CURE: Simulation-Augmented Autotuning in Robotics

Md Abir Hossen, Sonam Kharade, Jason M. O'Kane, *Senior Member, IEEE*, Bradley Schmerl, *Senior Member, IEEE*, David Garlan, *Life Fellow, IEEE*, and Pooyan Jamshidi

***Abstract*—Robotic systems are typically composed of various subsystems, such as localization and navigation, each encompassing numerous configurable components (e.g., selecting different planning algorithms). Once an algorithm has been selected for a component, its associated configuration options must be set to the appropriate values. Configuration options across the system stack interact nontrivially. Finding optimal configurations for highly configurable robots to achieve desired performance poses a significant challenge due to the interactions between configuration options across software and hardware that result in an exponentially large and complex configuration space. These challenges are further compounded by the need for transferability between different environments and robotic platforms. Data efficient optimization algorithms (e.g., Bayesian optimization) have been increasingly employed to automate the tuning of configurable parameters in cyber-physical systems. However, such optimization algorithms converge at later stages, often after exhausting the allocated budget (e.g., optimization steps, allotted time) and lacking transferability. This article proposes causal understanding and remediation for enhancing robot performance (CURE)—a method that identifies causally relevant configuration options, enabling the optimization process to operate in a reduced search space, thereby enabling faster optimization of robot performance. CURE abstracts the causal relationships between various configuration options and the robot performance objectives by learning a causal model in the source (a low-cost environment such as the Gazebo simulator) and applying the learned knowledge to perform optimization in the target (e.g., *Turtlebot 3* physical robot). We demonstrate the effectiveness and transferability of CURE by conducting experiments that involve varying degrees of deployment changes in both physical robots and simulation.**

***Index Terms*—Causal inference, optimization, robot testing, robotics and cyber-physical systems.**

I. Introduction

A ROBOTIC system is composed of hardware and software components that are integrated within a physical machine. These components interact to achieve specific goals in a physical environment. Unfortunately, robots are prone to a wide variety of faults [1]. Incorrect configurations (called *misconfigurations*) in robotic algorithms are one of the most prevalent causes of such faults [2], [3], [4]. Misconfigurations can cause various bugs [5], [6] leading to crashes, robots becoming unstable, deviations from planned trajectory, controller faults, and nonresponsiveness. Several studies have reported misconfigurations as one of the key reasons for cyber-physical system failures. Such misconfigurations caused 19.6% of autonomous aerial vehicles (AAV) bugs [7], 27.25% of autonomous vehicle bugs [8] (a faulty configuration in actuation layer even caused the vehicle to collide with a static object on the curb [9]) and 55% of traffic dispatch algorithm bugs [10]. All of these issues were fixed by configuration changes.

Most robotic algorithms require customization through configuration parameters to suit certain tasks and situations. For example, most AAV controllers include a wide range of configurable parameters that can be customized to different vehicles, flight conditions, or even particular tasks (e.g., when speed is more important than energy use). Finding configurations that optimize performance on a given task is a challenging problem for designers and end users [11]. A developer might request a feature such as *"Create a tool to automatically tune navigation2 node parameters using state-of-the-art machine learning techniques."* [12]. In another instance, a developer encounters a planner performance issue [13] and asks *"I have tuned this for almost 5–6 hours. Sometimes it is going toward the goal but still failing in the middle of the trajectory."* After several back-and-forth communications, the algorithm designer concludes, *"I cannot provide personalized tuning assistance to every user."* In addition, developers aim to maintain the performance of the tuned parameters when deployment changes (e.g., from the robot operating system (ROS) 1 to *ROS 2*) to avoid retuning. Specifically, the optimal configuration determined in one environment often becomes suboptimal in another, as demonstrated in Fig. 1.

Our solution: In this work, we propose causal understanding and remediation for enhancing robot performance (CURE), a multiobjective optimization method that finds optimal configurations for robotic platforms, converges faster than the state-of-the-art, and transfers well from simulation to real robot and even to new untrained platforms. CURE has two main phases. In Phase 1, CURE reduces the search space by eliminating configuration options that do not affect the performance objective causally. For this, we collect observational data in a low-cost source

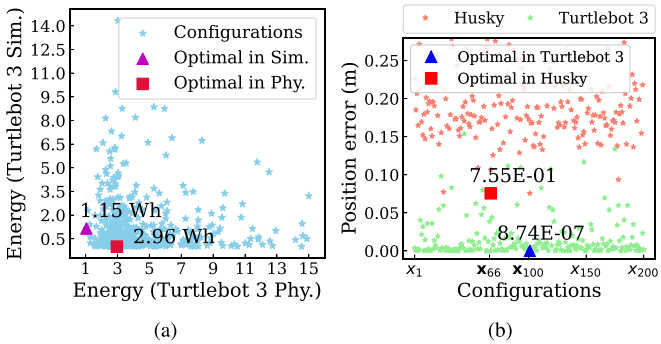
Received 5 September 2024; revised 17 December 2024; accepted 3 February 2025. Date of publication 6 March 2025; date of current version 21 April 2025. This work was supported in part by the National Science Foundation under Award 2107463 and in part by the National Aeronautics and Space Administration under Award 80NSSC20K1720. This article was recommended for publication by Associate Editor F. Amigoni and Editor J. Kober upon evaluation of the reviewers' comments. *(Corresponding author: Md Abir Hossen.)*

Md Abir Hossen, Sonam Kharade, and Pooyan Jamshidi are with the University of South Carolina, Columbia, SC 29208 USA (e-mail: abir.hossen786@gmail.com; kharadesonam@gmail.com; pjamshid@cse.sc.edu).

Jason M. O'Kane is with the Texas A&M University, College Station, TX 77840 USA (e-mail: jokane@tamu.edu).

Bradley Schmerl and David Garlan are with the Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: schmerl@cs.cmu.edu; garlan@cs.cmu.edu).

Digital Object Identifier 10.1109/TRO.2025.3548546



(a) (b)

Fig. 1. Nontransferability of optimal configurations across different environments/platforms. (a) Optimal configuration for *Turtlebot 3* in simulation differs from its physical counterpart. (b) Optimal configuration for *Turtlebot 3* is not suitable in *Husky*.

environment, such as simulation. Then, a causal model is learned on the basis of the data, representing the underlying causal mechanisms that influence robot performance. We then estimate the causal effects of options on performance objectives. Finally, we reduce the search space to a subset of options that have non-negligible causal effects. In Phase 2, CURE performs traditional Bayesian optimization in the target environment, but only over the reduced search space, to find the optimal configuration. We show that CURE not only finds the optimal configuration faster than the state-of-the-art, but the learned causal model in the simulation speeds up optimization in the real robot. The results demonstrate that the learned causal model is transferable across similar but different settings, that is, environments, mission/tasks, and for new robotic platforms. In other words, the existence of a common abstract structure (the causal relations between options, system-level variables, and performance objectives) is invariant across domains, and the behavior of specific features of the environment remains constant across domains.

Evaluations: We evaluated CURE in terms of its *effectiveness* and *transferability* across two tasks: navigation and manipulation. The navigation task forms the core of our experiments, using two highly configurable robotic systems (*Husky* and *Turtlebot 3*) under varying degrees of deployment changes. The manipulation task involves simulating a robot arm (*Franka Emika Panda*) in *Gazebo* to demonstrate CURE's adaptability by complementing the effectiveness evaluation. We compared CURE with traditional multiobjective Bayesian optimization (MOBO) using the Ax framework [14], and RidgeCV [15], [16] integrated with MOBO to reduce the search space. Our results indicate that compared to MOBO, CURE finds a configuration that improves performance by $2\times$ and achieves this improvement with gains in efficiency of $4.6\times$ when we transfer the knowledge learned from *Husky* in simulation to *Turtlebot 3* physical robot.

Contributions: The contributions of our work are as follows.

1) We propose CURE, a multiobjective optimization method that operates in the reduced search space involving causally relevant configuration options and allows faster convergence.
2) We conducted a comprehensive empirical study by comparing CURE with state-of-the-art optimization methods in both simulation and real robots under different severities of deployment changes, and studied effectiveness and transferability.
3) The code and data are available at: https://github.com/softsys4ai/cure

II. Related Work

In this work, we focus on performance optimization through the lens of causality. Specifically, we learn a causal model from a low-cost environment and utilize causal knowledge to optimize performance in the target system. This section groups related work into four categories: optimizing robotic parameters, machine learning (ML) for performance modeling, transfer learning strategies, and causal analysis in configurable systems.

A. Optimization Techniques in Robotic Configurations

Researchers have considered robotic algorithms as a closed-box, as the objective functions in most robotic problems can only be accessible through empirical experiments. Evolutionary algorithms [17], [18] have been used to find optimal configurations in dynamic-window approach (DWA) [19] algorithm. However, the application of evolutionary algorithms in robotic systems is hindered by the limited availability of observations and the difficulty in extracting meaningful information from these observations due to the presence of noise. Approaches such as variational heteroscedastic Gaussian process (GP) regression [20] and Bayesian optimization with safety constraints [21] attempt to address these challenges, but struggle with high-dimensional search spaces, yield only local improvements, and lack transferability across different environments and platforms. Furthermore, the complexity of environmental dynamics models, coupled with the biases introduced by optimization formulation, poses significant challenges. Moreover, formalizing safety constraints that allow for computationally efficient solutions, specifically solutions in polynomial time with closed-form expressions, is complex if at all feasible.

B. Learning-Based Methods for Performance Modeling

Expanding on traditional optimization techniques, machine learning methods offer diverse approaches to improve robotic performance. Approaches such as learning from demonstration [22], learning human-aware path planning [23], and mapping sensory inputs to robot actions [24], [25] have been widely applied to robot navigation beyond fine-tuning configuration parameters, as opposed to heavily relying on human expertise. These methods aim to replace classical methods, casting doubt on the robustness, generality, and safety of the systems. To provide a deeper understanding of performance behavior in robotic algorithms, performance influence models [26], [27], [28] can be used. These models predict system performance by capturing important options and interactions that influence performance behavior using machine learning and sampling heuristics. However, performance influence models face limitations in adapting to unexpected environments due to not being able to capture changes in the performance distribution and often

produce incorrect explanations [4]. In addition, the collection of training data for these models is costly and requires extensive human supervision.

C. Transfer Learning for Performance Modeling

Addressing the challenges of adapting to unexpected environments and costly data collection in learning-based methods, transfer learning accelerates optimization by selectively reusing knowledge from previous tasks. Techniques such as simulation-to-real learning [29], [30] and transferring Pareto frontiers across different platforms [31] improve sampling efficiency and improve training datasets. Each of these techniques uses the predicted transfer learning frameworks based on correlational analysis. However, changes in the environment and robotic platform can cause a distribution shift. The ML models used in these transfer learning approaches are vulnerable to spurious correlations [32], [33].

D. Causal Analysis in Configurable Systems

While machine learning techniques excel in uncovering correlations between variables, their ability to identify causal links is limited [34]. Using the information encoded in causal models, we can benefit from analyses that are only possible when we explicitly employ causal models, such as interventional and counterfactual analyses [34], [35]. Causal analysis has been used for various debugging and optimization tasks in configurable systems, including finding the root cause of intermittent failures in database applications [36], detecting and understanding the root causes of the defect [37], [38], and improving fault localization [39]. The causality analysis in these studies is confined to a single environment and platform, while our approach transfers causal knowledge across different environments and platforms. In robotic systems, the causal models learned in simulation are used to find explanations for failures in real robots [4], [40]. However, such methods are limited to identifying root causes of failures, whereas our approach extends beyond diagnosis to also prescribe remedies, new configuration option values that rectify the failure.

III. Problem Formulation and Challenges

In this section, we first motivate our work by illustrating how an optimal configuration found in one environment often becomes suboptimal in another. We then formally define the problem and describe the challenges.

A. Motivating Scenario

We motivate our work by demonstrating the nontransferability of traditional Bayesian optimization through a simple experiment for robot navigation. In particular, we explore two deployment scenarios: 1) *Sim2Real:* Transferring the optimal configurations for energy consumption identified from simulations to the *Turtlebot 3* physical robot [see Fig. 1(a)] *Real2Real:* Transferring the optimal configurations for position error[1] identified from *Husky* to *Turtlebot 3* [see Fig. 1(b)]. In both scenarios, we observe that the optimal configurations identified by Bayesian optimization in the source environments fail to retain their optimality in the target environment. We observe that energy consumption increases by $2.57\times$, and a significant increase in position error is observed by 8.64×10^5 times.

[1]defined as the Euclidean distance between goal position and robot's actual position

B. Problem Formulation

Consider a highly configurable robot with d distinct configurations. Let X_i indicate the configuration parameter i, which can be assigned a value from a finite domain $\text{Dom}(X_i)$. In general, X_i may be set to the following:

1) a real number (e.g., the number of iterative refinements in a localization algorithm, the frequency of the controller) within specified bounds, denoted as $X_i \in [\underline{X}_i, \bar{X}_i]$, where $\underline{X}_i$ and $\bar{X}_i$ are the lower and upper bounds, respectively;
2) binary (e.g., whether to enable recovery behaviors);
3) categorical (e.g., planner algorithm names).

