A causal-based approach to explain, predict and prevent failures in robotic tasks

Citation for the original published paper (version of record):
Diehl, M., Ramirez-Amaro, K. (2023). A causal-based approach to explain, predict and prevent failures in robotic tasks. Robotics and Autonomous Systems, 162.
http://dx.doi.org/10.1016/j.robot.2023.104376

Robotics and Autonomous Systems 162 (2023) 104376

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

A causal-based approach to explain, predict and prevent failures in robotic tasks

Maximilian Diehl *, Karinne Ramirez-Amaro

Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, 41296, Sweden

ARTICLE INFO

Article history:
Available online 4 February 2023

Keywords:
Causality in robotics
Failure prediction and prevention
Explainable AI

ABSTRACT

Robots working in human environments need to adapt to unexpected changes to avoid failures. This is an open and complex challenge that requires robots to timely predict and identify the causes of failures in order to prevent them. In this paper, we present a causal-based method that will enable robots to predict when errors are likely to occur and prevent them from happening by executing a corrective action. Our proposed method is able to predict immediate failures and also failures that will occur in the future. The latter type of failure is very challenging, and we call them timely-shifted action failures (e.g., the current action was successful but will negatively affect the success of future actions). First, our method detects the cause–effect relationships between task executions and their consequences by learning a causal Bayesian network (BN). The obtained model is transferred from simulated data to real scenarios to demonstrate the robustness and generalization of the obtained models. Based on the causal BN, the robot can predict if and why the executed action will succeed or not in its current state. Then, we introduce a novel method that finds the closest success state through a contrastive Breadth-First-Search if the current action was predicted to fail. We evaluate our approach for the problem of stacking cubes in two cases; (a) single stacks (stacking one cube) and; (b) multiple stacks (stacking three cubes). In the single-stack case, our method was able to reduce the error rate by 97%. We also show that our approach can scale to capture various actions in one model, allowing us to measure the impact of an imprecise stack of the first cube on the stacking success of the third cube. For these complex situations, our model was able to prevent around 95% of the stacking errors. Thus, demonstrating that our method is able to explain, predict, and prevent execution failures, which even scales to complex scenarios that require an understanding of how the action history impacts future actions.

1. Introduction

Robots that act in human environments have to handle the execution of various tasks, which requires them to adapt their plan execution flexibly to unexpected changes in the environment [1]. Due to the complexity of these environments, we expect failures in various forms and for various reasons [2], one example being execution failures. The ability to explain their own actions [3], particularly when failures have occurred [4,5], is, therefore, an essential skill of such robots. However, diagnosis capabilities are not only crucial for detecting the causes [6] but could also be used to learn from failures and prevent them from happening [7].

Generating explanations is conceptually based on causality methods [8], which are typically implemented through statistical techniques that learn a mapping between possible causes (preconditions) and the action-outcome (effect) [9,10]. First, we need to investigate how robots can utilize prior experience to reason and consequently generate an explanation about what and why an action execution went wrong [11]. For example, if a robot fails to execute the task of stacking a cube on top of another one, it should be able to explain that the execution failed because the upper cube was dropped too far to the left of the lower cube. In our previous work [11], we proposed a causal-based method to produce explanations when a failure was detected based on a causal Bayesian network. Upon failures, explanations were generated by contrastively comparing the variable parametrization associated with the failed activity with its closest parametrization that would have led to successful execution. Therefore, the next challenge is to use the acquired experience to predict failures in order to prevent them.

The causal relations obtained by the Bayesian networks can be used to predict how likely a particular parametrization of causes will produce failures. In this paper, we propose an extension of [11] which makes use of the prediction capabilities of the learned BNs to prevent failures from happening. When the

* Corresponding author.
E-mail addresses: diehlm@chalmers.se (M. Diehl), karinne@chalmers.se (K. Ramirez-Amaro).

