CARE: Finding Root Causes of Configuration Issues in Highly-Configurable Robots

Md Abir Hossen[1], Sonam Kharade[1], Bradley Schmerl[2], Javier Cámara[3], Jason M. O'Kane[4], Ellen C. Czaplinski[5], Katherine A. Dzurilla[5], David Garlan[2] and Pooyan Jamshidi[1]

***Abstract*—Robotic systems have subsystems with a combinatorially large configuration space and hundreds or thousands of possible software and hardware configuration options interacting non-trivially. The configurable parameters are set to target specific objectives, but they can cause functional faults when incorrectly configured. Finding the root cause of such faults is challenging due to the exponentially large configuration space and the dependencies between the robot's configuration settings and performance. This paper proposes CARE—a method for diagnosing the root cause of functional faults through the lens of causality. CARE abstracts the causal relationships between various configuration options and the robot's performance objectives by learning a causal structure and estimating the causal effects of options on robot performance indicators. We demonstrate CARE's efficacy by finding the root cause of the observed functional faults and validating the diagnosed root cause by conducting experiments in both physical robots (*Husky* and *Turtlebot 3*) and in simulation (*Gazebo*). Furthermore, we demonstrate that the causal models learned from robots in simulation (e.g., *Husky* in *Gazebo*) are transferable to physical robots across different platforms (e.g., *Husky* and *Turtlebot 3*).**

***Index Terms*—robotics and autonomous systems, causal inference, robotics testing**

I. INTRODUCTION

ROBOTIC systems are highly configurable, typically composed of multiple subsystems (e.g., localization, navigation), each of which has numerous configurable components (e.g., selecting path planning algorithms in the planner). Once an algorithm has been selected for a component, its associated parameters must be set to the appropriate values (e.g., `use grid path = True`). The configuration space in such robotic systems is combinatorially large, with hundreds if not thousands of software and hardware configuration choices that interact non-trivially with one another. Indeed, incorrectly specified configuration options are one of the most common causes of system failure [1]. The configuration space in robotic systems directly impacts mission objectives (e.g., navigating from one place to another), enabling trade-offs in the objective space (e.g., the time that it takes to reach the target location(s) vs. the energy consumption for the task). The magnitude of the trade-off (even for the same configuration option) is dictated by the characteristics of the operating environment (e.g., the roughness of the surface). Unfortunately, configuring robotic systems to meet specified requirements is challenging and error-prone [2]. Incorrect configuration (called *misconfiguration*) can cause buggy behavior, resulting in *functional* and/or *non-functional faults*.[1] Misconfigured parameters specified during design time can cause unexpected behavior at run time [3]. In addition, the operating environment may change during a mission [4], [5] and may require changing the configuration values on the fly [6]. The aforementioned challenges make debugging robots a difficult task.

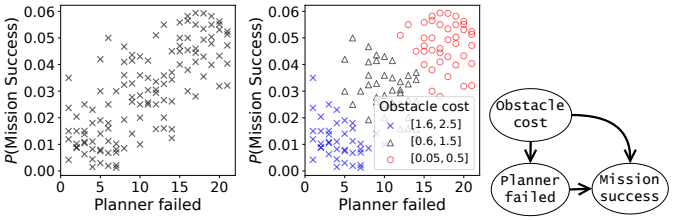


Fig. 1: An example showing the effectiveness of causality. (a) Incorrect reasoning (b) correct correlation after incorporating `obstacle cost` as a confounder; (c) the causal model correctly captures `obstacle cost` as a common cause to explain the robot's behavior.

To handle the challenges in performance debugging and analysis, performance influence models [7]–[9] have received significant attention. Such models predict the performance behavior of systems by capturing the important options and interactions that influence the performance behavior. However, performance influence models built using predictive methods suffer from several shortcomings, including (i) failing to capture changes in the performance distribution when deployed in unexpected environments [10], (ii) producing incorrect explanations as illustrated in Fig. 1b, (iii) lack of transferability among common hardware platforms that use the same software stack [11], and (iv) collecting the training data for predictive models from physical hardware is expensive and requires constant human supervision [12]. Traditional statistical debug-

Manuscript received: January, 16, 2023; Revised April, 15, 2023; Accepted April, 29, 2023. This paper was recommended for publication by Editor Aleksandra Faust upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by National Aeronautics and Space Administration (Award 80NSSC20K1720) and National Science Foundation (Award 2107463).

[1]M. A. Hossen, S. Kharade, and P. Jamshidi are with College of Engineering and Computing, University of South Carolina, SC, USA (e-mail: abir.hossen786@gmail.com; skharade@mailbox.sc.edu; pjamshid@cse.sc.edu)

[2]B. Schmerl, and D. Garlan are with School of Computer Science, Carnegie Mellon University, PA, USA (e-mail: [schmerl; garlan]@cs.cmu.edu)

[3]J. Cámara is with ITIS Software, Universidad de Málaga, Málaga, Spain (e-mail: jcamara@uma.es)

[4]J. M. O'Kane is with Department of Computer Science and Engineering, Texas A&M University, TX, USA (e-mail: jokane@tamu.edu)

[5]E. C. Czaplinski, and K. A. Dzurilla are with Jet Propulsion Laboratory, California Institute of Technology, CA, USA (e-mail: [Ellen.C.Czaplinski; Katherine.A.Dzurilla]@jpl.nasa.gov)

*Code and data are available at https://github.com/softsys4ai/care.

Digital Object Identifier (DOI): see top of this

[1]We define *functional faults* as failures to accomplish the mission objective (e.g., the robot could not reach the target location(s) specified in the mission specification). The *non-functional faults* (interchangeably used as *performance faults*) refer to severe performance degradation (e.g., the robot reached the target location(s); however, it consumed more energy, or it took more time than the specified performance goal in the mission specification).

ging techniques [13] based on correlational predicates, such as Cooperative Bug Isolation (CBI), can be used to debug system faults. However, statistical debugging is hindered by the need for large-scale data and the inherent difficulty of pattern recognition in high-dimensional spaces [14], which can be challenging for robotic systems with non-linear interactions between variables.

To address this problem, we present an approach called CARE (Causal Robotics DEbugging) to diagnose the root causes of functional faults caused by misconfigurations in highly-configurable robotic systems through the lens of causality. Causal models enable interventional and counterfactual analyses [15] and can accommodate for unobserved confounders [16]. These factors are important because certain variables that cannot be modified or have not been directly observed may exist, avoiding spurious correlations [17]. Debugging using causal models can also help with designing robust policies [18] that can adapt to different environments by identifying causal relationships between variables and testing policy effectiveness. These advantages make using a causal model more effective than traditional statistical debugging (e.g., CBI).