The configuration space is mathematically a Cartesian product of all the domains of the parameters of interest $\mathcal{X} = \text{Dom}(X_1) \times \cdots \times \text{Dom}(X_d)$. Then, a configuration $\mathbf{x}$, which is in the configuration space $\mathbf{x} \in \mathcal{X}$, can be instantiated by setting a specific value for each option within its domain, $\mathbf{x} = \langle X_1 = x_1, X_2 = x_2, \ldots, X_d = x_d \rangle$. Finding a configuration that uniformly optimizes all objectives is typically not possible; instead, there is a tradeoff between them. Pareto optimal solutions signify the prime balance among all objectives. In the context of minimization, a configuration $\mathbf{x}$ is said to *dominate* another configuration $\mathbf{x}'$ if $f(\mathbf{x}) \leq f(\mathbf{x}')$. A configuration $\mathbf{x} \in \mathcal{X}$ is called *Pareto-optimal* if it is not dominated by any other configuration $\mathbf{x}' \in \mathcal{X}$, where $\mathbf{x} \neq \mathbf{x}'$. The goal is to find $\mathbf{x}^*$, a configuration that gives rise to Pareto-optimal performance in the multiobjective space (e.g., f_1 : failure rate, f_2 : mission time, f_3 : energy consumption), given some constraints (h : safety). Here, we assume that the performance measure can be evaluated in experiments for any configuration $\mathbf{x}$, and we do not know the underlying functional representation of the performance. The problem can be generalized by defining an arbitrary number of performance objectives (if they can be computed over a finite time horizon). Mathematically, we represent performance objectives as black-box functions that map from a configuration space to a real-valued one: $f(\mathbf{x}) : \mathcal{X} \to \mathcal{R}$. In practice, we learn f by sampling the configuration space and collecting the observations data, i.e., $y_i = f(\mathbf{x}_i) + \epsilon_i$ with $\epsilon \sim \mathcal{N}(0, \sigma^2)$. In other words, we only partially know the response function through observations $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^d$, $|\mathcal{D}| \ll |\mathcal{X}|$. We define the problem formally as follows:

$$\boldsymbol{x}^* = \arg\min_{\mathbf{x} \in \mathcal{X}} f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x}), \text{s.t.} : h(\mathbf{x}) \geq 0 \quad (1)$$

where $\mathbf{x}^* \in \mathcal{X}$ is a Pareto-optimal configuration and adhere to the safety constraints.

C. Challenges

In this article, our objective is to propose a solution to address the following key challenges.

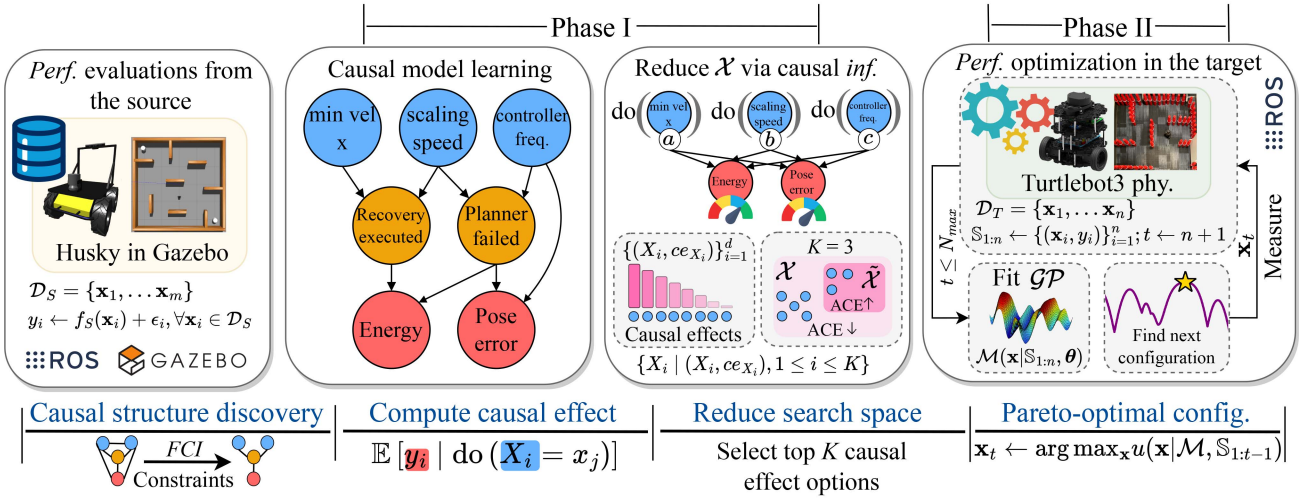


Fig. 2. CURE overview.

1) Software-Hardware Interactions and Exponentially Growing Configuration Space: A robotic system consists of software components (e.g., localization, navigation, and planning), hardware components (e.g., computer and sensors onboard), and middleware components (e.g., ROS), with most components being configurable. The configuration space of only 100 parameters with only 10 possible values for each comprises of 10^{100} possible configurations. (For comparison, the number of atoms in the universe is estimated to be only 10^{82}.) Therefore, the task of finding Pareto-optimal configurations for highly configurable robots and other cyberphysical systems is orders of magnitude more difficult because of software-hardware interactions, compared with software systems.

2) Reality Gap and Negative Transfer From STR: Robot simulators have been extensively used in testing new behaviors before the new component is used in real robots. However, the measurements from simulators typically contain noise, and the observable effect for some configuration options may not be the same in a real robot operating in a real environment, and in some cases, such effect may even have the opposite effect. Therefore, any reasoning based on the model predictions learned based on simulation data may become misleading. Such a reality gap between the sim and real exists due to unobservable confounders as a result of simplifications in the sim. Still, there exist stable relationships between configuration options and performance objectives in the two environments that can facilitate performance optimization of real robots.

3) Multiple Objectives: It is common to find multiple performance objectives in mission specifications (e.g., mission time, energy, and safety). Typically, the objectives involved in the specification are independent of each other [41], but in some cases they can be correlated and conflicting; for example, faster task completion could lead to higher energy consumption. Therefore, finding the optimal configuration (for a given robotic platform in a specific environment and for a specific task) should be treated as a multiobjective optimization problem.

4) Costly Acquisition of Training Data and the Safety Critical Nature of Robotic Systems: Algorithm parameters can be manually adjusted by experiments on real robots or by using massive amounts of training data when the robotic system contains elements that are difficult to hard-code (e.g., computer vision components) [42]. However, collecting training data from real robots is time-consuming and often requires constant human supervision [43]. To guarantee the safe behavior of the robot, the practitioner must either meticulously select configurations that are safe or acquire an ample amount of representative data that lead to safe behavior.

IV. Causal Understanding and Remediation for Enhancing Robot Performance

To solve the optimization problem described in Section III, we propose a novel approach, called CURE. The high-level overview of CURE is shown in Fig. 2. CURE works in two phases. In Phase I, CURE reduces the search space for the optimization problem using data from the source environment, while in Phase II, CURE performs a black-box optimization in the reduced search space on the target platform. To elaborate on the details, in Phase I, CURE learns a structural causal model that enforces structural relationships and constraints between variables using performance evaluations from the source platform (e.g., *Husky* in simulation). Specifically, we learn a causal model for a set of random samples[2] taken in the source environment.[3] The configuration options are then ranked by measuring their *average causal effect* (ACE) on the performance objectives through causal interventions. Options with the largest causal effect are selected to reduce the search space. Next, in Phase II, CURE

[2]Instead of random samples, other partial designs (e.g., Latin Hypercube) could have been used, however, we experimentally found that random samples give rise to more reliable conditional independence tests in the structure learning algorithm.

[3]Here, the source environment could be a simulator like Gazebo or another robotic platform. The assumption is that the source is an environment in which we can intervene at a lower cost.

performs a black-box optimization in the reduced search space given a fixed sampling budget in the target platform (e.g., the physical *Turtlebot 3*). Specifically, CURE searches for Pareto-optimal configurations in the target, iteratively fits a surrogate model to the samples, and selects the next sample based on an acquisition function until the budget is exhausted. CURE's high-level procedure is described in Algorithm 1.

A. Phase I: Reducing the Search Space Via Causal Inference

Phase I begins by recording performance metrics for s initial configurations $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_s, y_s)\}$ in the source environment (Algorithm 1: lines 1–2). We define the following three types of variables to learn the causal structure:

1) software-level configuration options (e.g., hyperparameters in different algorithms [44]) and hardware-level options (e.g., sensor frequency),
2) intermediate performance metrics (e.g., different system events in ROS) that map the influence of configuration options on performance objectives, and
3) end-to-end performance objectives (e.g., task completion rate, mission time).

We also define structural constraints (e.g., $X_i \nrightarrow X_j$) over the causal structure to incorporate domain knowledge that facilitates learning with low sample sizes.[4]

To discover the causal structure, we use an existing structure learning algorithm *fast causal inference* (FCI). We select FCI because 1) it can identify unobserved confounders [35], [45], and 2) it can handle variables of various typologies, such as nominal, ordinal, and categorical given a valid conditional independence test. Algorithm 2 describes the details of our causal learning procedure. It starts by constructing an undirected fully connected graph G, where the nodes represent the variables (options, intermediate variables, performance metrics). Next, we evaluate the independence of all pairs of variables conditioned on all remaining variables using Fisher's z test [46] to remove the edges between independent variables. Finally, a *partial ancestral graph* (PAG) is generated (Algorithm 2: line 2), orienting the undirected edges using the edge orientation rules [35], [45], [47].

A PAG is composed of directed, undirected, and partially directed edges. The partially directed edges must be fully resolved to discover the true causal relationships. We employ the information-theoretic *LatentSearch* algorithm proposed by Kocaoglu [48] to orient partially directed edges in PAG through entropic causal discovery (line 3). For each partially directed edge, we follow two steps: (i) establish if we can generate a latent variable (with low entropy) to serve as a common cause between two vertices; (ii) if such a latent variable does not exist, then pick the direction which has the lowest entropy. For the first step, we assess whether there could be an unmeasured confounder (say Z) that lies between two partially oriented nodes (say X and Y). *LatentSearch* outputs a joint distribution $q(X, Y, Z)$ that can be used to compute the entropy $H(Z)$ of the unmeasured confounder Z. Following the Kocaoglu guidelines, we set an entropy threshold $\theta_r = 0.8 \times \min\{H(X), H(Y)\}$. If the entropy $H(Z)$ of the unmeasured confounder falls *below* this threshold, then we declare that there is a simple unmeasured confounder Z (with a low enough entropy) to serve as a common cause between X and Y and accordingly replace the partial edge with a bidirected ($\leftrightarrow$) edge. When there is no latent variable with sufficiently low entropy, there are two possibilities: 1) the variable X causes Y; then there is an arbitrary function $f(\cdot)$ such that $Y = f(X, E)$, where E is an exogenous variable (independent of X) that accounts for system noise; or 2) the variable Y causes X; then there is an arbitrary function $g(\cdot)$ such that $X = g(Y, \tilde{E})$, where

[4] e.g., there should not be any causal connections between configuration options and their values are determined independently.

Algorithm 1: CURE.

Input: Configuration space $\mathcal{X}$, Maximum budget $N_{\max}$, Response function f, Kernel function $\boldsymbol{K}_\theta$, Hyper-parameters $\boldsymbol{\theta}$, Design sample size n, and learning cycle N_l
Output: $\boldsymbol{x}^*$ and learned model $\mathcal{M}$

Dimension Reduction Phase

1 Sample $s \leq N_{\max}$ random configurations from $\mathcal{X}$ within the bounds $X_i \in [\underline{X}_i, \bar{X}_i]$ to form the initial design sample set $\mathcal{D}_S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_s\}$
2 Obtain *performance measurements* of the initial design in the source environment, $y_i \leftarrow f_S(\boldsymbol{x}_i) + \epsilon_i, \forall \boldsymbol{x}_i \in \mathcal{D}_S$
3 $\mathcal{G} \leftarrow$ Learn a causal model on $\mathcal{D}_S$ using Algorithm 2.
4 Estimate the average causal effects of the configuration options by intervening on X_i: $\mathrm{CE}_{X_i} \leftarrow 1/N \sum_{j=1}^{N} \mathbb{E}[Y_i \mid \mathrm{do}(X_i = x_j)] - \mathbb{E}[Y_i \mid \mathrm{do}(X_i = a)]$, where a is the default value of option X_i.
5 Reduce the search space by selecting the top K options with the largest causal effect: $\tilde{\mathcal{X}} \subset \mathcal{X}$

Configuration Optimization Phase

6 Choose an initial sparse design (Sobol sequences) in $\tilde{\mathcal{X}}$ to find an initial design samples $\mathcal{D}_T = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$
7 Obtain *performance measurements* of the initial design in the target environment, $y_i \leftarrow f_T(\boldsymbol{x}_i) + \epsilon_i, \forall \boldsymbol{x}_i \in \mathcal{D}_T$
8 $\mathbb{S}_{1:n} \leftarrow \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}; t \leftarrow n + 1$
9 $\mathcal{M}(\boldsymbol{x}|\mathbb{S}_{1:n}, \boldsymbol{\theta}) \leftarrow$ Fit a $\mathcal{GP}$ model to the design
10 **while** $t \leq N_{\max}$ **do**
11 **if** $(t \mod N_l = 0)$ **then**
12 $\boldsymbol{\theta} \leftarrow$ *Learn* the kernel hyper-parameters by maximizing the likelihood
13 **else**
14 Find *next configuration* $\boldsymbol{x}_t$ by optimizing the selection criteria over the estimated response surface given the data, $\boldsymbol{x}_t \leftarrow \arg\max_{\boldsymbol{x}} u(\boldsymbol{x}|\mathcal{M}, \mathbb{S}_{1:t-1})$
15 Obtain performance for the *new configuration* $\boldsymbol{x}_t$, $y_t \leftarrow f_T(\boldsymbol{x}_t) + \epsilon_t$
16 Add the newly measured configuration to the measurement set: $\mathbb{S}_{1:t} = \{\mathbb{S}_{1:t-1}, (\boldsymbol{x}_t, y_t)\}$
17 *Re-fit* a new GP model $\mathcal{M}(\boldsymbol{x}|\mathbb{S}_{1:t}, \boldsymbol{\theta})$
18 $t \leftarrow t + 1$
19 $(\boldsymbol{x}^*, y^*) = \min \mathbb{S}_{1:N_{\max}}$

Algorithm 2: Causal Model Learning.