https://doi.org/10.1016/j.robot.2023.104376

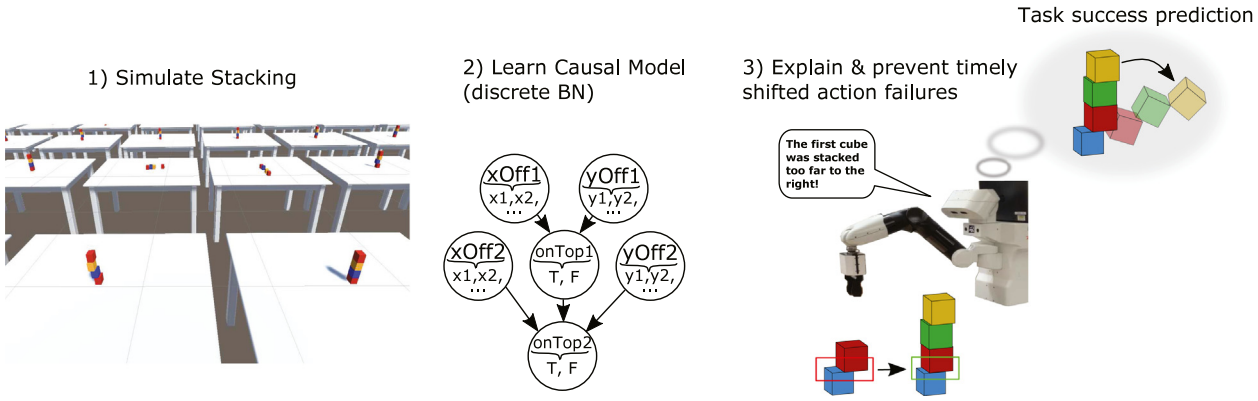


Fig. 1. Depicts our method to allow robots to explain, predict, and prevent failures. First, a causal model is learned from simulations (step 1, 2). Then, this model is used to predict the success of an action given the current state and finds corrective actions in case the action is expected to fail (step 3), even in case of timely shifted action failures. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

prediction of a failure has a high probability given the current state, our method finds an alternative execution state, which is expected to result in a successful action execution. This alternative state is found through Breadth-First-Search (BFS) in a similar fashion as in [11], which allows the agent not only to prevent failures but, at the same time, to provide explanations for its corrective actions.

Predicting and preventing errors is particularly difficult if the effects of an action are not immediately flawed but become problematic in future actions [12]. For example, the error was produced on the first action, but the consequence (a stacking error) is only observed after the third action (in the future). We call these cases timely shifted action errors. In such cases, the models need to consider the history of the previous actions. Fig. 1 depicts the case of a robot building a tower of four cubes. The second (red) cube is not stacked entirely centered with respect to the bottom (blue) cube. Even if this particular stack can be considered successful on its own, it negatively impacts the overall stability of the tower, which might become a problem later after the second or third stack. This challenging problem is also addressed in this paper, and we show that our causal-based method scales to these complex cases by detecting causal links over the history of several actions, effectively predicting and preventing action failures.

To summarize, our contributions are as follows:

- We propose a causal-based method that allows robots to understand possible causes for errors and predict how likely an action will succeed.
- We then introduce a novel method that utilizes these prediction capabilities to find corrective actions which will allow the robot to prevent failures from happening.
- Our algorithm proposes a solution to the complex challenge of timely shifted action effects. By detecting causal links over the history of several actions, the robot can effectively predict and prevent failures, even if the root of a failure lies in a previous action.

2. Related work

In this related work section, we initially review how causality is currently used in the area of robotics, then discuss different types of explainable models that describe cause–effect relations before presenting other methods for failure prevention. Finally, we review the concept of contrastive explanations, which is a vital aspect of our failure explanation and prevention method.

2.1. Causality in robotics

Even though the importance of causality is increasingly acknowledged, it is still an underexplored topic in the robotics community [5,13]. One of its important purposes is the ability to discern task-relevant from irrelevant variables in data. This feature is taken advantage of, for example, in CREST [14] (Causal Reasoning for Efficient Structure Transfer), where causal interventions on environment variables are used to discover which variables affect a Reinforcement Learning (RL) policy. Consequently, excluding irrelevant variables was found to positively affect the generalizability and sim-to-real transfer of the policy. Another application was presented in [15], where a set of task-agnostic learning rules was defined to learn causal relations in a physical task. In particular, through repeated interaction with its environment, a humanoid iCub robot learned a causal relationship between the weight of objects and its ability to increase the water level, while other variables, like color, were found to be irrelevant. Another paper [16] has the objective of learning causal relations between actions in household tasks. From human demonstrations in Virtual Reality, they discovered a causal link between opening a drawer and retrieving plates. A causal approach to tool affordance learning was presented in [13]. Their goal was to equip a robot with the ability to work with new tools more effectively through prior experience with different tools. Also [17] exploits the ability to learn the causal relevance of variables to discover dependencies between objects, actions, constraint features, and the task of grasp selection. Their main objective is to find the best grasp conditional on constraints and object features. The presented approaches and our method have in common that the framework of causality is used to (a) detect causal links between certain preconditions in the environment and the success of an action [14–16] and (b) make use of that knowledge to do something in an optimal fashion (e.g., picking the best tool [13] or grasp [17]). However, those methods have not explored how this causal understanding can be used to explain, predict and prevent failures from happening. Furthermore, none of these works discussed the problem of timely shifted action errors, which we are addressing in this paper.

