Springer, 2019, pp. 105–124.
- [33] S. Gerasimou, R. Calinescu, S. Shevtsov, and D. Weyns, "UNDERSEA: an exemplar for engineering self-adaptive unmanned underwater vehicles," in *IEEE/ACM 12th Int. Symp. on Software Engineering for Adaptive and Self-Managing Systems*, Buenos Aires, Argentina, May 2017, pp. 83–89.
- [34] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Computer Aided Verification*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 585–591.
- [35] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A Storm is coming: A modern probabilistic model checker," in *Computer Aided Verification*, ser. LNCS, R. Majumdar and V. Kunčák, Eds., vol. 10427. Cham: Springer, 2017, pp. 592–600.
- [36] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv preprint arXiv:1610.03518*, 2016.
- [37] Y. Yu, "Towards sample efficient reinforcement learning," in *IJCAI*, 2018, pp. 5739–5743.
- [38] Robotis, "Robotis(2019) turtlebot3 – e-manual, waffle pi," [Online] <https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>. (Accessed on 02 August 2021).
- [39] Y. Yang, J. Zhu, X. Zhang, and X. Wang, "Active disturbance rejection control of a flying-wing tailsitter in hover flight," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6390–6396.
- [40] H. Seo, D. Lee, C. Y. Son, C. J. Tomlin, and H. J. Kim, "Robust trajectory planning for a multirotor against disturbance based on hamilton-jacobi reachability analysis," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 3150–3157.
- [41] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *Proc. of the 31st Int. Conf. on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 111–121.
- [42] R. Calinescu, C. Ghezzi, K. Johnson, M. Pezzé, Y. Rafiq, and G. Tamburrelli, "Formal verification with confidence intervals to establish quality of service properties of software systems," *IEEE Transactions on Reliability*, vol. 65, no. 1, pp. 107–125, 2016.
- [43] D. D. Woods, *Resilience engineering: Concepts and precepts*. CRC Press, 2017.
- [44] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.
- [45] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 3. IEEE, 2004, pp. 2149–2154.