Input: Design samples $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$ from $\mathcal{X}$ with outcomes $y_i = f(\boldsymbol{x}_i) + \epsilon_i, \forall \boldsymbol{x}_i \in \mathcal{D}$

Output: Acyclic-directed mixed graph G_{ADMG}

1 Initialize a fully connected undirected graph G

2 Apply Fisher's z test to remove the edges between independent variables and then orient the edges to get G_{PAG}.

3 **for each** *partial edge in* G_{PAG} **do**

4 Resolve the partial edge using the LatentSearch algorithm [48].

5 The resolved graph composed of directed and bi-directed edges: G_{ADMG}

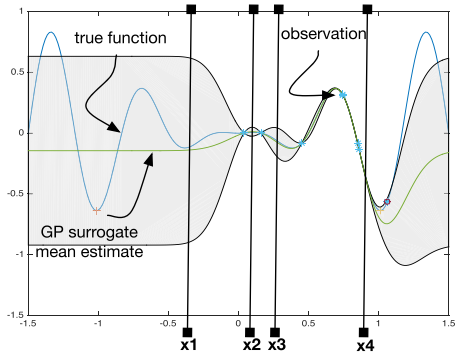


Fig. 3. Example of 1D GP model: GPs provide mean estimates and uncertainty in estimations, i.e., variance.

$\tilde{E}$ is an exogenous variable (independent of Y) that accounts for noise in the system. The distribution of E and $\tilde{E}$ can be inferred from the data. With these distributions, we measure the entropies $H(E)$ and $H(\tilde{E})$. If $H(E) < H(\tilde{E})$, then it is simpler to explain X causes Y (that is, the entropy is lower when $Y = f(X, E)$) and we choose $X \to Y$. Otherwise, we choose $Y \to X$.

The final causal model is an acyclic-directed mixed graph (ADMG). When interpreting a causal model, we view the nodes as variables and the arrows as the assumed direction of causality, whereas the absence of an arrow shows the absence of direct causal influence between variables. To quantify the influence of a configuration option on a performance objective, we need to locate the causal paths. A causal path $P_{X \rightsquigarrow Y}$ is a directed path that originates from a configuration option X to a subsequent nonfunctional property $\mathcal{S}$ (e.g., planner failed) and ends at a performance objective Y. For example, $X \to \mathcal{S} \to Y$ denotes X causes Y through a subsequent node $\mathcal{S}$ on the path. We discover $P_{X \rightsquigarrow Y}$ by backtracking the nodes corresponding to each of the performance objectives until we reach a node without a parent. We then measure the ACE, by measuring the causal effects of the configuration options on the performance metrics and taking the average over the causal paths. We then rank the configuration options according to their ACE: $\{(X_i, \text{CE}_{X_i})\}_{i=1}^{d}$, where $\text{CE}_{X_i} \geq \text{CE}_{X_{i+1}}$ for all $i3d$. Finally, we select a subset of configuration options with the highest ACE: $\{X_i \mid (X_i, \text{CE}_{X_i}), 1 \leq i \leq K\}$, $K \leq d$, and reduce the search space to $\tilde{\mathcal{X}} \subset \mathcal{X}$ (Algorithm 1: lines 4–5).

B. Phase II: Performance Optimization Through Black-Box Optimization With Limited Budget

In the configuration optimization phase (lines 6–18), we search for Pareto optimal configurations using an active learning approach that operates in the reduced search space in the target environment. Here, the target environment is typically the target robotic platform that we want to optimize. The assumption is that any intervention in the target environment is costly and that we typically assume a small sampling budget. In some situations, we could assume that the cost of measuring configurations varies. For example, if the likelihood of violating safety confidence is high for a specific configuration, we could assign a higher cost to that configuration because it may damage the robot. We leave this assumption for future work. Specifically, we start by bootstrapping optimization by randomly sampling the reduced configuration space to produce an initial design $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}_i \in \tilde{\mathcal{X}}$. After obtaining the measurements regarding the initial design, CURE then fits a GP model to the design points $\mathcal{D}$ to form our belief about the underlying response function. The while loop in Algorithm 1 iteratively updates the belief until the budget runs out: As we accumulate the data $\mathbb{S}_{1:t} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{t}$, where $y_i = f_T(\mathbf{x}_i) + \epsilon_i$ with $\epsilon \sim \mathcal{N}(0, \sigma^2)$, a prior distribution $\Pr(f_T)$ and the likelihood function $\Pr(\mathbb{S}_{1:t}|f_T)$ form the posterior distribution: $\Pr(f_T|\mathbb{S}_{1:t}) \propto \Pr(\mathbb{S}_{1:t}|f_T)\Pr(f_T)$. We describe the steps of Phase II as follows.

1) Bayesian Optimization With GP: Bayesian optimization is a sequential design strategy that allows us to perform global optimization of black-box functions [49]. The main idea of this method is to treat the black-box objective function $f(\boldsymbol{x})$ as a random variable with a given prior distribution and then optimize the posterior distribution of $f(\boldsymbol{x})$, given experimental data. In this work, we use GPs to model this black-box objective function at each point $\boldsymbol{x} \in \mathbb{X}$. That is, let $\mathbb{S}_{1:t}$ be the experimental data collected in the first t iterations, and let $\mathbf{x}_{t+1}$ be a candidate configuration that we can select to run the next experiment. Then, the probability that this new experiment could find an optimal configuration using the posterior distribution will be assessed

$$\Pr(f_{t+1}|\mathbb{S}_{1:t}, \mathbf{x}_{t+1}) \sim \mathcal{N}(\mu_t(\mathbf{x_{t+1}}), \sigma_t^2(\mathbf{x}_{t+1}))$$

where $\mu_t(\mathbf{x_{t+1}})$ and $\sigma_t^2(\mathbf{x}_{t+1})$ are suitable estimators of the mean and standard deviation of a normal distribution used to model this posterior. The main motivation behind the choice of GPs as prior here is that it offers a framework in which reasoning can be based not only on mean estimates, but also on variance, providing more informative decision making. The other reason is that all the computations in this framework are based on a solid foundation of *linear algebra*. Fig. 3 illustrates Bayesian optimization based on GP using a one-dimensional response surface. The blue curve represents the unknown true posterior distribution, while the mean is shown in green, and the confidence interval 95% is shaded. Stars indicate measurements carried out in the past and recorded in $\mathbb{S}_{1:t}$ (i.e., observations). The configuration corresponding to $\mathbf{x}_1$ has a large confidence interval due to the lack of observations in its neighborhood.

On the contrary, $\mathbf{x}_4$ has a narrow confidence since neighboring configurations have been experimented with. The confidence interval in the neighborhood of $\mathbf{x}_2$ and $\mathbf{x}_3$ is not large, and correctly our approach does not decide to explore these zones. The next configuration $\mathbf{x}_{t+1}$, indicated by a small circle on the right side of $\mathbf{x}_4$, is selected based on a criterion that will be defined later. A GP is a distribution over functions, specified by its mean and covariance

$$y = f(\mathbf{x}) \sim \mathcal{GP}(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{2}$$

where $k(\mathbf{x}, \mathbf{x}')$ defines the distance between $\mathbf{x}$ and $\mathbf{x}'$. Assume $\mathbb{S}_{1:t} = \{(\mathbf{x}_{1:t}, y_{1:t}) | y_i := f(\mathbf{x}_i)\}$ to be the collection of observations t. The function values are drawn from a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$, where $\boldsymbol{\mu} := \mu(\mathbf{x}_{1:t})$

$$\mathbf{K} := \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_t) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_t, \mathbf{x}_1) & \dots & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix}. \tag{3}$$

In the while loop in CURE, given the observations we accumulated so far, we intend to fit a new GP model

$$\begin{bmatrix} \mathbf{f}_{1:t} \\ f_{t+1} \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{\mu}, \begin{bmatrix} \mathbf{K} + \sigma^2\mathbf{I} & \mathbf{k} \\ \mathbf{k}^\intercal & k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \end{bmatrix}\right) \tag{4}$$

where $\mathbf{k}(\mathbf{x})^\intercal = [k(\mathbf{x}, \mathbf{x}_1) \quad k(\mathbf{x}, \mathbf{x}_2) \quad \dots \quad k(\mathbf{x}, \mathbf{x}_t)]$ and $\mathbf{I}$ is identity matrix. Given (4), the new GP model can be drawn from this new Gaussian distribution

$$\Pr(f_{t+1} | \mathbb{S}_{1:t}, \mathbf{x}_{t+1}) = \mathcal{N}(\mu_t(\mathbf{x_{t+1}}), \sigma_t^2(\mathbf{x}_{t+1})) \tag{5}$$

where

$$\mu_t(\mathbf{x}) = \mu(\mathbf{x}) + \mathbf{k}(\mathbf{x})^\intercal(\mathbf{K} + \sigma^2\mathbf{I})^{-1}(\mathbf{y} - \boldsymbol{\mu}) \tag{6}$$

$$\sigma_t^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) + \sigma^2\mathbf{I} - \mathbf{k}(\mathbf{x})^\intercal(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}(\mathbf{x}). \tag{7}$$

These posterior functions are used to select the next point $\mathbf{x}_{t+1}$.

2) Configuration Selection Criteria: The selection criteria is defined as $u : \mathbb{X} \to \mathbb{R}$ that selects $\mathbf{x}_{t+1} \in \mathbb{X}$, should $f(\cdot)$ be evaluated next (*step 7*)

$$\mathbf{x}_{t+1} = \arg\max_{\mathbf{x} \in \mathbb{X}} u(\mathbf{x} | \mathcal{M}, \mathbb{S}_{1:t}). \tag{8}$$

Although there are several different criteria in the literature for multiobjective optimization [50], [51], [52], CURE utilizes expected hypervolume improvement (EHVI). EHVI has demonstrated its strength in balancing exploration and exploitation, and in producing Pareto fronts (PFs) with excellent coverage and faster optimization [53]. EHVI operates by assessing the expected improvement of a given point in the solution space in terms of the hypervolume (HV) measure—a widely accepted metric for comparing the quality of solutions in multiobjective optimization. EHVI is particularly useful in robotic applications, where the solution landscape can be highly complex and multidimensional. The steps of Algorithm 1 are illustrated in Fig. 4. First, an initial design based on random sampling is produced [see Fig. 4(a)]. Second, a GP model is fitted to the initial design [see Fig. 4(b)]. The model is then used to calculate the selection criteria [see Fig. 4(c)]. Finally, the configuration

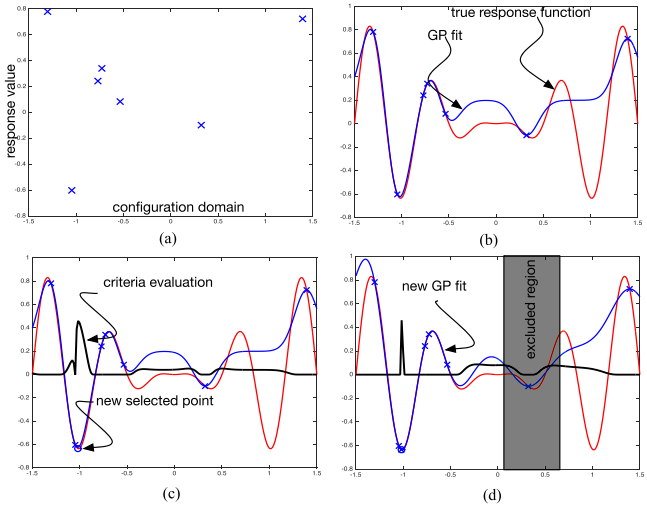


Fig. 4. Illustration of configuration parameter optimization. (a) Initial observations. (b) GP model fit. (c) Choosing the next point. (d) Refitting a new GP model.

that maximizes the selection criteria is used to run the next experiment and provide data to reconstruct a more accurate model [see Fig. 4(d)].