CARE works in three phases: In Phase I, we first learn a causal model from observational data—dynamic traces measuring the performance objectives (e.g., energy, mission success, etc.) while the robot performs a mission under different configuration settings. The causal model captures the causal relationships between configuration options and the robot's performance objectives. In Phase II, we use the causal graph to identify the *causal paths*—paths that lead from configuration options to a performance objective. Next, in Phase III, we determine the configuration options with the highest causal effect on a performance objective by measuring each path's average causal effect to diagnose the functional faults' root causes. Our numerical studies confirm that CARE obtains 87% accuracy, 83% precision, and 81% recall on the target platform (*Turtlebot 3*) when reusing the causal model constructed from a source platform (*Husky* in simulation). Moreover, CARE achieved 27% more accuracy and 24% more F1-score compared to CBI. Our contributions are as follows:

- We propose CARE (§III), a novel framework for finding the root causes of the configuration bugs in robotic systems.
- We evaluate CARE, conducting a comprehensive empirical study (§IV) in a controlled environment across multiple robotic platforms, including *Husky* and *Turtlebot 3* both in simulation and physical robots.
- We demonstrate the transferability of the causal models by learning the causal model in the *Husky* simulator and reusing it in the *Turtlebot 3* physical platform (§IV-C).

II. PROBLEM DESCRIPTION

A. Motivating scenarios

To motivate the approach, we use the DARPA Subterranean Challenge [19] to illustrate the following scenarios. This setting requires autonomous ground robots to work in adverse environments such as fog, debris, dripping water, or mud and to navigate sloped, declining, and confined passageways. In this case, the mission objective is to stop the robot perpendicular to the position of a particular artifact and transmit its location to the control station.

a) Functional fault due to configuration bug: Fig. 2a shows a scenario where the robot stops 0.5 m away from the target location and transmits incorrect artifact locations to the control station. A cause for this fault might be a delay in data transformation. For instance, the sensor transmits data at 1 Hz, and the robot travels at 0.5 m/s. As a result, when the `costmap` (which stores and updates information about obstacles in the environment using sensor data) receives data from the sensor, it is a second old, and the robot has already traveled 0.5 m away from that position.

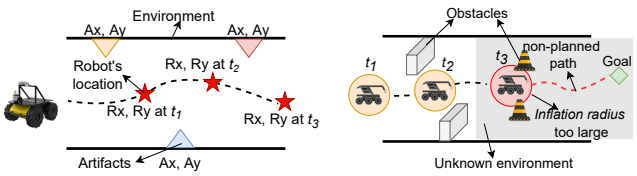
b) Functional fault due to change in environment: Extending the previous scenario, suppose the obstacle locations are unknown to the robot. Fig. 2b shows a scenario where at t_3 the robot encounters unique obstacles that are too close together, violating the `inflation radius` (which specifies the object's maximum sensing distance), defined before deployment, resulting in an indecisive robot that is stuck in place.

c) Incorrect reasoning about the robot's behavior: We perform a simple experiment for robot navigation, recording the number of failures in path planning (`planner failed`) and probability of `mission success`. Fig. 1a shows the distribution of the P(`mission success`) with respect to `planner failed`. We observe that an increase in `planner failed` leads to a higher P(`mission success`), which is counter-intuitive. Such a trend is typically captured by statistical reasoning in ML models. Incorporating `obstacle cost` along the trajectory as a confounder (Fig. 1b) correctly shows an increase in `planner failed` corresponding to a decrease in the P(`mission success`) (negative correlation). The causal model (Fig. 1c) correctly captures `obstacle cost` as a common cause to explain the correct relation between the `planner failed` and P(`mission success`). The arrows denote the assumed direction of causation, whereas the absence of an arrow shows the absence of direct causal influence between variables.

d) Challenges: A typical debugging approach to finding the root causes of such functional faults might be trial-and-error. However, this process requires non-trivial human effort due to the large configuration space. Even after finding the optimal fix (e.g., a new value for a configuration option), the new fix is not guaranteed to function in different environments (as in Fig. 2b). Another typical performance debugging strategy is building performance influence models, such as regression models. However, performance influence models are unable to capture changes in the performance distribution when deployed in an unseen environment (*non-transferable*) and produce incorrect explanations, as illustrated in Fig. 1.

B. Causal reasoning for robotics

We formulate the problem of finding root causes for functional faults in robotic systems using an abstraction of a causal model utilizing *Directed Acyclic Graphs* (DAGs) [17]. The causal model encodes performance variables, functional nodes (which define functional dependencies between performance variables, such as how variations in one or multiple variables determine variations in other variables), causal links that interconnect performance nodes with each other via functional nodes, and constraints to define assumptions we require in performance modeling (e.g., the configuration options cannot be the child node of performance objectives). Given a robotic system that intermittently encounters functional faults,



(a) Configuration bug (b) Change in environment

Fig. 2: Different functional faults. (a) Delay in data transformation results in a functional fault where the robot stops 0.5 m away from the target location and transmits incorrect artifact locations; (b) Change in environment results in an indecisive robot stuck in place. Circles surrounding the robot represent the `inflation radius`.

we aim to find the root causes of such faults by querying a causal model learned from observational data. We start by formalizing the problem of finding the causal directions from configuration options to performance objectives that indicate a functional fault. This problem can be subdivided into two parts: (a) learning—discovery of the causal relationship between nodes, and (b) inference—identification of the root causes for a functional fault using the learned causal model. Consider a configurable robotic system $\mathcal{A}$ which has a set of manipulable (or configurable) variables $\mathcal{X}$ that can be intervened upon, a set of non-manipulable variables $\mathcal{S}$ (non-functional properties of the system such as metrics that evaluate the performance) that can not be intervened, and a set of performance objectives $\mathcal{Y}$. We define the causal graph discovery problem formally:

Problem 2.1 (*Learning*). Given the observational data D, recover the causal graph $\mathcal{C}_\mathcal{G}$ that encodes the dependency structure between $\mathcal{X}, \mathcal{S}$, and $\mathcal{Y}$ of $\mathcal{V}$ such that the following structural constraints are satisfied:

$$v_i \nrightarrow v_j \;\; \forall v_i \in \mathcal{X} \subset \mathcal{V}, \;\; \forall v_j \in \mathcal{Y} \subset \{\mathcal{V} \setminus \mathcal{X} \setminus \mathcal{S}\}$$

The second part of the problem is to find the root cause of functional fault using the learned causal model. We formulate the inference problem to estimate the average causal effect of the configuration option on the performance objectives as:

Problem 2.2 (*Inference*). Given the causal graph $\mathcal{C}_\mathcal{G}$, determine the configuration option in $\mathcal{X}$, which is the root cause for the observed functional fault characterized by performance objectives $\mathcal{Y}$ as follows:

$$\{v_i^*\} = \arg\max_{v_i} ACE(v_i, v_j^*),$$

where $\{v_i^*\} \subset \mathcal{X}$ is the set of root causes (configuration options), $\{v_j^*\} \subset \mathcal{Y}$ are the performance objectives characterizing the functional fault, and ACE represents the average causal effect—the average difference between potential outcomes under different treatments [15].