2.2. Learning explainable models of cause–effect relations

The planning community captures cause–effect relationships in the form of (probabilistic) planning operators [18]. Some works proposed the concept of task execution models, which combines

symbolic preconditions and a function approximation for the success model [9], based on Gaussian Process models. They evaluate their method by learning how to grasp handles. The authors noted that a simulated environment could be incorporated for a faster and more extensive experience acquisition, as proposed in [17], which is a technique we employ to learn our Bayesian Networks. Human virtual demonstrations have been used to construct planning operators to learn cause–effect relationships between actions and observed state-variable changes [18]. Bauer et al. learn probabilistic action effects of dropping objects into different containers [10], with the goal of generalizing the probability predictions for a variety of objects, like bowls and bread boxes. They utilize an ontology to find out how closely related the objects are but do not consider object properties. Our approach has several advantages over the works mentioned in this subsection. While we share the general objective of learning generalizable prediction models that map preconditions to action effects, our proposed causal approach allows us, as opposed to [10], to learn which object properties and environment preconditions are relevant for the investigated actions. This feature is also the basis for our method to explain, predict and prevent failures. Furthermore, our method scales to scenarios where the effect of a previous action has an impact on subsequent actions, as opposed to [9].

2.3. Failure prevention

In [2], approaches for fault detection and diagnosis in Robotic systems are classified into data-driven, model-based, and knowledge-based. The ability to diagnose and correct robot action failures is discussed in [7] for the problem of robot grasping. They proposed a method that diagnoses potential causes (unmet preconditions) for a failure and, through sampling, finds a parametrization that maximizes predicted execution success under the known execution model. This method works well when errors occur immediately after the action. We, however, propose a model that captures the action history in order to prevent errors that either occur in the future or build up cumulatively over several actions. In [12] the authors propose a method that would also be able to detect the cause of failures, even if it is not related to the currently executed action. They utilize Hierarchical Hidden Markov Models (HHMMs) to represent and track failures over time, which allows them to generate a list of possible causes with different likelihoods when a failure is encountered during a plan execution. However, they have not utilized this information to prevent failure through corrective actions, as we propose.

2.4. Contrastive explanations

An important element of many explainable AI methods like Explainable AI Planning (XAIP) [19] is the concept of contrastive explanations. This concept draws parallels to the way that humans generate explanations [4]. Typically the focus of work that falls under the umbrella of XAIP are questions like *why the plan contains a particular action a_1 and not action a_2?* [19,20] or they focus more on the actual communication of plan execution failures [21]. We utilize this concept in a different way and for a different purpose. We search for contrastive failure causes, which allow us to explain why failures might occur in the future and find corrective actions to prevent them from happening.

3. Our approach to explaining, predicting and preventing failures

We propose and present a multi-step approach to predicting and preventing failures, which consists of four main steps:

1. We start by explaining the task of variable identification (Section 3.1). In this step, an action is represented in terms of a set of random variables that describe possible preconditions and action effects.
2. Then, we learn a causal model using the identified variables from step 1 based on BN learning (Section 3.2). BN learning is typically divided into learning the causal connections between the variables (structure learning) and learning conditional probability distributions (parameter learning).
3. In Section 3.3, we elaborate on how we use the obtained causal model to explain failures after they have occurred.
4. Finally, in Section 3.4, we expand our method to address the problem of predicting when a failure is likely to occur and preventing failures.

3.1. Variable definitions and assumptions

Our method for explaining, predicting, and preventing failures is based on detecting causal relations between possible causes and effects of an action.[1] We describe each action in terms of a set of random variables $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$. The choice of the number of (n) variables is up to a human experiment designer. However, there are several aspects that need to be considered: we conceptually split $\mathbf{X}$ into a subset of treatment (cause) variables $C \subset \mathbf{X}$ and outcome (effect) variables $E \subset \mathbf{X}$. Then, the goal is to measure the effect of treatment variables on the action outcome. In other words, C and E differ as we can decide and set values for variables in C, while outcome variables are not actively set but measured at the end or throughout the experiment.