3) Model Fitting: Here, we provide some practical considerations to make GPs applicable for configuration optimization. In CURE, as shown in Algorithm 1, the covariance function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ dictates the structure of the response function that we fit to the observed data. For integer variables, we implemented the Matérn kernel [54]. The main reason behind this choice is that along each dimension of the configuration response functions, a different level of smoothness can be observed. Matérn kernels incorporate a smoothness parameter $\nu > 0$ that allows greater flexibility in modeling such functions. The following is a variation of the Matérn kernel for $\nu = 1/2$

$$k_{\nu=1/2}(\mathbf{x}_i, \mathbf{x}_j) = \theta_0^2 \exp(-r) \tag{9}$$

where $r^2(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^\intercal\boldsymbol{\Lambda}(\mathbf{x}_i - \mathbf{x}_j)$ for some positive semidefinite matrix $\boldsymbol{\Lambda}$. For categorical variables, we implement the following [55]:

$$k_\theta(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\Sigma_{\ell=1}^d(-\theta_\ell\delta(\mathbf{x}_i \neq \mathbf{x}_j))\right) \tag{10}$$

where d is the number of dimensions (i.e., the number of configuration parameters), θ_ℓ adjust the scales along the function dimensions, and δ is a function gives the distance between two categorical variables using Kronecker delta [55], [56]. TL4CO uses different scales $\{\theta_\ell, \ell = 1 \dots d\}$ on different dimensions as suggested in [54] and [56], this technique is called automatic relevance determination. After learning the hyperparameters (step 6), if the ℓth dimension turns out to be irrelevant, then θ_ℓ will be a small value and, therefore, will be discarded. This is particularly helpful in high-dimensional spaces where it is difficult to find the optimal configuration. Although the kernel controls the structure of the estimated function, the prior mean $\mu(\mathbf{x}) : \mathbb{X} \to \mathbb{R}$ provides a possible offset for our estimate. By default, this function is set to a constant $\mu(\mathbf{x}) := \mu$, which is inferred from observations [56]. However, the prior mean

function is a way of incorporating expert knowledge, and if it is available, then we can use this knowledge. Fortunately, we have collected extensive experimental measurements and based on our datasets, we observed that, for robotic systems, there is typically a significant distance between the minimum and the maximum of each function (see Figs. 17 and 18). Therefore, a linear mean function $\mu(\mathbf{x}) := \mathbf{a}\mathbf{x} + b$ allows for more flexible structures and provides a better fit for the data than a constant mean. We only need to learn the slope for each dimension and the offset (denoted $\mu_\ell = (\mathbf{a}, b)$). Due to the heavy learning computation (*step 12* in Algorithm 1), this process is computed only for every N_lth iteration. To learn the hyperparameters of the kernel and also the prior mean functions, we maximize the marginal likelihood [56] of the observations $\mathbb{S}_{1:t}$. To do that, we train the GP model (6) with $\mathbb{S}_{1:t}$. We optimize the marginal likelihood using multistarted quasi-Newton hill climbers [54]. For this purpose, we used the Ax + BoTorch library. Using the kernel defined in (10), we learn $\boldsymbol{\theta} := (\theta_{0:d}, \mu_{0:d}, \sigma^2)$, which comprises the hyperparameters of the kernel and the mean functions. The learning is performed iteratively, resulting in a sequence of $\boldsymbol{\theta}_i$ for $i = 1 \ldots \lfloor \frac{N_{\max}}{N_\ell} \rfloor$.

V. Experiments and Results

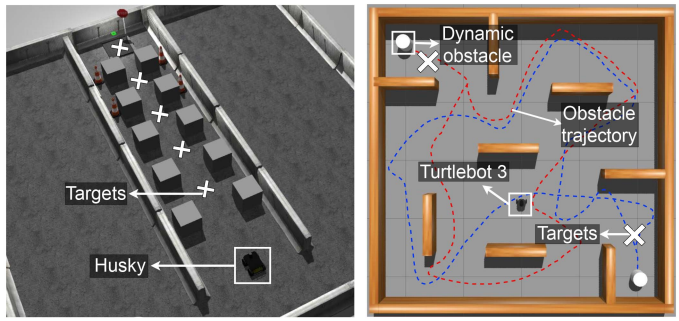
To evaluate this work, we answer the following research questions (RQs).

1) *RQ1 (Effectiveness)*: How effective is CURE in the following:
   1) ensuring optimal performance;
   2) utilizing the budget;
   3) respecting the safety constraints compared to the baselines?
2) *RQ2 (Transferability)*: How does the effectiveness of CURE change when the severity of deployment changes varies [e.g., environment and platform change (PC)]?

We answered these questions in a robot navigation task, using *Husky* and *Turtlebot 3* platforms. In addition, to illustrate adaptability of CURE to different tasks, we also demonstrate RQ1 on a robot manipulation task, using the *Franka Emika Panda* platform in *Gazebo*.

A. Experimental Setup

1) Robot Navigation: We simulate *Husky* and *Turtlebot 3* in *Gazebo* to collect the observational data by measuring the performance metrics (e.g., planner failed) 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 physical robots, but CURE also works with data from physical robots. We deploy the robot in a controlled indoor environment and direct the robot to navigate autonomously to the target locations [see Fig. 5(a)]. The robot was expected to encounter obstacles and narrow passageways, where the locations of the obstacles were unknown prior to deployment. The mission was considered successful if the robot reached each of the target locations. We fixed the goal tolerance parameters (xy_goal_tolerance=0.2, and yaw_goal_tolerance=0.1) to determine whether a target



(a) (b)

Fig. 5. Simulated environments for *Husky* and *Turtlebot 3*. The dashed lines in (b) show the trajectory of the dynamic obstacles. (a) Husky in Gazebo. (b) Turtlebot 3 in Gazebo.

was reached. We defined the following properties for the ROS navigation stack [44].

1) *Task completion rate:* $\mathcal{T}_{\text{cr}} = (\sum \text{Tasks}_{\text{completed}})/(\sum \text{Tasks})$.
2) *Traveled distance:* Distance traveled from start to destination.
3) *Mission time:* Total time to complete a mission.
4) *Position error:* Euclidean distance between the actual target position and the position reached by the robot, $E_{\text{dist}} = \sqrt{\sum_{i=1}^{n}(t_i - r_i)^2}$, where t and r denote the target and position reached by the robot, respectively.
5) *Recovery executed:* Number of rotate recovery and clear costmap recovery executed per mission.
6) *Planner failed:* Number of times the planner failed to produce a path during a mission.

2) Robot Manipulation: We simulate the *Franka Emika Panda* in *Gazebo* and perform a pick-and-place task using the *Moveit* [57] motion planning framework. To learn a causal model, we measure the following performance objectives under different configuration settings: 1) *Average trajectory jerk:* Rate of change of acceleration, averaged across all joints and time steps, we define average jerk $= \frac{1}{N}\sum_{t=1}^{N}\sqrt{\sum_{j=1}^{7}(\frac{a_j(t)-a_j(t-1)}{\Delta t})^2}$, where N is the total number of time steps, $a_j(t)$ is the acceleration of joint j at time t, and Δt is the time interval between consecutive time steps; and 2) *Task execution time:* The total execution time from picking up an object to placing.

B. Evaluation

To learn a causal model from the source (a low-cost environment), we generated the values for the configurable parameters using random sampling and recorded the performance metrics (the intermediate layer of the causal model that maps the influence of the configuration options to the performance objective) for different values of the configurable parameters. We use a budget of 200 iterations for each method. When running each method for the same budget, we compare the Pareto front (PF) and Pareto HV. The PF is the set of objective vectors corresponding to all Pareto-optimal configurations in the configuration space $\mathcal{X}$. The Pareto HV is commonly used to

measure the quality of an estimated PF [58], [59]. We define the PF and HV as follows:

$$\text{PF} = \{(f_j(\mathbf{x}))_{j=1}^{m} \mid \mathbf{x} \in \mathcal{X} \text{ is Pareto-optimal}\} \quad (11)$$

$$\text{HV}(\mathbf{x}^*, f^{\text{ref}}) = \Lambda\left(\bigcup_{\mathbf{x}_n^* \in \mathbf{x}^*} \prod_{j=1}^{m} [f_j(\mathbf{x}_n^*), f_j^{\text{ref}}]\right) \quad (12)$$

where $\text{HV}(\mathbf{x}^*, f^{\text{ref}})$ resolves the size of the dominated space covered by a nondominated set $\mathbf{x}^*$, f^{ref} refers to a user-defined reference point in the objective space, and $\Lambda(.)$ refers to the Lebesgue measure. In our experiments, we fixed the f^{ref} points to the maximum observed values of each objective among all the methods.

To compare the efficiency of each method, we define an efficiency metric $\eta = (\sum_{k=1}^{n} \mathcal{T}_k)/(\sum_{k=1}^{n} k)$, where $\mathcal{T}_k$ is a binary variable taking values 0 or 1, denoting the success of a task during the kth iteration. We also compare the number of unsuccessful execution (e.g., when the robot failed to complete a task) and the number of constraint violations (e.g., when the robot completed the task but violated a constraint). We compared CURE with the following baselines.

1) *MOBO:* We implement MOBO using Ax [14]—an optimization framework that can optimize discrete and continuous configurations.
2) *RidgeCV [15], [16]:* A feature extraction method that selects the important features based on the highest absolute coefficient. We use RidgeCV to determine the important configuration options and generate a reduced search space, which consists of only the important configuration options. We then perform an optimization using MOBO on the reduced search space.

C. RQ1: Effectiveness

We evaluated the effectiveness of CURE in finding an optimal configuration compared to the baselines. We collect observational data by running a mission 1000 times from *Husky* in simulation under different configuration settings and recorded the performance objectives. In Fig. 6, the histograms of performance objectives are depicted along the diagonal line, while scatter plots illustrating pairs of performance objectives are displayed outside the diagonal. The histograms of performance objectives, namely planner failed, recovery executed, obstacle distance, and energy, have shapes similar to one half of a Gaussian distribution. Scatter plots depicting different pairs of performance objectives, such as mission time, distance traveled, and energy, exhibit positive linear relationships. We selected energy and position error as the two performance objectives given the imperative to incorporate uncorrelated objectives in the multiobjective optimization framework, underscored by their lowest correlation coefficient, ensuring the diversity of the optimization criteria. We then learn a causal model using observational data. The search space was reduced according to the estimated causal effects on performance objectives and constraints by selecting top K configuration options (e.g., $\{\text{Energy}_{\text{top}K}\} \cup \{\text{PoseError}_{\text{top}K}\} \cup \{\text{Safety}_{\text{top}K}\}$) and performed optimization using Algorithm 1.

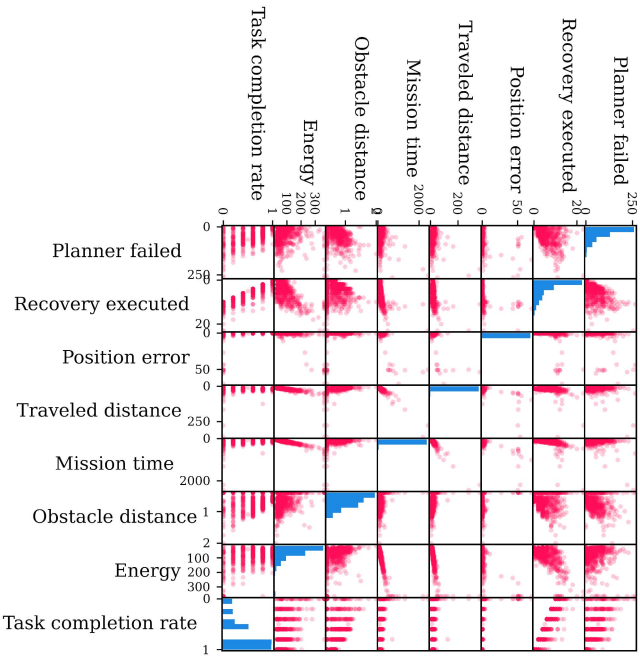


Fig. 6. Correlation between different performance objectives derived from observational data.

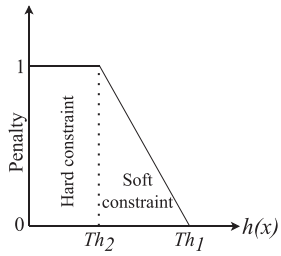


Fig. 7. Penalty function.