III. CARE: CAUSAL ROBOTICS DEBUGGING

We propose a novel approach, called CARE, to find and reason about the intricate relations between configuration options and their effect on the performance objectives in highly configurable robotic systems. CARE works in three phases: (i) The observational data is generated by measuring the performance metrics and performance objectives under different configuration settings (see ① in Fig. 3) to construct the graphical causal model (see ③ in Fig. 3) enforcing the structural constraints (see ② in Fig. 3). (ii) The causal model is used to determine the paths that lead from configuration options to the performance objectives (see ④ in Fig. 3). (iii) The

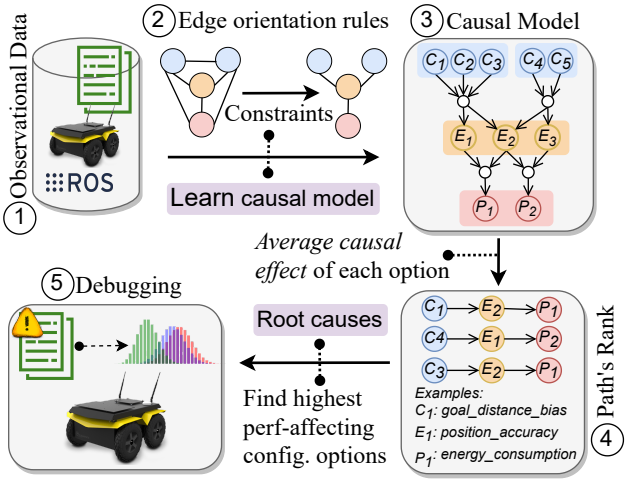


Fig. 3: Overview of CARE.

configuration options that have the highest causal effect on the performance objective were determined, measuring the average causal effect of each path to diagnose the functional faults (see ⑤ in Fig. 3).

A. Learning the causal model

We design a three-layer structure causal model defining three variable types: (i) software-level configuration options associated with different algorithms (e.g., `goal distance bias` [20]), and hardware-level options (e.g., `sensor frequency`), (ii) intermediate performance variables (non-manipulable variables) that map the influence of the configuration options to the performance objectives (e.g., `position accuracy`), and (iii) end-to-end performance objectives (e.g., energy). We classify the performance variables as non-manipulable and manipulable variables to reduce the number of variables that require intervention. Note that the level of debugging can vary [11], and the abstraction level of the variables in the causal model depends on the debugger and can go all the way down, even to the hardware level [21]. To build the three-layer structure, we define two specific constraints over causal models: (i) variables that can be manipulated (e.g., using prior experience, the user may want to restrict the variables that do not have a significant impact on performance objectives); (ii) structural constraints (e.g., configuration options do not cause other options). Such constraints enable incorporating domain knowledge that facilitates learning with low sample sizes. Several methods are proposed to extract the causal graphical model from data in the literature. These belong to two categories: constraint-based techniques and score-based techniques. We specifically use *Fast Causal Inference* (hereafter, FCI) [16], a constraint-based technique for identifying the causal model guiding robot performance. We select FCI because it identifies the unobserved confounders (common latent causes that have not been, or cannot be, measured); it can handle various data types (e.g., nominal, ordinal, and categorical) given a valid conditional independence test. When the FCI algorithm is applied to observational data, a *Partial Ancestral Graph* (PAG) [17], representing a causal structure in the presence of latent variables, is produced. Each edge in the PAG denotes the ancestral connections between the vertices. For a comprehensive theoretical foundation, we refer the reader to [22], [23]. To discover the true causal relationship between two variables, the causal graph must be

Algorithm 1: CM$(\text{data}, \mathcal{V}, \mathcal{G})$

Input: data, dense graph $\mathcal{G}$, Vertex set $\mathcal{V}$
Output: Set of $\mathcal{D}$ and $\mathcal{B}$ to build the ADMG

1 $\mathcal{D} \leftarrow \varnothing$, $\mathcal{B} \leftarrow \varnothing$
2 **while** $\mathcal{V} \in \mathcal{G}$ **do**
3 $S_c \leftarrow$ Apply structural constraints on $\mathcal{G}$
4 $\mathcal{G}_\partial$ = FCI($data$, Fisher z-test, S_c)
5 **for each** $\partial \in \mathcal{G}_\partial$ **do**
6 Compute entropies $H(v_i), H(v_j), H(\mathcal{Z})$
7 $\theta_r = 0.8 \min\{H(v_i), H(v_j)\}$
8 **if** $H(\mathcal{Z}) < \theta_r$ **then**
9 Replace $v_i \circ\!\!\rightarrow v_j$ with $v_i \leftrightarrow v_j$ in $\mathcal{B}$
10 **else**
11 $v_j = f(v_i, E)$ where $[E \perp v_i]$
12 $v_i = g(v_j, \widehat{E})$ where $[\widehat{E} \perp v_j]$
13 Compute the entropies $H(E)$ and $H(\widehat{E})$
14 **if** $H(E) < H(\widehat{E})$ **then**
15 Replace $v_i \circ\!\!-\!\!\circ v_j$ with $v_i \longrightarrow v_j$ in $\mathcal{D}$
16 **else**
17 Replace $v_i \circ\!\!-\!\!\circ v_j$ with $v_j \longrightarrow v_i$ in $\mathcal{D}$
18 **return** $\mathcal{D}, \mathcal{B}$

fully resolved [21] such that there are no $v_i \circ\!\!\rightarrow v_j$ (v_i causes v_j, or there are unmeasured confounders that cause both v_i and v_j), and $v_i \circ\!\!-\!\!\circ v_j$ (v_i causes v_j, or v_j causes v_i, or there are unmeasured confounders that cause both v_i and v_j) edges. We define the partial edge resolving problem formally as follows:

Problem 3.1 (*Resolve Partially Directed Edges*). Given a causal partial ancestral graph [17] $\mathcal{G}_\partial = (\mathcal{V}, \partial)$, the partial edge resolving problem involves replacing each partial edge ∂ with a directed edge $\mathcal{D}$ or a bi-directed edge $\mathcal{B}$ based on some threshold θ.