We can collect data for learning causal models either from simulations or from the real world. A data sample d consists of a particular parametrization of the previously defined set of variables $\mathbf{X}$, which we denote as $d = \{X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n\}$, where n denotes the number of variables, and $x_1, \ldots, x_n$ the concrete variable values of that particular parametrization. We currently assume that each collected data sample d is complete, and thus contains a value for all variables in $\mathbf{X}$. However, there are also Bayesian network learning methods that can deal with incomplete datasamples [22]. We sample values for the causes C randomly. Randomized controlled trials are referred to as the gold standard for causal inference [3] and allow us to avoid the danger of unmeasured confounders. Consequently, we can call the detected relations between the variables $\mathbf{X}$ causal and not merely correlations, which can be spurious. Another advantage of causal models is that they can also answer interventional queries.

We describe the success of an action in terms of a set of variables $G \subset E$. Furthermore, $\mathbf{X}_{goal} = \{ d_{goal_1}, d_{goal_2}, \ldots, d_{goal_h} \}$ is a set that contains all possible variable parametrizations that denote a successful action execution, where h denotes the number of possible successful action outcomes. Each goal parametrization $d_{goal_l} \forall l \in \{1, 2, \ldots, h\}$, describes one possible variable assignment of G, that are possible successful action outcomes. Then, an action is successful *iff* its parametrization of goal variables $d_g = \{X_{g_1} = x_{g_1}, X_{g_2} = x_{g_2}, \ldots, X_{g_m} = x_{g_m}\}$ of G, $d_g \in \mathbf{X}_{goal}$, where $X_{g_i} \in G \quad \forall i = \{1, \ldots, m\}$, m denotes the number of variables that are relevant for specifying the success of an action and $x_{g_1}, \ldots, x_{g_m}$ are the concrete variable values of variables in G. Thus d_g is a sub-sample of the corresponding action parametrization d. In general, G does not necessarily need to contain all variables E but depends on the actual goal that one aims to measure, which we will illustrate in the following two examples: let us assume we want to measure the effect of two different cancer treatments $T = \{\text{treat1}, \text{treat2}\}$ on a variable which describes

[1] Note that our method can be applied to actions such as *stacking a cube*, or tasks, such as *stacking several cubes*.

if the patient is cancer-free $F = \{1, 0\}$ after the treatment. In this case, the only outcome variable H is also the one we use to measure the treatment success, thus $G = E = \{H\}$ and $m = 1$. Furthermore we can only define one possible $d_{goal} = \{F = 1\}$, thus $h = 1$. However, let us now assume that our treatments could have two potential side effects $S1$ and $S2$, of which $S1$ might be deadly as well and must therefore be avoided. In this second example not all outcome variables are part of the variables that we use to define the action success $G = \{H, S1\} \in E = \{H, S1, S2\}$ and $d_{goal} = \{F = 1, S1 = 0\}$. Therefore, $m = 2$ and $h = 1$. These two examples showed that the choice of G and d_{goal} is task dependent. It is out of the scope of this paper to learn $\mathbf{X}_{goal}$ and instead we assume it is provided. However, the robot has no a-priori knowledge about which variables in $\mathbf{X} = X_1, X_2, \ldots, X_n$ are in C or E, nor how they are related.

3.2. Our proposed pipeline to learn causal models

A Bayesian Network (BN) is defined as a *directed acyclic graph* (DAG) $\mathcal{G} = (\mathbf{V}, A)$, where $\mathbf{V} = \{X_1, X_2, \ldots, X_n\}$ represents a set of nodes which correspond to the random variables $\mathbf{X}$ that describe our action, and A is the set of arcs [23] that denotes all relations between the variables. This dependency allows to factorize the *joint probability distribution* of a BN into *local probability distributions*, where each random variable X_i only depends on its direct parents Π_{X_i}:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\Pi_{X_i}) \tag{1}$$

To learn a BN from data, we first learn the structure of the DAG and then retrieve the local probability distributions, which is referred to as parameter learning. Many structure learning algorithms cannot handle continuous variables as parents of categorical variables [23,24]. We, therefore, perform quantile discretization [25] on all continuous random variables in $\mathbf{X}$. The range of each variable in $\mathbf{X}$ depends on the collected data samples d. Quantile discretization divides each continuous variable $\mathbf{X}$ into k intervals which contain an approximately similar number of data samples d who fall into these intervals. During this discretization step, we retrieve a list of all variable intervals which we denote as X_{int}. In the current pipeline we choose the number of discretization intervals heuristically. Please refer to [26] for a discussion on more automated discretization methods.