1) Setting: For the *Husky* robot, we set the objective thresholds $\text{Energy}_{\text{Th}} = 40$ Wh and $\text{PoseError}_{\text{Th}} = 0.18$ m. We compute the HV using (12) by setting the f^{ref} points at 400 for energy and 35 for position error within the coordinate system. We incorporate the safety constraint $h(\mathbf{x})$ by defining a test case, where the robot must maintain a minimum distance from obstacles to avoid collisions. We incorporate a user defined penalty function (see Fig. 7) for each instance $0 \leq \alpha h(\mathbf{x}) \leq 1$ that penalizes $\mathcal{T}_{\text{cr}}$ if $h(\mathbf{x})$ is violated. In Fig. 7, Th_1 is a soft constraint threshold and Th_2 is a hard constraint threshold. That is, we penalize $\mathcal{T}_{\text{cr}}$ gradually if $\text{Th}_1 > h(x) > \text{Th}_2$ and give the maximum penalty if $h(x) < \text{Th}_2$ to ensure safety. In our experiments, we set $\text{Th}_1 = 0.25$ and $\text{Th}_2 = 0.18$. We defined the safety constraint: $\mathcal{T}_{\text{cr}} - \frac{1}{N}\sum_{k=0}^{N} \alpha_k h(\mathbf{x}) \geq \theta$, where θ is a user-defined threshold. In our experiments, we set $\theta = 0.8$. For the manipulation task, we set the f^{ref} points at 16 for task execution time and 113 for average trajectory jerk.

2) Results: CURE performed better than MOBO and RidgeCV-MOBO in finding a PF with a higher HV, as shown

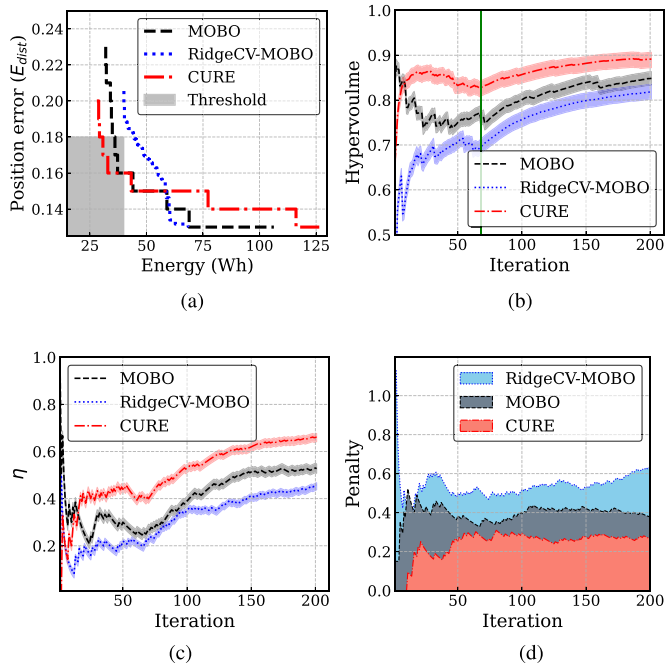


Fig. 8. Effectiveness of CURE and baseline methods for the navigation task: (a) PF; (b) HV, (c) efficiency; and (d) safety penalty response obtained by CURE and other approaches for *Husky* in simulation. The vertical green line in (b) shows the number of initial trails before fitting the GP model.

in Fig. 8. In our experiments, we observed a comparable PF between CURE and MOBO [see Fig. 8(a)], which can be attributed to MOBO's exploration of an extensive search space that includes all possible configuration options. On the contrary, CURE confines its exploration to a reduced search space, composing only configuration options with a greater causal effect on performance objectives. Although CURE and MOBO have a similar PF, CURE achieved a higher HV with a less amount of budget [see Fig. 8(b)]. Fig. 9 illustrates the budget utilization of CURE and baseline methods. CURE demonstrated better budget utilization, as reflected in the increased density of purple-colored data points surrounding the PF and the achievement of a higher $\mathcal{T}_{\text{cr}}$ in fewer iterations compared to the baseline methods. When comparing the penalty response given, we observed CURE selected configuration options that achieved the lower penalty, as shown in Fig. 8(d). Furthermore, CURE outperformed the baselines in terms of efficiency, achieving a $1.3\times$ improvement over MOBO and achieved this improvement $2\times$ faster compared to MOBO [as shown in Fig. 8(c)]. RidgeCV-MOBO, however, underperformed, mainly because it was unable to identify the core configuration options influencing the performance objectives [see Figs. 8(b), (c) and 9(b)]. Moreover, CURE continuously outperformed the baselines in the manipulation task (see Fig. 10). Therefore, CURE is more effective in finding optimal configurations compared to the baselines.

D. RQ2: Transferability

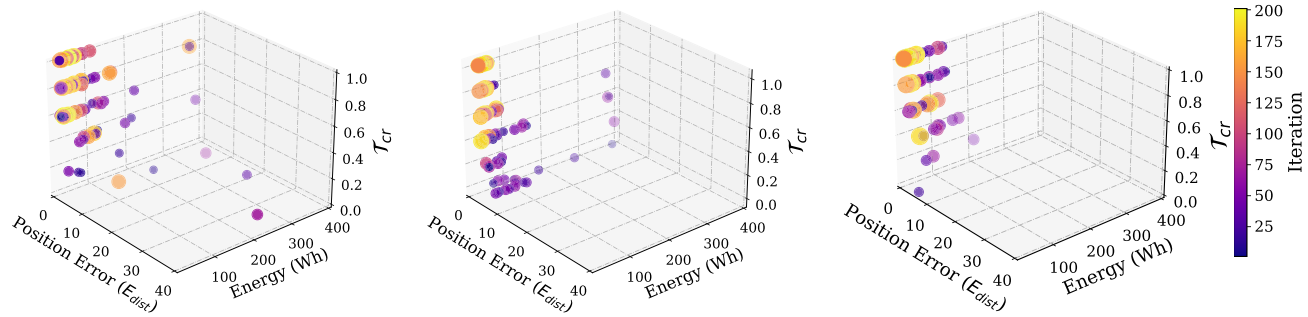
Understanding CURE's sensitivity to different degrees of deployment changes, such as transfer of the causal model learned from a source platform (e.g., *Gazebo* simulation) to a target platform (e.g., real robot), is critical. Sensitivity analysis is especially crucial for such scenarios, considering that distribution shifts can occur during deployment changes. We answer RQ2 through an empirical study. We examine different levels of severity in deployment changes, where severity is determined by the number of changes involved. For example, a deployment change is considered more severe when both the robotic platform and the operating environment change, as opposed to changes limited solely to the environment.

1) Setting: We consider *Husky* and *Turltebot 3* in simulation as the source and *Turtlebot 3* physical robot as the target. We evaluate two deployment scenarios (see Fig. 11): (i) *STR:* We trained the causal model using Algorithm 2 on observational data obtained by conducting a mission 1000 times using *Turtlebot 3* in *Gazebo* environment [see Fig. 5(b)]. The robot was expected to encounter dynamic obstacles [the trajectories of the obstacles are shown in Fig. 5(b)]. The mission was considered successful if *Turtlebot 3* reached each of the target locations. Subsequently, we used the causal model learned from simulation (environment A) to the *Turtlebot 3* physical robot for performance optimization in two distinct environments (environment B and C). (ii) *STR and PC:* We consider the change of two categories, the STR and robotic PC. In particular, we applied the causal model used in RQ1 (learned using *Husky* in simulation) to the *Turtlebot 3* physical robot in a real environment, as shown in Fig. 11. We use the identical experimental setting for the *Husky* as described in Section V-C. For *Turtlebot 3*, we set the objective thresholds, $\text{Energy}_{\text{Th}} = 2$ Wh and $\text{PoseError}_{\text{Th}} = 0.1$ m. We compute the HV using (12) by setting the f^{ref} points at 19.98 for energy and 3 for position error within the coordinate system. We also set $\text{Th}_1 = 0.25$ and $\text{Th}_2 = 0.15$ in the penalty function (see Fig. 11).

2) Results: As shown in Fig. 12, CURE continuously outperforms the baselines in terms of HV [see Fig. 12(a)], PF [see Fig. 12(b)], efficiency [see Fig. 12(c)], penalty response [see Fig. 12(d)], and violations and failures [see Fig. 12(e)] for each severity changes. Specifically, compared to MOBO, CURE finds a configuration with $1.5\times$ higher HV in STR setting (low severity), and $2\times$ higher HV when we change the platform in addition to sim-to-real (high severity). Moreover, CURE achieved efficiency gains of $2.2\times$, and $4.6\times$ over MOBO with low and high severity of deployment changes, respectively. To provide insights into the factors contributing to CURE's enhanced performance, we compared constraint violation θ_V and task failure $\mathcal{T}_F$, revealing reductions of 48% in θ_V, while also demonstrating 28% lower $\mathcal{T}_F$ under high severity changes compared to RidgeCV-MOBO. Therefore, we conclude that CURE performs better compared to the baseline methods as the deployment changes become more severe.

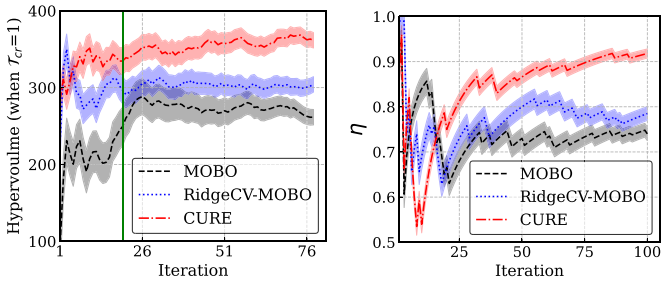
VI. Performance and Sensitivity Analysis of CURE

To explain CURE's advantages over other methods, we conducted a case study employing the same experimental setup described in Section V-C. We also demonstrate CURE's sensitivity by varying the top K values. Our key findings are discussed in the following.



(a) MOBO (b) RidgeCV-MOBO (c) CURE

Fig. 9. CURE demonstrates a denser surface response near the PF and achieved higher $\mathcal{T}_{cr}$ in fewer iterations for the navigation task, resulting in better budget utilization compared to baselines. (a) MOBO. (b) RidgeCV-MOBO. (c) CURE.



(a) (b)

Fig. 10. Effectiveness of CURE and baseline methods for the manipulation task. (a) HV. (b) Efficiency.

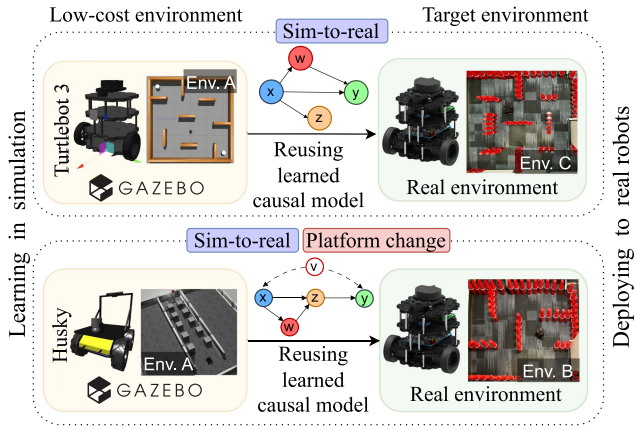


Fig. 11. STR: applying the knowledge of the learned causal model using *Turtlebot 3* in simulation to the *Turtlebot 3* physical robot. STR and PC: transferring the causal model learned using *Husky* in simulation to the *Turtlebot 3* physical robot.

A. CURE's Efficient Budget Utilization Is Attributed to a Comprehensive Evaluation of the Core Configuration Options

For a more comprehensive understanding of the optimization process, we visually illustrate the response surfaces of three pairs of options, each with varying degrees of ACE in energy. Fig. 13(b) contains options with high ACE values, while Fig. 13(d) contains only options with lower ACE values. Options with ACE values close to the median are presented in Fig. 13(c). We observe that response surfaces with higher ACE values are more complex compared to those with lower ACE values. Fig. 13(b)–(d) also shows that CURE explored a range of configurations within the range by systematically varying configurations associated with higher ACE values than those associated with lower ones. In particular, because they have the lowest ACE, the pair of options involving trans_stopped_vel and max_scaling_factor was not considered by CURE in the optimization process, avoiding allocating the budget to less effective options. In contrast, both MOBO and RidgeCV-MOBO wasted the budget exploring less effective options [see Fig. 13(d)]. Note that the option pair involving Min_vel_x and scalling_speed in Fig. 13(b), which exhibits the highest ACE, was not identified by RidgeCV-MOBO. We also observe that due to having a larger search space (entire configuration space), MOBO struggled to explore regions effectively (exhibits a more denser data distribution) compared to CURE. In our previous study [4], we evaluated the accuracy of the key configuration options identified using causal inference through a comprehensive empirical study. Therefore, CURE strategically prioritize core configuration options with high ACE values, ensuring efficient budget utilization and demonstrating a better understanding of such complex behavior, while bypassing less effective options.