We use Algorithm 1 for learning the causal model (CM). First, we build a dense graph $\mathcal{G}$ by connecting all pairs of configuration options, performance metrics, and performance objectives with an undirected edge. Unlike configuration options, the intermediate layer's variables can not be modified. However, they can be observed and measured to understand how the causal effect of changing configurations propagates to a performance objective. The skeleton of the causal model is recovered by enforcing the structural constraints (e.g., no connections between configuration options, as in line 3 of Algorithm 1). Next, we evaluate the independence of all pairs of variables conditioned on all remaining variables using Fisher's exact test [24]. A PAG is generated, orienting the undirected edges by employing the edge orientation rules [17] (line 4 of Algorithm 1). The obtained PAG must be fully resolved (no ∂ between two vertices) to discover the true causal relationships. We resolve the FCI-generated PAG by evaluating if an unmeasured confounder $(\mathcal{Z})$ is present between two partially oriented nodes (v_i, v_j). Employing the information-theoretic approach based on entropy [25] produces a joint distribution $q(v_i, v_j, \mathcal{Z})$. We compute the entropy $H(\mathcal{Z})$ of $\mathcal{Z}$. Comparing the $H(\mathcal{Z})$ with θ_r (entropy threshold, $\theta_r = 0.8 \min\{H(v_i), H(v_j)\}$), we determine $\forall \mathbb{P}$ if $\exists \mathcal{Z} \in \partial$, as shown in lines 6-17 of Algorithm 1, where E and $\widehat{E}$ are the extrinsic variables responsible for system noise ($v_i \perp E$, $v_j \perp \widehat{E}$). The final causal model is an *Acyclic Directed Mixed Graph* (ADMG) [26].

Algorithm 2: CPWE$(\mathcal{V}, \mathcal{D}, \mathcal{B})$

Input: data, $\mathcal{V}$, $\mathcal{D}$, $\mathcal{B}$
Output: Rank of the causal paths

1 $\mathbb{P} \leftarrow \varnothing$, $K \leftarrow \varnothing$
2 ADMG $\leftarrow \{\mathcal{V}, \mathcal{D}, \mathcal{B}\}$
3 **while** $\mathcal{V}[\mathcal{Y}] \in$ *ADMG* **do**
4 $\mathbb{P} \leftarrow$ All causal paths from an $\mathcal{V}[\mathcal{Y}]$ node
5 **for** $i \leftarrow P_1$ **to** P_n **do**
6 Compute $\mathbb{P}_{ACE}$ using Equation 2
7 $K \leftarrow$ SORTDESCENDING$(\mathbb{P}_{ACE})$
8 **return** K

B. Causal effect estimation

To determine the root cause of a functional fault from the causal graph, we need to extract the paths (referred to as *causal paths*) from $\mathcal{C}_\mathcal{G}$. A causal path is a directed path originating from $\mathcal{X}$ (e.g., configuration options) to a subsequent non-functional property $\mathcal{S}$ (e.g., performance metrics) and terminating at $\mathcal{Y}$ (e.g., performance objectives). Our goal is to find an ordered subset of $\mathbb{P}$ that defines the causal path from the root cause of the functional fault (a manipulable variable that causes the functional fault) to the performance objective indicating the functional fault (say x_i causes a functional fault F through a subsequent node s_i in the path, assuming $(\exists x_i \in \mathcal{X}) \wedge (\exists s_i \in \mathcal{S})$; e.g., $x_i \longrightarrow s_i \longrightarrow \mathcal{Y}_F$). We define the causal path discovery problem as follows:

Problem 3.2 (*Causal Path Discovery*). Given a causal graph $C_\mathcal{G} = (\mathcal{V}, \mathcal{D}, \mathcal{B})$ that encodes the dependency structure between $\mathcal{X}, \mathcal{S}$ and $\mathcal{Y}$, and a performance objective $\mathcal{Y}_F \in \mathcal{V}$ indicating a specific functional fault, the causal path discovery problem seeks a path $\mathbb{P} = \langle v_0, v_1, \ldots, v_n \rangle$ such that the following conditions hold:

- v_o is the root cause of the functional fault and $v_n = \mathcal{Y}_F$.
- $\forall\ 0 \le i \le n,\ v_i \in \mathcal{V}$ and $\forall\ 0 \le i \le n,\ (v_i, v_{i+1}) \in (\mathcal{D} \vee \mathcal{B})$.
- $\forall\ 0 \le i \le j \le n$, v_i is a counterfactual cause of v_j.
- $|\mathbb{P}|$ is maximized.

We extract the causal paths and measure the average causal effect of the extracted causal paths on the performance objectives $(\mathcal{Y})$, and rank the paths from highest to lowest using Algorithm 2: Causal Paths With Effect (CPWE). CPWE simplifies the complicated causal graph using path extraction and ranking to a few useful causal paths to determine the configurations that most influence the performance objectives. Causal paths are discovered by backtracking from the nodes corresponding to each performance objective until we reach a node with no parents. The discovered paths are then ranked by measuring the causal effect of a node's value change (say V_1) on its subsequent node V_2 in the path. We express this using the *do-calculus* [15] notation: $\mathbb{E}[V_2 \mid \text{do}(V_1 = x)]$ that represents the expected value of V_2 if we set the value of node V_1 to x. The *average causal effect* (ACE) of $V_1 \longrightarrow V_2$ is calculated across all acceptable V_1 values as follows:

$$\text{ACE}(V_2, V_1) = \frac{1}{N} \sum_{\forall x, y \in V_1} \mathbb{E}[V_2 \mid \text{do}(V_1 = y)] \ - \ \mathbb{E}[V_2 \mid \text{do}(V_1 = x)], \quad (1)$$

where N is the total number of acceptable values of V_1. $\text{ACE}(V_2, V_1)$ will be larger if V_1 yields a larger change in V_2. We calculate the ACE for the entire causal path extending Equation 1 as follows:

$$\mathbb{P}_{ACE} = \frac{1}{K} \sum \text{ACE}(v_j, v_i) \quad (2)$$

The configuration options found on paths with larger $\mathbb{P}_{ACE}$ are likely to have a higher causal effect on the corresponding performance objective. The top K paths with the largest $\mathbb{P}_{ACE}$ values were selected for each performance objective.