3.2.1. Structure learning

To learn the causal relations $\mathcal{G} = (\mathbf{V}, A)$ between the variables, methods such as the PC[2] [27] algorithm can be used. This algorithm is a *constraint-based* algorithm, and it uses statistical tests to determine conditional independence relations from the data [25]. PC is one of many structure learning algorithms [22], which could be used with equal eligibility for this step. Please refer to [22] for an extensive review of other structure learning algorithms. Note that learning plausible assumptions about causal relations from data is still an active field of research [28]. For example, in some cases, it is challenging to uniquely determine the direction of causal relations without additional interventional experiments, additional domain knowledge, or certain assumptions about the data distribution [29]. Therefore, in the following we assume that the causal relations of $\mathcal{G}$ indeed accurately estimate the data and the underlying physical processes of the action that is represented by $\mathcal{G}$.

[2] The PC algorithm is named after its two authors Peter Spirtes and Clark Glymour.

3.2.2. Parameter learning

The purpose of this step is to fit functions that reflect the *local probability distributions*, of the factorization in formula (1). As we work with discretized variables, the objective is the estimation of all entries of the conditional probability tables $P(X_i|\Pi_{X_i}) \quad \forall i = \{1, \ldots, n\}$. Popular methods for this step include the Maximum Likelihood Estimator (MLE) or bayesian methods such as the Maximum-A-Posteriori (MAP) estimation. In the following, we use MLE to estimate the local probability distributions.

3.3. Our proposed method to explain failures

In our previous work [11], we proposed a method to generate contrastive failure explanations, which uses the obtained causal Bayesian network to compute success predictions (summarized in Alg. 1). The goal of our earlier method was to explain failures after they were committed by the robot. Therefore, one of the inputs of this procedure is the variable parametrization that resulted in the action that will produce a failure x_{failure}, which is defined as a parametrization d, whose sub-sample $d_g \notin \mathbf{X}_{goal}$. Therefore, Alg. 1 finds the closest discretized variable parametrization $x_{\text{solution}_{\text{int}}}$, whose sub-sample $d_g \in \mathbf{X}_{goal}$ and its corresponding success probability prediction p_{solution}.

Algorithm 1 Get closest successful variable parametrization from causal model

Input: failure variable parametrization x_{failure}, structural equations $P(X_i|\Pi_{X_i})$, discretization intervals of all model variables X_{int}, success threshold ϵ, goal parametrizations $\mathbf{X}_{goal}$

Output: solution variable parametrization $x_{\text{solution}_{\text{int}}}$, solution success probability prediction p_{solution}

1: **procedure** GetClosestSuccIntervals($x_{\text{failure}}, P(X_i|\Pi_{X_i}), X_{\text{int}}, \epsilon, \mathbf{X}_{goal}$)
2: $x_{\text{current}_{\text{int}}} \leftarrow$ GetIntervalFromValues($x_{\text{failure}}, X_{\text{int}}$)
3: $P \leftarrow$ GenerateTransitionMatrix(X_{int})
4: $q \leftarrow [x_{\text{current}_{\text{int}}}]$
5: $v \leftarrow []$
6: **while** $q \neq \emptyset$ **do**
7: $node \leftarrow$ Pop(q)
8: $v \leftarrow$ Append($v, node$)
9: **for all** transitions $t \in P(node)$ **do**
10: $child \leftarrow$ Child($P, node, t$)
11: **if** $child \notin q, v$ **then**
12: $p_{\text{solution}} = P(d_g \in \mathbf{X}_{goal}|\Pi_G = child)$
13: **if** $p_{\text{solution}} > \epsilon$ **then**
14: $x_{\text{solution}_{\text{int}}} \leftarrow child$
15: Return($p_{\text{solution}}, x_{\text{solution}_{\text{int}}}$)
16: **else**
17: $q \leftarrow$ Append($q, child$)