B. CURE Leverages the Knowledge Derived From the Causal Model Learned on the Source Platform

In Fig. 13(a), we compare the adjacency matrix between causal graphs learned from the source and target platforms, respectively. We compute the adjacency matrix A from a causal graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, as follows:

$$A[i][j] = \begin{cases} 1, & \text{if } (i,j) \in E \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

where (i, j) represents the edge from vertex i to vertex j. In particular, both causal graphs share a significant overlap, providing a rationale for CURE's enhance performance when transferring the causal model learned from a source (e.g., *Husky* in simulation) to a target (e.g., *Turtlebot 3* physical platform). Therefore, a causal model developed on one platform or environment can be leveraged as prior knowledge on another,

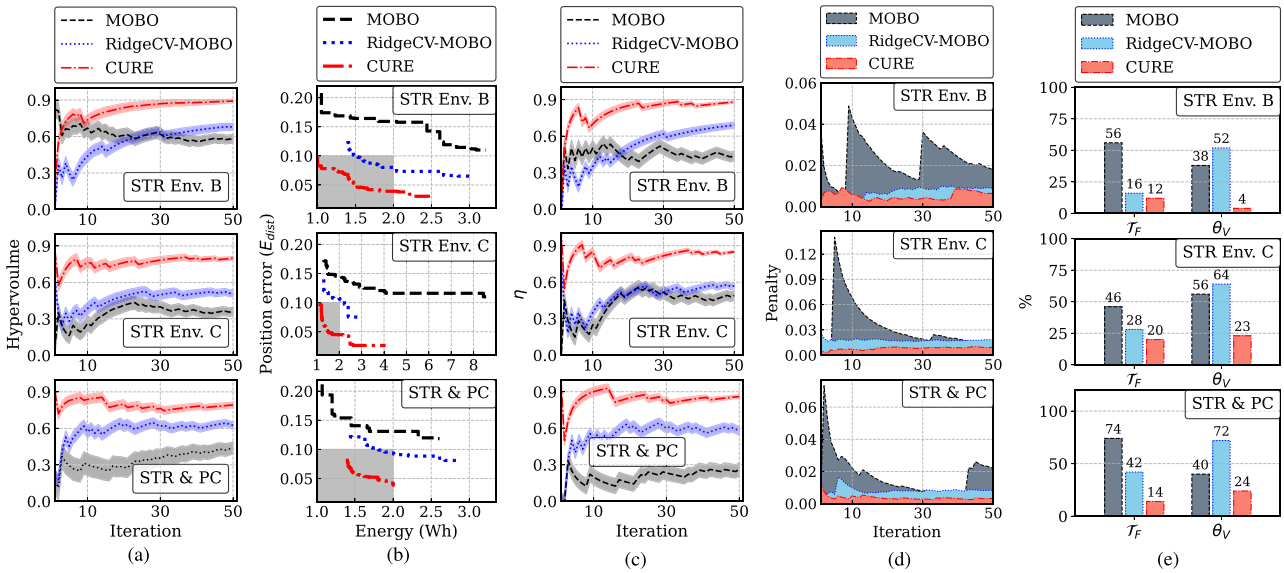


Fig. 12. Transferability of CURE and baseline methods for the navigation task: (a) HV; (b) PF; (c) Efficiency; (d) safety penalty response; and (e) θ_V and $\mathcal{T}_F$; under varying severity of deployment changes.

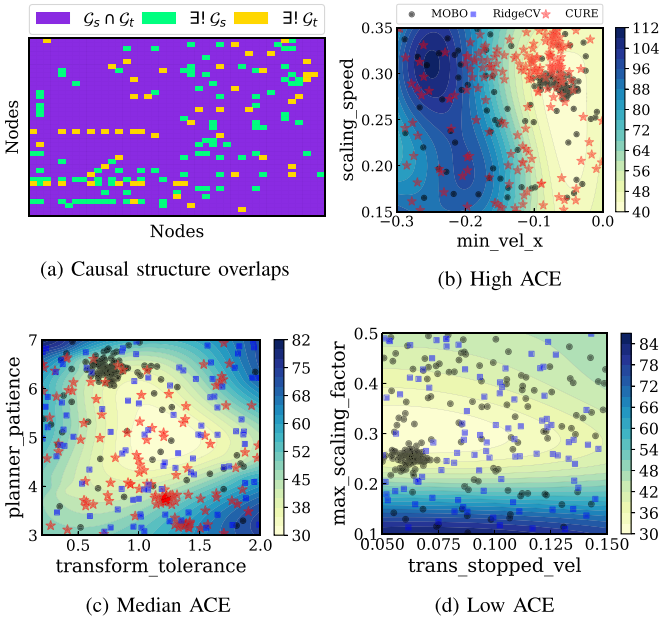


Fig. 13. (a) Significant overlap between causal structures (common edges are represented as purple squares) developed in *Husky* ($\mathcal{G}_s$) and *Turtlebot 3* ($\mathcal{G}_t$). Unique edges are represented as green and yellow squares in $\mathcal{G}_s$ and $\mathcal{G}_t$, respectively. (b), (c), and (d) Contour plot with options of different causal effects. The color bar indicates the energy values, where lower values indicate better performance.

demonstrating the cross-platform applicability and usefulness of the acquired causal understanding.

C. How Sensitive Is CURE When the Value of Top K Varies?

We investigate CURE's performance with different K values and how it affects the optimization process. We conduct

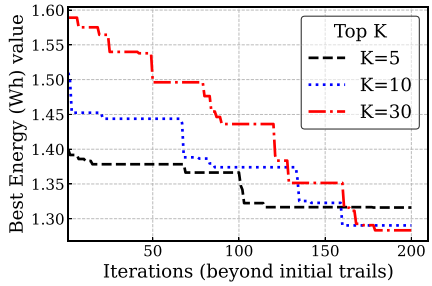


Fig. 14. Sensitivity of CURE under different top K values.

a single-objective optimization on the *Turtlebot 3* platform to demonstrate the sensitivity of CURE. As shown in Fig. 14, there is a tradeoff between the top K values and the iterations required to achieve high-quality solutions. Smaller K values allow the optimization process to quickly find low energy values but may limit exploration, leading to early plateauing. Conversely, larger K values enable more extensive exploration, leading to more gradual improvements and potentially better solutions, but requiring more iterations. This is because, when the search space is smaller, the optimization process can exploit known good areas more effectively. In contrast, a larger search space requires more exploration, which extends the optimization process. One approach for selecting K is to define a threshold on the ACE values and select options that exceed this threshold. This can be done by using a threshold defined as $\{X \mid X_{\text{ACE}} > \mu_{\text{ACE}} + \sigma_{\text{ACE}}\}$, where μ_{ACE} is the mean and σ_{ACE} is the standard deviation of the ACE values. Alternatively, a threshold based on the percentile of ACE values can be employed, such as selecting options with ACE values greater than the 75th percentile. We leave this selection up to the practitioner as user preferences may vary depending on the task, environment, and robotic system.

VII. Discussion

A. Usability of CURE

The design we have proposed is general and extendable to other robotic systems but would require some engineering effort. In particular, to apply CURE to a novel problem, the practitioner must identify the following:

1) configuration options;
2) performance metrics;
3) key performance indicators (KPIs).

Note that the abstraction level of the variables in the causal model depends on the practitioner and can go all the way down to the hardware level. In defining the metrics and KPIs, guidelines provided by the National Institute of Standards and Technology can be used [60], [61]. These guidelines help classify variables as nonmanipulable in the three-layer causal model design [4], which simplifies the performance modeling process by allowing a clear distinction between configurable and performance variables. Moreover, we provide various performance metrics and performance objectives for mobile robot navigation and robot manipulation tasks in Section V.

B. Limitations

1) Causal Model Error: The NP-hard complexity of causal discovery introduces a challenge [62], implying that the identified causal model may not always represent the ground-truth causal relationships among variables. It is crucial to recognize the potential for discrepancies between the causal structure discovered and the actual structures. However, such causal models can still be employed to achieve better performance compared to ML-based approaches in systems optimization [63] and debugging tasks [4], because causal models avoid capturing spurious correlations [45].

2) Potential Biases When Transferring the Causal Model: Caution must be exercised when reusing the entire causal graph learned from the source platform, as differences between causal graphs in the two platforms [as indicated by the green and yellow squares in Fig. 13(a), representing edges unique to the source and target, respectively] can induce bias. It is crucial to discover new causal connections [indicated by the yellow squares in Fig. 13(a)] on the target platform based on observations. Given the small number of edges to be discovered, this task can easily be accomplished with a limited number of observational samples from the target platform.

C. Future Directions

1) Incorporating Causal Gaussian Process (CGP): Using CGP in the optimization process has the potential to capture the behavior of the performance objective better compared to traditional GP [64]. Unlike GP, CGP represents the mean using interventional estimates via do-calculus. This characteristic renders CGP particularly useful in scenarios with a limited amount of observational data or in areas where observational data is not available.

2) Updating the Causal Model at Run-Time: There is potential in employing an active learning mechanism that combines the source causal model $\mathcal{G}_s$ with a new causal model $\mathcal{G}_t$ learned from a small number of samples from the target platform. This approach is particularly promising considering the limitations discussed In Section VII-B.

3) Dynamically selecting top K at Run-Time: In our framework, K is a hyperparameter and its value is defined by the practitioner. Motivated by Fig. 14, there is potential for implementing a dynamic selection approach. This approach would start with a lower K and progressively increase the K if the objective reaches a plateau.

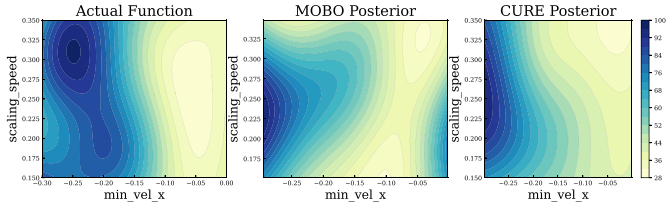


Fig. 15. CURE demonstrates a better understanding about the performance behavior compared to MOBO. The actual function was derived from 1000 observational samples. The color bar indicates energy values.

VIII. Conclusion

In this article, we presented CURE, a multiobjective optimization method that identified optimal configurations for robotic systems. CURE converged faster than the baseline methods and demonstrated effective transferability from simulation to real robots, and even to new untrained platforms. CURE constructs a causal model based on observational data collected from a source environment, typically a low-cost setting such as the *Gazebo* simulator. We then estimate the causal effects of configuration options on performance objectives, reducing the search space by eliminating configuration options that have negligible causal effects. Finally, CURE employs traditional Bayesian optimization in the target environment, but confines it to the reduced search space, thus efficiently identifying the optimal configuration. Empirically, we have demonstrated that CURE not only finds the optimal configuration faster than the baseline methods, but the causal models learned in simulation accelerate optimization in real robots. Moreover, our evaluation shows the learned causal model is transferable across similar but different settings, encompassing different environments, mission/tasks, and new robotic systems.

Appendix A
Additional Details

A. Background and Definitions

1) Configuration Space $\mathcal{X}$: Consider X_i as the ith configuration option of a robot, which can be assigned a range of values (e.g., categorical, Boolean, and numerical). The configuration space $\mathcal{X}$ is a Cartesian product of all options and a configuration $\mathbf{x} \in \mathcal{X}$ in which all options are assigned specific values within the permitted range for each option. Formally, we define the following.

1) Configuration option: $X_1, X_2, \ldots, X_d$.

TABLE I
CONFIGURATION OPTIONS IN MOVE BASE

Configuration Options	Option Values/Range	
	Husky	Turtlebot 3
controller_frequency	3.0 - 7.0	5.0 - 15.0
planner_patience	3.0 - 7.0	3.0 - 7.0
controller_patience	3.0 - 7.0	10.0 - 20.0
conservative_reset_dist	1.0 - 5.0	1.0 - 5.0
planner_frequency	0.0	5.0
oscillation_timeout	5.0	3.0
oscillation_distance	0.5	0.2

TABLE II
CONFIGURATION OPTIONS IN COSTMAP COMMON

Configuration Options	Option Values/Range	
	Husky	Turtlebot 3
publish_frequency	1.0 - 6.0	5.0 - 20.0
resolution	0.02 - 0.15	0.02 - 0.15
transform_tolerance	0.2 - 2.0	0.2 - 2.0
update_frequency	1.0 - 6.0	5.0 - 20.0

TABLE III
CONFIGURATION OPTIONS IN COSTMAP COMMON INFLATION

Configuration Options	Option Values/Range	
	Husky	Turtlebot 3
cost_scaling_factor	1.0 - 20.0	3.0 - 20.0
inflation_radius	0.3 - 1.5	0.3 - 2.0

TABLE IV
CONFIGURATION OPTIONS IN DWAPLANNERROS

Configuration Options	Option Values/Range	
	Husky	Turtlebot 3
acc_lim_theta	1.5 - 5.2	2.0 - 4.5
acc_lim_trans	0.1 - 0.5	0.05 - 0.3
acc_lim_x	1.0 - 5.0	1.5 - 4.0
acc_lim_y	0.0	0.0
angular_sim_granularity	0.1	0.1
forward_point_distance	0.225 - 0.725	0.225 - 0.525
goal_distance_bias	5.0 - 40.0	10.0 - 40.0
max_scaling_factor	0.1 - 0.5	0.1 - 0.4
max_vel_theta	0.5 - 2.0	1.5 - 4.0
max_vel_trans	0.3 - 0.75	0.15 - 0.4
max_vel_x	0.3 - 0.75	0.15 - 0.4
max_vel_y	0.0	0.0
min_vel_theta	1.5 - 3.0	0.5 - 2.5
min_vel_trans	0.1 - 0.2	0.08 - 0.22
min_vel_x	-0.3 - 0.0	-0.3 - 0.0
min_vel_y	0.0	0.0
occdist_scale	0.05 - 0.5	0.01 - 0.15
oscillation_reset_angle	0.1 - 0.5	0.1 - 0.5
oscillation_reset_dist	0.25	0.25
path_distance_bias	10.0 - 50.0	20.0 - 45.0
scaling_speed	0.15 - 0.35	0.15 - 0.35
sim_granularity	0.015 - 0.045	0.015 - 0.045
sim_time	0.5 - 3.5	0.5 - 2.5
stop_time_buffer	0.1 - 1.5	0.1 - 1.5
theta_stopped_vel	0.05 - 0.15	0.05 - 0.15
trans_stopped_vel	0.05 - 0.15	0.05 - 0.15
twirling_scale	0.0	0.0
vth_samples	10 - 30	20 - 50
vx_samples	3 - 10	10 - 30
vy_samples	0 - 15	0 - 5
xy_goal_tolerance	0.2	0.08
yaw_goal_tolerance	0.1	0.17

2) Option value: $x_1, \ldots, x_d$.
3) Configuration: $\mathbf{x} = \langle X_1 = x_1, \ldots, X_d = x_d \rangle$.
4) Configuration space: $\mathcal{X} = \text{Dom}(X_1) \times \cdots \times \text{Dom}(X_d)$.