IV. Experiments and Results

Using the *Husky* and *Turtlebot 3* platforms as case study systems, we answer the following research questions (RQ):

- RQ1 (Accuracy): To what extent are the root causes determined by CARE the true root causes of the observed functional faults?
- RQ2 (Transferability): To what extent can CARE accurately detect misconfigurations when deployed in a different platform?

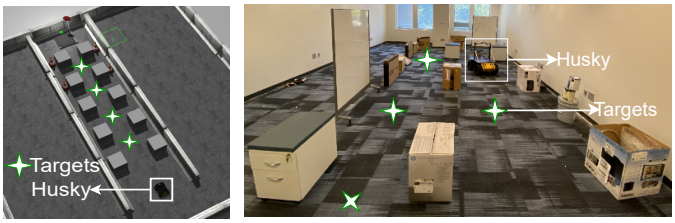
A. Experimental setup

We simulate *Husky* in *Gazebo* to collect the observational data by measuring the performance metrics (e.g., traveled distance) and performance objectives (e.g., energy consumption) under different configuration settings to train the causal model. Note that we use simulator data to evaluate the transferability of the causal model to the physical robots, but CARE also works with data from physical robots. We deployed the robot in a controlled indoor environment and directed the robot to autonomously navigate to the five target locations (Fig. 4). The robot was expected to encounter obstacles and narrow passageways, where the locations of the obstacles were unknown before deployment. The mission was considered successful if the *Husky* robot reached each of the five target locations. We used Euclidean distance between the commanded and measured positions as a threshold to determine if a target was reached. We generated the values for the configurable parameters using random sampling. We recorded the performance metrics for different values of the configurable parameters. We used the navigation task as a test case and defined the following performance metrics for the ROS navigation stack [20]:

1) *Traveled distance (TD)*: Traveled distance from start to destination.
2) *Robustness in narrow space (RNS)*: We define $\text{narrow space} = \text{Robot}_{\text{footprint}} + \text{Footprint}_{\text{padding}}$, and $\text{RNS} = \frac{1}{N_s}\sum_{i=1}^{N_s}\text{Passed}_{N_s}$, where N_s is the total number of narrow spaces in the known environment, and Passed_{N_s} is the narrow spaces that the robot successfully passed.
3) *Mission time*: Total time (*minute*) to complete a mission.
4) *Recovery executed (RE)*: Number of `rotate recovery` and `clear costmap recovery` executed per mission.
5) *Replanning path (RP)*: Number of replanning paths performed by the planner during a mission.
6) *Error rotating to the goal (ERG)*: Number of errors when rotating to a goal per mission execution. If the robot reaches the goal and stops, we check if there is a potential collision while rotating.

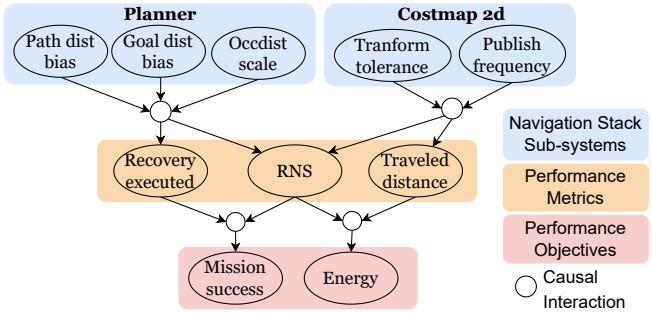
Additionally, we integrate the Gazebo battery plugin [27] to the *Husky* simulator to measure energy consumption. We developed *Reval*[2]—a tool to evaluate ROS-based robotic systems, and collected observational data while the *Husky* performed a mission. Additional details about our experiments can be found in the supplementary materials*.

[2]https://github.com/softsys4ai/Reval.git

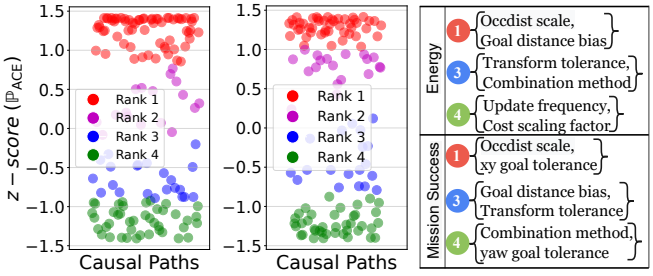


(a) Simulated environment (b) Real environment

Fig. 4: Experimental environments, (a) simulated in *Gazebo*, (b) a real environment located at the University of South Carolina.



(a) A partial causal model for ROS navigation stack discovered in our experiments using the *Husky* simulator.



(b) Energy (c) Mission success (d) Sets of options

Fig. 5: Ranking configuration options applying Algorithm 2.

B. RQ1: Accuracy

We answer RQ1, validating the root causes determined by CARE for both *Husky* in simulation and the physical robot by comparing the *variance* (σ^2) of the performance objectives and performance metrics for different configuration options. Recall that our overall goal is to determine the parameters that influence the performance objective most. By comparing the σ^2, we analyze whether changing the value of a configuration option noticeably affects the performance distribution (options that have a stronger influence are likely to have high variability). We train the causal model using Algorithm 1 on observational data obtained by running a mission 400 times under different configuration settings. A partial causal model resembles the one in Fig. 5a. Next, we compute the $\mathbb{P}_{ACE}$ for each causal path from the causal model using Algorithm 2. The rank of the causal paths is depicted in Fig. 5b-c. Parameters that achieve a higher rank are likely to have spurious values, hence the root cause of the functional fault. We selected two configuration options from Rank 1, Rank 3, and Rank 4 and defined three sets (see Fig. 5d). In our experiment, Rank 2 was discarded because the values of $\mathbb{P}_{ACE}$ (for Rank 2) are too close to Rank 1 and Rank 3. We conducted 50 trials for each rank and recorded the energy, mission success,

TABLE I: Comparison of the variance (σ^2) of different ranks for energy and mission success using the *Husky* platform.

			$\sigma^2_{rank_1}$	$\sigma^2_{rank_3}$	$\sigma^2_{rank_4}$
Husky simulator	Objective	Energy	24.32	13.78	7.27
		Mission success	11.77	11.22	8.01
	Energy	Traveled distance	4.93	3.14	2.90
		Replanning path	123.39	100.97	97.73
		Recovery executed	3.63	2.88	2.70
		Mission time	46.96	29.21	20.93
	Mission success	RNS	0.26	0.25	0.20
		Recovery executed	3.21	2.94	2.79
		Error rotating to goal	44.01	40.59	16.13
		Mission time	153.93	57.75	38.85
Husky physical	Objective	Energy	64.43	26.57	12.43
		Mission success	2.46	2.17	1.60
	Energy	Traveled distance	9.84	5.22	4.23
		Replanning path	126.38	109.07	108.87
		Recovery executed	3.50	2.98	2.67
		Mission time	34.39	25.90	16.93
	Mission success	RNS	0.29	0.23	0.21
		Recovery executed	3.17	2.83	1.26
		Error rotating to goal	56.11	33.59	22.14
		Mission time	62.85	42.43	35.39

and performance metrics by altering only those parameter values contained in the sets (Fig. 5d) while leaving all other parameters to their default values (supplementary materials*). For instance, from Rank 1, we only changed the values of `occdist scale` and `goal distance bias`. Fig. 6 shows violin plots to demonstrate the distribution of the trails for each rank, where the width of each curve corresponds with the frequency of $y-axis$ values. During optimization or debugging, we aimed to prioritize the configuration options which had the strongest influence on the performance objective (e.g., to debug energy fault, ACE of `occdist scale` > `transform tolerance` > `update frequency`). As depicted in Fig. 6a and 6b, for both energy and mission success $\sigma^2_{rank_1} > \sigma^2_{rank_3} > \sigma^2_{rank_4}$. The performance metrics are the confounding variables that influence the performance objectives (e.g., traveled distance $\longrightarrow$ energy, RNS $\longrightarrow$ mission success) and can be treated as the performance variance indicators. For instance, Rank 1 : σ^2_{TD} > Rank 3 : σ^2_{TD} > Rank 4 : σ^2_{TD} (Fig. 6c, 6d, 6e) causes ↑ Rank 1 : σ^2_{energy} (Fig.6a). Similarly, for mission success, Rank 1 : σ^2_{RNS} > Rank 3 : σ^2_{RNS} > Rank 4 : σ^2_{RNS} causes ↓ Rank 4 : $\sigma^2_{mission\ success}$. Table I summarizes the variance for different ranks achieved using the *Husky* platform. We observe that $\sigma^2_{rank_1} > \sigma^2_{rank_3} > \sigma^2_{rank_4}$ for all performance metrics and performance objectives, both in the *Husky* simulator and physical robot, demonstrating that configuration options which rank higher (Fig. 5d) have the strongest influence on the performance objectives. Moreover, CARE achieved 95% accuracy when comparing the predicted root causes with the ground truth data (see Fig. 7b).

C. RQ2: Transferability

The configuration options that specify the hardware characteristics of the physical platform differ across robotic systems (e.g., sensor frequency), and these hardware characteristics can significantly impact the performance of the tasks carried out by the robotic systems. We answer RQ2 by reusing the causal model in a different robotic platform. We reuse the causal model constructed from a source platform, e.g., the *Husky* simulator, to diagnose the root causes of a functional fault in a target platform, e.g., *Turtlebot 3*. We follow the identical experimental setup outlined in §IV-A to record the

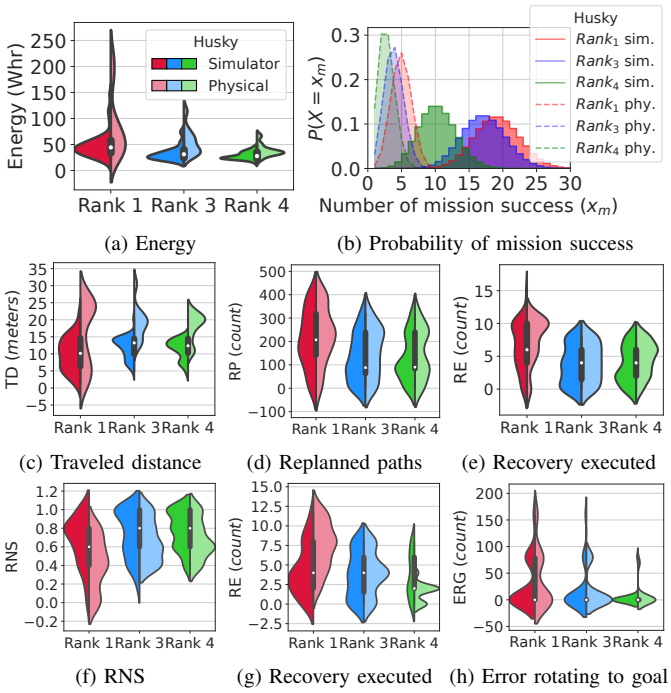


Fig. 6: Comparing the distribution of different ranks for energy (a) and mission success (b), where (c), (d), (e) represent the performance metrics of energy, and (f), (g), (h) represent the performance metrics of mission success.

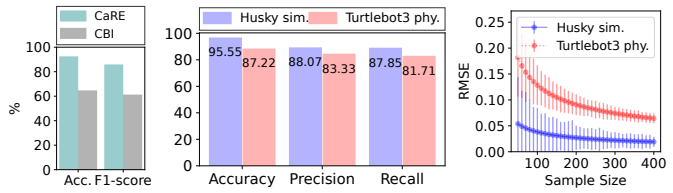
performance metrics for the *Turtlebot*.

a) Ground truth: We measured 400 samples, varying the configuration options for both *Husky* in simulation and *Turtlebot 3*. We used a threshold of 0.02 over the *RMSE* to determine the number of samples required for accurately learning the causal model. We curated a ground truth of functional faults using the ground truth data. In particular, we curated the ground truth for the two performance objectives: (i) configurations that result in a mission failure (functional fault), and (ii) configurations that achieved energy consumption worse than the 99^{th} percentile are labeled as '*faulty*' (non-functional fault). The ground truth contains 20 functional faults (10 mission success, 10 energy), and each has two to four root causes.

b) Baseline: We compared CARE against the state-of-the-art Cooperative Bug Isolation (CBI) [28]—a statistical debugging method that uses a feature selection algorithm. We selected CBI for its use of statistical methods similar to ours and its ability to identify multiple root causes. However, unlike our approach, CBI relies on correlations instead of causation to identify the root causes of the fault. We computed the *Importance score* [28] by computing $Failure(P)$, $Context(P)$, and $Increase(P)$ for different configuration options and objectives. Based on the importance score, we ranked the configuration options similarly to Fig. 5. In our experiments, we set the confidence intervals to 95% to eliminate configuration options with low confidence due to few observations but a high $Increase(P)$.

c) Results: Given a set of test data, ground truth, and CARE's predictions on the test data, we evaluated the predictions by dividing them into true and false positives and negatives (*TP, FP, TN,* and *FN*). Subsequent metrics include:

- *Accuracy*: The measure of the predicted root causes that



(a) Baseline (b) CARE accuracy (c) CARE error

Fig. 7: Comparing CARE against CBI (a), and demonstrating CARE's transferability (b), (c) by reusing the causal model constructed from *Husky* in simulation, to diagnose the root causes of the functional faults in the *Turtlebot 3* physical robot.

match the ground truth root causes, $(TP+TN)/(TP+FP+TN+FN)$.