2) Partial Ancestral Graph: Each edge in the PAG denotes the ancestral connections between the vertices. A PAG is composed of the following types of edges.

1) $A \longrightarrow B$: The vertex A causes B.
2) $A \longleftrightarrow B$: There are unmeasured confounders between the vertices A and B.
3) $A \circ\!\!\rightarrow B$: A causes B, or there are unmeasured confounders that cause both A and B.
4) $A \circ\!\!-\!\!\circ B$: A causes B, or B causes A, or there are unmeasured confounders that cause both A and B.

For a comprehensive theoretical foundation on these ideas, we refer the reader to [47], [65], [66]

3) Causal Model $\mathcal{G}$: A causal model is an ADMG [67], [68], which encodes performance variables, functional nodes (which defines 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. An ADMG is defined as a finite collection of vertices, denoted by V, and directed edges $E_{\mathcal{d}}$ (ordered pairs $E_{\mathcal{d}} \subset V \times V$, such that $(v, v) \notin E_{\mathcal{d}}$ for any vertex v), together with bidirected edges, denoted by $E_{\mathcal{b}}$ (unordered pairs of elements of V). If $(v, w) \in E_{\mathcal{b}}$ then $v \leftrightarrow w$, and if in addition $(v, w) \in E_{\mathcal{d}}$ then $v \leftrightarrow^{\rightarrow} w$.

4) Causal Paths $P_{X \rightsquigarrow Y}$: We define $P = \langle v_0, v_1, \ldots, v_n \rangle$ so that the following conditions hold.

1) v_o is the root cause of the functional fault and $v_n = y_F$.
2) $\forall \ 0 \leq i \leq n, \ \ v_i \in V$ and $\forall \ 0 \leq i \leq n, \ \ (v_i, v_{i+1}) \in (E_{\mathcal{d}} \vee E_{\mathcal{b}})$.
3) $\forall\, 0 \leq i \leq j \leq n$, v_i is a counterfactual cause of v_j.
4) $|P|$ is maximized.

5) Why Do Robotic Systems Fail?: A robotic system may fail to perform a specific task or deteriorate from the desired performance due to the following.

1) *Hardware faults:* Physical faults of the robot's equipment (e.g., faulty controller).

TABLE V
CONFIGURATION OPTIONS IN MOVEIT CHMOP PLANNING

Configuration options	Option Values/Range
planning_time_limit	1.0 - 10.0
max_iterations	1 - 500
max_iterations_after_collision_free	1 - 10
smoothness_cost_weight	0.05 - 5.0
obstacle_cost_weight	0.0 - 1.0
learning_rate	0.001 - 0.5
smoothness_cost_velocity	0.0 - 10.0
smoothness_cost_acceleration	0.0 - 10.0
smoothness_cost_jerk	0.0 - 10.0
ridge_factor	0.0 - 0.01
use_pseudo_inverse	True, False
pseudo_inverse_ridge_factor	0.00001 - 0.001
joint_update_limit	0.05 - 5.0
collision_clearance	0.05 - 2.0
collision_threshold	0.01 - 0.15
use_stochastic_descent	True, False
enable_failure_recovery	True, False
max_recovery_attempts	0 - 10

TABLE VI
ACE VALUES OF THE CONFIGURATION OPTIONS

Configuration Options	ACE			
	Energy	Positional error	$\mathcal{T}_{cr}$	Safety
scaling_speed	199.349	65.980	0.454	0.633
min_vel_x	115.496	18.764	0.566	0.217
controller_frequency	25.370	3.695	0.050	0.019
publish_frequency	19.598	6.026	0.223	0.016
sim_time	15.570	4.680	0.110	0.062
acc_lim_x	12.589	3.050	0.013	0.016
stop_time_buffer	12.130	3.292	0.069	0.011
inflation_radius	11.267	3.663	0.049	0.017
path_distance_bias	10.550	2.559	0.033	0.021
max_vel_theta	10.507	0.394	0.026	0.062
update_frequency	9.250	3.362	0.118	0.019
vth_samples	8.599	2.083	0.028	0.021
cost_scaling_factor	8.565	1.092	0.014	0.021
min_vel_theta	8.152	0.049	0.027	0.016
conservative_reset_dist	7.693	2.808	0.025	0.014
planner_patience	7.532	2.656	0.022	0.069
transform_tolerance	7.103	3.892	0.148	0.038
vy_samples	5.614	2.107	0.021	0.017
goal_distance_bias	5.159	1.365	0.028	0.013
vx_samples	4.901	0.847	0.088	0.014
forward_point_distance	4.877	1.100	0.032	0.006
controller_patience	4.116	4.613	0.031	0.016
acc_lim_theta	4.101	1.835	0.043	0.006
occdist_scale	2.803	0.804	0.035	0.000
acc_lim_trans	2.349	0.818	0.015	0.003
max_vel_trans	2.080	0.307	0.007	0.000
oscillation_reset_angle	1.791	0.715	0.028	0.007
max_vel_x	1.150	0.057	0.000	0.003
min_vel_trans	0.948	0.146	0.002	0.000
resolution	0.188	0.266	0.010	0.001
sim_granularity	0.114	0.000	0.001	0.000
trans_stopped_vel	0.106	0.042	0.002	0.000
max_scaling_factor	0.059	0.062	0.021	0.005
theta_stopped_vel	0.000	0.000	0.000	0.002

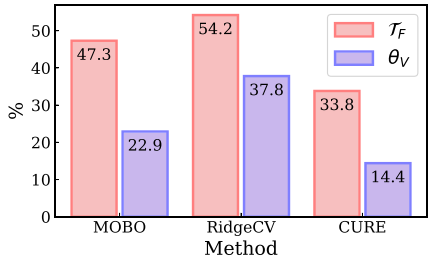


Fig. 16. θ_V and $\mathcal{T}_F$ for RQ1.

2) *Software faults:* Faulty algorithms and/or faulty implementations of correct algorithms (e.g., incorrect cognitive behavior of the robot).
3) *Interaction faults:* Failures that result from uncertainties in their environments.

The software stack is typically composed of multiple components (e.g., localization and navigation), each with a plethora of configuration options (different planner algorithms and/or parameters in the same planner algorithm). Similarly to software components, hardware components also have numerous configuration options. Incorrect configurations can cause a functional fault (the robot cannot perform a task successfully) or a non-functional fault (the robot may be able to finish tasks, but with undesired performance).

6) Nonfunctional Fault: The nonfunctional faults (interchangeably used as *performance faults*) refer to cases where the robot can perform the specified task but cannot meet the specified performance requirements of the task specification. For example, the robot reached the target location(s); however, it consumed more energy. We define the nonfunctional property $\mathcal{N} = \{p_1, \ldots, p_n\}$, where $p_1, \ldots, p_n$ represents different nonfunctional properties of the robotic system (e.g., energy, mission time) and p_j is the value of jth $\mathcal{N}$. The specified performance goal is denoted as p_{js}. Performance failure occurs when $p_j \not\models p_{js}$. Extending the previous scenario, let E^i be the energy consumption during task i and let T^i be the mission completion time. The specified performance goals for the task are indicated as $E_{s->t} <= \text{en}, T_{s->t} <= \text{tt}$, respectively. A nonfunctional fault can be defined as $N_F = (E^i > \text{en}) \vee (T^i > \text{tt})$.

B. Additional Details About Experimental Setup

1) Configuration Options in ROS Nav Core *and* Moveit*:* Table I–IV show the configuration space for each component in the ROS navigation stack and Table V shows the configuration space in *Moveit chomp planning* used in our experiments. We fixed the goal tolerance parameters (xy_goal_tolerance and yaw_goal_tolerance) to determine if a target was reached. Complex interactions between options (intra or inter components) give rise to a combinatorially large configuration space.

C. Additional Details for Evaluation

1) RQ1 Additional Details: We also compared θ_V and $\mathcal{T}_F$, revealing reductions of 8.5% in θ_V, while also demonstrating lower 13.5% $\mathcal{T}_F$ compared to MOBO as shown in Fig. 16.

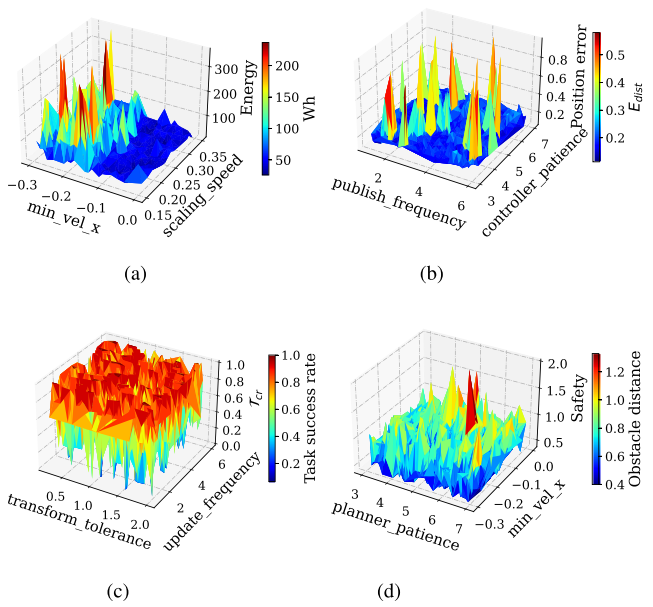


Fig. 17. Pairwise interactions between high ACE configuration options, performance objectives, and constraints, derived from observational data. (a) Interaction with energy. (b) Interaction with position error. (c) Interaction with $\mathcal{T}_{\text{cr}}$. (d) Interaction with safety constraint.

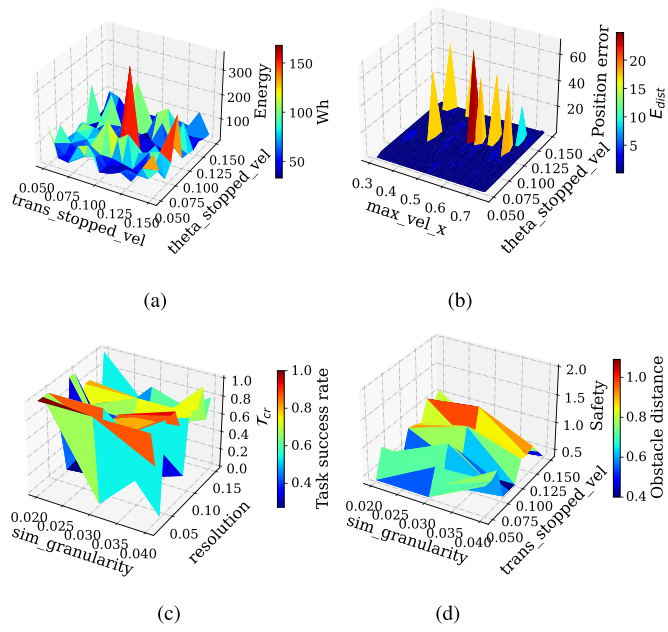


Fig. 18. Pairwise interactions between low ACE configuration options, performance objectives, and constraints, derived from observational data. (c) Interaction with $\mathcal{T}_{\text{cr}}$. (d) Interaction with safety constraint.

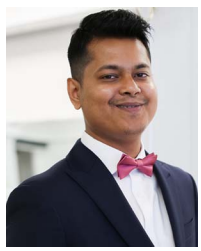
2) ACE Values of Configuration Options: Table VI shows the corresponding ACE values of the configuration options on the performance objectives and constraints. We set the top $K = 5$, represented by blue. Note that CURE reduces the search space from 34 configuration options to 10 by eliminating configuration options that do not affect the performance objective causally.

3) Observational Data Additional Details: In Figs. 17 and 18, we visualize the interactions between core configuration options (pairwise) and their influence on the energy, position error, task success rate, and the safety constraint from the observational data. We observe that the surface response of configuration options with higher ACE values is complex than those with lower ACE values.