- *Precision*: The ratio of true root causes among the predicted ones, $TP/(TP+FP)$.
- *Recall*: The ratio of true root causes that are correctly predicted, $TP/(TP+FN)$.
- *F1-score*: The harmonic mean of precision and recall, $2\times(\text{precision}\times\text{recall})/(\text{precision}+\text{recall})$.
- *RMSE*: Weighted difference between the predicted and true root causes. For example, if $\widehat{y}$ is the predicted root cause of a functional fault and y is root cause in the ground truth, we measure *RMSE* = $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\text{ACE}(y)-\text{ACE}(\widehat{y}))^2}$, where ACE is computed using Equation 1.

Fig. 7 shows the results in diagnosing the root causes of the mission success and energy faults. The total *accuracy* is computed using $\frac{1}{N_{obj}}\sum_{i=1}^{N_{obj}} acc.$, where N_{obj} is the number of performance objectives; similarly for *precision* and *recall*. CARE achives 27% more accuracy, and 24% more F1-score compared to CBI (Fig. 7a). We computed the *accuracy* in Fig. 7a using $(\text{Husky}_{total\_acc.}+\text{Turtlebot3}_{total\_acc.})/2$; similarly for F1-score. We observe that reusing CARE in *Turtlebot 3* obtains 8% less accuracy, 4% less precision, and 6% less recall compared to the source platform (*Husky* simulator). We also observe higher *RMSE* in the *Turtlebot 3* platform (the total *RMSE* is computed using $\sum_{i=1}^{N_{obj}} RMSE$). However, if we increase the sample size, CARE incrementally updates the internal causal model with new samples from the target platform to learn the new relationships, and we observe a decrease in *RMSE* (see Fig. 7c). Therefore, the model does transfer reasonably well.

V. DISCUSSION

a) Usability of CARE*:* While our proposed design is general and can be extended to include new variables and other robotics systems, it would require some additional engineering efforts. In particular, to add a new variable to the causal model, the following steps would be required: (i) identifying the manipulable and non-manipulable variables, (ii) profiling the observational data related to the new variable, including its corresponding performance objectives, (iii) learning and adding the causal relationships of the new variable to the existing model. Furthermore, to support a new robotic system, in addition to step 1, profiling the observational data for the entire configuration space would be required to train the causal model. We provide a tool for this in our codebase*, but it is currently limited to ROS-based systems.

b) Why did CARE *outperform CBI?:* CARE discovers the root causes of the configuration bugs by learning a causal model that focuses on the configurations that have the highest causal effect on the performance objectives, eliminating the irrelevant configuration options. For instance, while finding the root causes of the functional and non-functional faults on the performance objectives, CBI reported 116 FP, whereas CARE reported only 13 FP (*Huksy* and *Turtlebot 3* combined), hence, achieving a higher F1-score compared to CBI (Fig. 7a). CBI reported a higher number of FP because it determines the root causes based on the correlation between variables. For instance, it identified planner failure rates increase the $P(\texttt{mission success})$, which is counter-intuitive. Therefore, an engineer would spend less time debugging and optimizing the parameters when using CARE.

c) Limitations: The efficacy of CARE depends on several factors, including the representativeness of the observational data and the presence of unmeasured confounders, which deteriorate the accuracy. In some cases, the causal model may be missing some important connections, resulting in identifying spurious root causes.

d) Future directions: For future work, we envision two possible avenues: empirical and technical. For the empirical aspect, CARE could be applied to improve autonomy in robotic spacecraft missions. The technical aspect could involve performing static analyses to extract the configurable parameters in an automated manner.

VI. RELATED WORK

a) Debugging Approaches: Prior work on highly configurable systems has revealed that the majority of functional faults are related to configuration space [29]. Previous approaches for debugging software systems have used performance influence models [7]–[9] to model configuration options as features and learn a corresponding prediction function. To debug and enhance robotic systems' performance, researchers use random testing such as fuzzing [30] and delta debugging [31] approaches. Moreover, several studies have proposed different methods to deal with the configuration bugs, such as discovering and fixing configuration bugs in co-robotic systems [32], statically identifying run-time architectural misconfigurations [2], and automatic parameter tuning [33]. Data-driven machine learning techniques [34]–[36] have also been widely applied to improve performance by fine-tuning configuration parameters or diagnosing misconfigurations, as opposed to heavily relying on human expertise. However, these techniques may not be effective at applying knowledge in different environments and may have difficulty retaining past information [10].

b) Causal learning for systems: Machine learning techniques have been proven effective at identifying correlations in data, though they are ineffective at identifying causes [15]. To address this challenge, several studies, including detecting and understanding the defect's root causes [37], improving fault localization [38], and reasoning about system's performance [21], utilize causal learning. Using the encoded information, we can benefit from analyses that are only possible when we explicitly employ causal models, in particular, interventional and counterfactual analyses [15], [16]. More recently, Swarmbug [39], a method for debugging configuration bugs in swarm robotics, utilized a causality-based approach to find and fix the misconfigurations in swarm algorithms. However, the Swarmbug method is specifically

designed for use in swarm robotics and is therefore only useful for diagnosing configuration bugs in swarm algorithms.

VII. Conclusion

We proposed CARE, a novel approach to determining the root causes of functional faults in robotic systems. CARE learns and exploits the robotic system's causal structure consisting of manipulable variables (configuration options), non-manipulable variables (performance metrics), and performance objectives. Then, given the causal model, CARE extracts the paths that lead from configuration options to the performance objectives and determines the configuration options that have the highest causal effect on the performance objective by computing the average causal effect of each path. Our evaluation shows that CARE effectively diagnoses the root cause of functional faults, and the learned causal model is transferable across different robotic systems.