REFERENCES

[1] E. Khalastchi and M. Kalech, "On fault detection and diagnosis in robotic systems," *ACM Comput. Surv. (CSUR)*, vol. 51, no. 1, pp. 1–24, 2018.
[2] H. Kim et al., "Patchverif: Discovering faulty patches in robotic vehicles," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 3011–3028.
[3] C. Jung et al., "Swarmbug: Debugging configuration bugs in swarm robotics," in *Proc. 29th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Foundations Softw. Eng.*, 2021, pp. 868–880.
[4] M. A. Hossen et al., "Care: Finding root causes of configuration issues in highly-configurable robots," *IEEE Robot. Autom. Lett.*, vol. 8, no. 7, pp. 4115–4122, Jul. 2023.
[5] K.-T. Xie et al., "Rozz: Property-based fuzzing for robotic programs in ROS," in *Proc. Int. Conf. Robot. Autom.*, 2022, pp. 6786–6792.
[6] T. Kim et al., "Rvfuzzer: Finding input validation bugs in robotic vehicles through control-guided testing," in *Proc. USENIX Secur. Symp.*, 2019, pp. 425–442.
[7] D. Wang et al., "An exploratory study of autopilot software bugs in unmanned aerial vehicles," in *Proc. 29th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Foundations Softw. Eng.*, 2021, pp. 20–31.
[8] J. Garcia et al., "A comprehensive study of autonomous vehicle bugs," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, 2020, pp. 385–396.
[9] S. Kim et al., "Drivefuzz: Discovering autonomous driving bugs through driving quality-guided fuzzing," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 1753–1767.
[10] P. Valle et al., "Automated misconfiguration repair of configurable cyber-physical systems with search: An industrial case study on elevator dispatching algorithms," in *Proc. IEEE/ACM 45th Int. Conf. Softw. Eng.: Softw. Eng. Practice*, Melbourne, Australia, 2023, pp. 396–408.
[11] "Challenges in optimizing configurations for robotic systems in real-world scenarios," 2023. Accessed: Mar. 18, 2025. [Online]. Available: https://github.com/softsys4ai/cure/wiki/Real-World-Optimization-Issues
[12] "Automatic parameter tuning feature request in ROS-2 navigation," 2019. Accessed: Mar. 18, 2025. [Online]. Available: https://github.com/ros-planning/navigation2/issues/1276
[13] "Local planner performance problem. in ROS-2 navigation," 2021. Accessed: Mar. 18, 2025. [Online]. Available: https://github.com/ros-planning/navigation2/issues/2439
[14] Meta, "Ax: Adaptive experimentation platform," 2019. Accessed: Mar. 18, 2025. [Online]. Available: https://ax.dev/docs/why-ax.html
[15] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.
[16] F. Pedregosa et al., "Scikitlearn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeCV.html
[17] A. Binch et al., "Context dependant iterative parameter optimisation for robust robot navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 3937–3943.
[18] A. Zhou et al., "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 32–49, 2011.
[19] D. Fox et al., "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, Jan. 1997.
[20] R. Ariizumi et al., "Multiobjective optimization based on expensive robotic experiments under heteroscedastic noise," *IEEE Trans. Robot.*, vol. 33, no. 2, pp. 468–483, Feb. 2016.
[21] F. Berkenkamp et al., "Bayesian optimization with safety constraints: Safe and automatic parameter tuning in robotics," *Mach. Learn.*, vol. 112, no. 10, pp. 3713–3747, 2023.
[22] B. D. Argall et al., "A survey of robot learning from demonstration," *Robot. Auton. Syst.*, vol. 57, no. 5, pp. 469–483, 2009.
[23] N. Pérez-Higueras et al., "Learning human-aware path planning with fully convolutional networks," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 5897–5902.
[24] M. Pfeiffer et al., "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 1527–1533.

[25] G. Kahn et al., "Badgr: An autonomous self-supervised learning-based navigation system," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 1312–1319, Feb. 2021.
[26] T. Chen and M. Li, "Do performance aspirations matter for guiding software configuration tuning? An empirical investigation under dual performance objectives," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 3, pp. 1–41 2023.
[27] N. Siegmund et al., "Performance-influence models for highly configurable systems," in *Proc. 10th Joint Meeting Foundations Softw. Eng.*, 2015, pp. 284–294.
[28] H. Ha and H. Zhang, "Performance-influence model for highly configurable software with Fourier learning and Lasso regression," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2019, pp. 470–480.
[29] A. Rai et al., "Using simulation to improve sample-efficiency of Bayesian optimization for bipedal robots," *J. Mach. Learn. Res.*, vol. 20, no. 1, pp. 1844–1867, 2019.
[30] R. Kaushik et al., "SafeAPT: Safe simulation-to-real robot learning using diverse policies learned in simulation," *IEEE Robot. Autom. Lett.*, vol. 7, no. 3, pp. 6838–6845, Mar. 2022.
[31] P. Valov et al., "Transferring Pareto frontiers across heterogeneous hardware environments," in *Proc. ACM/SPEC Int. Conf. Perform. Eng.*, 2020, pp. 12–23.
[32] M. S. Iqbal et al., "Cameo: A causal transfer learning approach for performance optimization of configurable computer systems," in *Proc. ACM Symp. Cloud Comput.*, 2023, pp. 555–571.
[33] C. Zhou et al., "Examining and combating spurious features under distribution shift," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 12857–12867.
[34] J. Pearl, *Causality*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
[35] P. Spirtes et al., *Causation, Prediction, and Search*. Cambridge, MA, USA: MIT Press, 2000.
[36] A. Fariha et al., "Causality-guided adaptive interventional debugging," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 431–446.
[37] B. Johnson et al., "Causal testing: Understanding defects' root causes," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, 2020, pp. 87–99.
[38] C. Dubslaff et al., "Causality in configurable software systems," in *Proc. 44th Int. Conf. Softw. Eng.*, 2022, pp. 325–337.
[39] Y. Küçük et al., "Improving fault localization by integrating value and predicate based causal inference techniques," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng.*, 2021, pp. 649–660.
[40] M. Diehl and K. Ramirez-Amaro, "Why did i fail? a causal-based method to find explanations for robot failures," *IEEE Robot. Autom. Lett.*, vol. 7, no. 4, pp. 8925–8932, Apr. 2022.
[41] R. B. Abdessalem et al., "Automated repair of feature interaction failures in automated driving systems," in *Proc. 29th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2020, pp. 88–100.
[42] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.
[43] A. Gupta et al., "Robot learning in homes: Improving generalization and reducing dataset bias," *Adv. Neural Inf. Process. Syst.*, vol. 31, pp. 9112–9122, 2018.
[44] "ROS navigation stack," 2020. Accessed: Mar. 18, 2025. [Online]. Available: http://wiki.ros.org/nav_core
[45] C. Glymour et al., "Review of causal discovery methods based on graphical models," *Front. Genet.*, vol. 10, 2019, doi: 10.3389/fgene.2019.00524.
[46] L. M. Connelly, "Fisher's exact test," *MedSurg Nurs.*, vol. 25, no. 1, pp. 58–60, 2016.
[47] D. Colombo et al., "Learning high-dimensional directed acyclic graphs with latent and selection variables," *Ann. Statist.*, vol. 40, pp. 294–321, 2012.
[48] M. Kocaoglu et al., "Applications of common entropy for causal inference," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, Red Hook, NY, USA: Curran Associates Inc., 2020, Art. no. 1469.
[49] B. Shahriari et al., "Taking the human out of the loop: A review of Bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, Jan. 2015.
[50] J. Knowles, "ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems," *IEEE Trans. Evol. Computation*, vol. 10, no. 1, pp. 50–66, Jan. 2006.
[51] D. Hernández-Lobato et al., "Predictive entropy search for multi-objective Bayesian optimization," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1492–1501.
[52] W. Ponweiser et al., "Multiobjective optimization on a limited budget of evaluations using model-assisted-metric selection," in *Proc. Int. Conf. Parallel Problem Solving From Nature*, 2008, pp. 784–794.
[53] S. Daulton et al., "Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 9851–9864, 2020.
[54] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.
[55] F. Hutter, "Automated configuration of algorithms for solving hard computational problems," Ph.D. dissertation, Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, Canada, 2009.
[56] B. Shahriari et al., "Taking the human out of the loop: A review of Bayesian optimization," in *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, 2016, doi: 10.1109/JPROC.2015.2494218.
[57] M. Görner et al., "MoveIt! task constructor for task-level motion planning," in *Proc. Int. Conf. Robot. Automat.*, 2019, pp. 190–196, doi: 10.1109/ICRA.2019.8793898.
[58] Y. Cao et al., "On using the hypervolume indicator to compare pareto fronts: Applications to multi-criteria optimal experimental design," *J. Statist. Plan. Inference*, vol. 160, pp. 60–74, 2015.
[59] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach," *IEEE Trans. Evol. Computation*, vol. 3, no. 4, pp. 257–271, Apr. 1999.
[60] T. A. Zimmerman, "Metrics and key performance indicators for robotic cybersecurity performance analysis," *NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology*, Gaithersburg, MD, 2019, doi: 10.6028/NIST.IR.8177.
[61] J. Falco et al., "Performance metrics and test methods for robotic hands," *Special Publication (NIST SP), National Institute of Standards and Technology*, Gaithersburg, MD, 2018, doi: 10.6028/NIST.SP.1227-draft.
[62] M. Chickering et al., "Large-sample learning of Bayesian networks is NP-hard," *J. Mach. Learn. Res.*, vol. 5, pp. 1287–1330, 2004.
[63] C. Dubslaff et al., "Causality in configurable software systems," in *Proc. 44th Int. Conf. Softw. Eng.*, 2022, pp. 325–337.
[64] V. Aglietti et al., "Causal Bayesian optimization," in *Proc. 23rd Int. Conf. Artif. Intell. Statist.*, Aug. 2020, pp. 3155–3164.
[65] D. Colombo and M. H. Maathuis, "Order-independent constraint-based causal structure learning," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3741–3782, 2014.
[66] J. Pearl et al., *Causal Inference in Statistics: A Primer*. Hoboken, NJ, USA: Wiley, 2016.
[67] T. Richardson and P. Spirtes, "Ancestral graph markov models," *Ann. Statist.*, vol. 30, no. 4, pp. 962–1030, 2002.
[68] R. J. Evans and T. S. Richardson, "Markovian acyclic directed mixed graphs for discrete data," *Ann. Statist.*, vol. 42, no. 4, pp. 1452–1482, 2014.



Md Abir Hossen received the B.S. degree in electrical and electronics engineering from American International University-Bangladesh, Dhaka, Bangladesh, in 2017 and the M.S. degree in electrical engineering from the South Dakota School of Mines and Technology, Rapid City, SD, USA, in 2021. He is currently working toward the Ph.D. degree in computer science with the Artificial Intelligence and Systems Laboratory (AISys), University of South Carolina, Columbia, SC, USA.

His research interests include artificial intelligence and robot learning.

Sonam Kharade received the M.Tech., B.Tech., and Ph.D. degrees in electrical engineering from Veermata Jijabai Technological Institute (VJTI), Mumbai, India, in 2022.

She was a Postdoc with the AISys, University of South Carolina, Columbia, SC, USA, and is currently a Postdoc with Argonne National Laboratory, Lemont, IL, USA. Her research focuses on the development of mathematical frameworks in control theory, incorporating machine learning techniques, and applying them to practical problems across various domains, such as robotics, and power systems.

Jason M. O'Kane (Senior Member, IEEE) received the B.S. degree from Taylor University, Upland, IN, USA, and the M.S. and Ph.D. degrees from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 2007, all in computer science.

He is currently a Professor of Computer Science and Engineering with Texas A&M University, College Station, TX, USA. His research interests include algorithmic robotics, planning under uncertainty, artificial intelligence, computational geometry, and motion planning.

Bradley Schmerl (Senior Member, IEEE) received the Ph.D. degree in computer science from Flinders University, Adelaide, Bedford Park, SA, Australia, in 1997.

He is currently a Principal Systems Scientist with the Institute for Software Research, Carnegie Mellon University, Pittsburgh, PA, USA. His research interests include software architecture, self-adaptive systems, and software engineering tools.

David Garlan (Life Fellow, IEEE) received the Ph.D. degree in computer science from Carnegie Mellon University, Pittsburgh, PA, USA, in 1987.

He is currently a Professor and an Associate Dean of the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA. His research interests include autonomous and self-adaptive systems, software architecture, formal methods, explainability, and cyberphysical systems.

Pooyan Jamshidi received the Ph.D. degree in computer science from Dublin City University, Dublin, Ireland.

He is currently an Assistant Professor with the Department of Computer Science and Engineering, University of South Carolina, Columbia, SC, USA, where he directs the AISys lab. and has completed Postdoctoral Research with Carnegie Mellon University, Pittsburgh, PA, USA, and Imperial College London, London, U.K. He has also worked in the industry; most recently, he was a visiting Researcher with Google, Menlo Park, CA, USA, in 2021. His work integrates various areas such as distributed systems, statistical and causal learning, and robotics, focusing on areas like autonomous systems, AI accelerators, and software/hardware codesign.

Dr. Jamshidi was the recipient of the USC 2022 Breakthrough Stars Award, specializes in developing resilient systems for dynamic environments.