ACKNOWLEDGMENT

Part of this research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004). In addition, this work has partly been supported by National Science Foundation (Awards 2233873, 2007202, and 2107463) and the Spanish Government (FEDER/MICINN-AEI) under projects TED2021-130523B-I00 and PID2021-125527NB-I00. We thank Hari Nayar, Michael Dalal, Ashish Goel, Erica Tevere, Anna Boettcher, Anjan Chakrabarty, Ussama Naal, Carolyn Mercer, Issa Nesnas, Matt DeMinico, Md Shahriar Iqbal, and Jianhai Su for contributions to the CARE framework and evaluations on the NASA testbeds: https://nasa-raspberry-si.github.io/raspberry-si/.

References

[1] M. Sayagh, N. Kerzazi *et al.*, "Software configuration engineering in practice interviews, survey, and systematic literature review," *IEEE Transactions on Software Engineering*, vol. 46, no. 6, pp. 646–673, 2020.

[2] C. S. Timperley, T. Dürschmid *et al.*, "Rosdiscover: Statically detecting run-time architecture misconfigurations in robotics systems," in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*. IEEE, 2022, pp. 112–123.

[3] Incorrect configuration causes unexpected behavior at run time in robotic systems. [Online]. Available: https://github.com/softsys4ai/care/wiki/Unexpected-behavior-at-run-time-in-robotic-systems

[4] Z. Liu, B. Chen *et al.*, "Mapper: Multi-agent path planning with evolutionary reinforcement learning in mixed dynamic environments," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 11 748–11 754.

[5] W. Merkt, V. Ivan *et al.*, "Continuous-time collision avoidance for trajectory optimization in dynamic environments," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 7248–7255.

[6] P. S. Schmitt, F. Wirnshofer *et al.*, "Planning reactive manipulation in dynamic environments," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 136–143.

[7] H. Ha and H. Zhang, "Performance-influence model for highly configurable software with fourier learning and lasso regression," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 470–480.

[8] T. Chen and M. Li, "Do performance aspirations matter for guiding software configuration tuning? an empirical investigation under dual performance objectives," *ACM Transactions on Software Engineering and Methodology*, 2022.

[9] S. Mühlbauer, S. Apel *et al.*, "Identifying software performance changes across variants and versions," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 611–622.

[10] B. Liu, X. Xiao *et al.*, "A lifelong learning approach to mobile robot navigation," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1090–1096, 2021.

[11] P. Jamshidi, N. Siegmund *et al.*, "Transfer learning for performance modeling of configurable systems: An exploratory analysis," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 497–508.

[12] A. Gupta, A. Murali *et al.*, "Robot learning in homes: Improving generalization and reducing dataset bias," *Advances in neural information processing systems*, vol. 31, 2018.

[13] M. Allamanis, E. T. Barr *et al.*, "A survey of machine learning for big code and naturalness," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–37, 2018.

[14] W. E. Wong, R. Gao *et al.*, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[15] J. Pearl, *Causality*. Cambridge university press, 2009.

[16] P. Spirtes, C. N. Glymour *et al.*, *Causation, prediction, and search*. MIT press, 2000.

[17] C. Glymour, K. Zhang *et al.*, "Review of causal discovery methods based on graphical models," *Frontiers in genetics*, vol. 10, p. 524, 2019.

[18] M. Kato, K. McAlinn *et al.*, "The adaptive doubly robust estimator and a paradox concerning logging policy," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1351–1364, 2021.

[19] T. Rouček, M. Pecka *et al.*, "Darpa subterranean challenge: Multi-robotic exploration of underground environments," in *International Conference on Modelling and Simulation for Autonomous Systems*. Springer, 2019, pp. 274–290.

[20] Ros navigation stack. [Online]. Available: http://wiki.ros.org/nav_core

[21] M. S. Iqbal, R. Krishna *et al.*, "Unicorn: reasoning about configurable system performance through the lens of causality," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 199–217.

[22] D. Colombo and M. H. Maathuis, "Order-independent constraint-based causal structure learning," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3741–3782, 2014.

[23] J. Pearl, M. Glymour *et al.*, *Causal inference in statistics: A primer*. John Wiley & Sons, 2016.

[24] L. M. Connelly, "Fisher's exact test," *Medsurg Nursing*, vol. 25, no. 1, pp. 58–60, 2016.

[25] M. Kocaoglu, S. Shakkottai *et al.*, "Applications of Common Entropy for Causal Inference," in *NeurIPS*, 2020.

[26] T. Richardson and P. Spirtes, "Ancestral graph markov models," *The Annals of Statistics*, vol. 30, no. 4, pp. 962–1030, 2002.

[27] M. Juhasz. Gazebo-ROS battery plugin. [Online]. Available: https://github.com/nilseuropa/gazebo_ros_battery

[28] B. Liblit, M. Naik *et al.*, "Scalable statistical bug isolation," *Acm Sigplan Notices*, vol. 40, no. 6, pp. 15–26, 2005.

[29] X. Han and T. Yu, "An empirical study on performance bugs for highly configurable software systems," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2016, pp. 1–10.

[30] T. Woodlief, S. Elbaum *et al.*, "Fuzzing mobile robot environments for fast automated crash detection," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 5417–5423.

[31] M. von Stein and S. Elbaum, "Automated environment reduction for debugging robotic systems," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 3985–3991.

[32] A. Taylor, S. Elbaum *et al.*, "Co-diagnosing configuration failures in co-robotic systems," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 2934–2939.

[33] W. P. N. dos Reis, O. Morandin *et al.*, "A quantitative study of tuning ros adaptive monte carlo localization parameters and their effect on an agv localization," in *2019 19th International Conference on Advanced Robotics (ICAR)*, 2019, pp. 302–307.

[34] M. Bhardwaj, B. Boots *et al.*, "Differentiable gaussian process motion planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 10 598–10 604.

[35] D. Teso-Fz-Betoño, E. Zulueta *et al.*, "Predictive dynamic window approach development with artificial neural fuzzy inference improvement," *Electronics*, vol. 8, no. 9, p. 935, 2019.

[36] X. Xiao, Z. Wang *et al.*, "Appl: Adaptive planner parameter learning," *Robotics and Autonomous Systems*, vol. 154, p. 104132, 2022.

[37] B. Johnson, Y. Brun *et al.*, "Causal testing: understanding defects' root causes," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 87–99.

[38] Y. Küçük, T. A. D. Henderson *et al.*, "Improving fault localization by integrating value and predicate based causal inference techniques," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 649–660.

[39] C. Jung, A. Ahad *et al.*, "Swarmbug: debugging configuration bugs in swarm robotics," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 868–880.