After retrieving the current intervals $x_{\text{current}_{\text{int}}}$ from the continuous variable parametrization (GetIntervalFromValues: L-2, Alg. 1), a transition matrix P is generated (GenerateTransition Matrix: L-3, Alg. 1). This transition matrix provides all possible single-interval-change transitions that will be used for the search procedure to find the closest successful variable parametrization. A state in the search tree is made up of a complete parametrization of the parent variables of the goal variables Π_G. Let us consider the example of $\mathbf{X} = \{X_1, X_2\}$ with two intervals x', x'' each. Then, all possible valid transitions for $node = (X_1 = x', X_2 = x')$ would be $child_1 = (X_1 = x', X_2 = x'')$ or $child_2 = (X_1 = x'', X_2 = x')$. In lines 6–17 (Alg. 1), the closest variable parametrization that fulfills the goal criteria of $P(d_g \in \mathbf{X}_{goal}|\Pi_G = child) > \epsilon$, is searched for based on Breadth-First-Search (BFS). ϵ is the success threshold and can be set heuristically. Choosing ϵ is

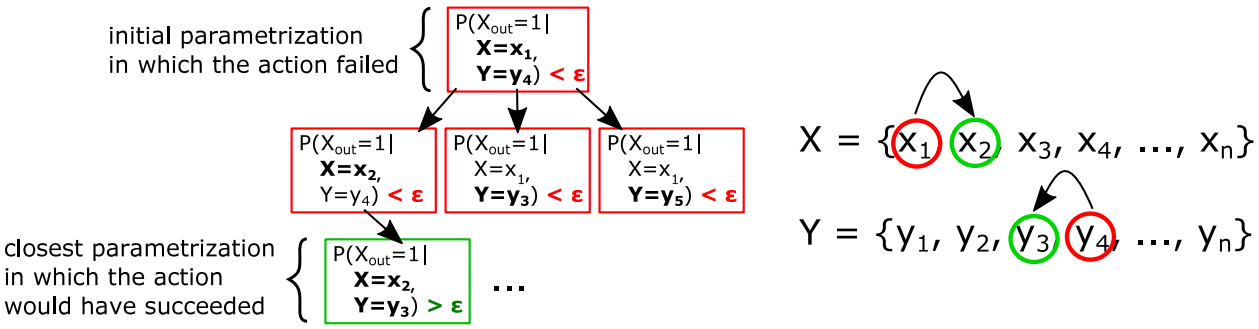


Fig. 2. Exemplifies how contrastive explanations are generated from the BFS search tree.
Source: Figure adapted from [11].

a trade-off between action success accuracy and search time. The search time increases with the value of ϵ, the number of causally relevant parent variables, and the number of discretization intervals per variable. Therefore, one could start with a high ϵ value close to 1.0 and reduce this value in case the processing time is too long for the given application. The function POP (L-7, Alg. 1) takes and removes the first element from the frontier list q and assigns it to the variable *node*. The function CHILD (L-10, Alg. 1) retrieves the variable parametrization from the children of all possible transitions at *node*.

Our approach to retrieve the closest success parametrization is motivated by Occam's razor: the simplest failure explanation is found through the least number of interval changes that are required to transit from unsuccessful to successful execution. Breadth-First-Search is guaranteed to find the shortest distance between a selected node and any goal node in a directed graph in terms of the number of transitions, and we, therefore, use BFS to find $x_{\text{solution}_{\text{int}}}$. In our current approach, every transition is weighted equally as we try to minimize human input. Then, the methodology behind our explanation generation is comparing the current variable intervals that lead to an execution failure $x_{\text{current}_{\text{int}}}$ with the closest intervals that would have been expected to lead to a successful task execution $x_{\text{solution}_{\text{int}}}$.

This process is visualized in Fig. 2, exemplified on two variables X and Y, which both have a causal effect on variable X_{out}. Given that $x_{\text{out}} = 1 \in X_{\text{goal}}$ would denote the successful action that is described through this causal model, the resulting explanation would be that the action has failed because $X = x_1$ instead of $X = x_2$ and $Y = y_4$ instead of $Y = y_3$.

3.4. Our proposed method to predict and prevent failures

In [11], we have used Alg. 1 to explain the reasons for a failure only after the failure has occurred. However, the causal model can also be used to predict the success probability of the current state prior to the actual execution. In this article, we, therefore, propose an extension of our method to prevent failures from happening when an error has been predicted with a high probability. This extension is presented in the new Alg. 2. In particular, we first retrieve the discretization intervals for the current variable parametrization in (GETINTERVALFROMVAL: L-2, Alg. 2) and query the causal model to predict the success probability for the current state (L-3, Alg. 2). In case the predicted probability is above a chosen threshold of ϵ, we continue with the execution based on the current parameters (L-4, Alg. 2). Please refer to Section 3.3 for a discussion on how to heuristically choose ϵ. If, however, the probability is below the threshold (L-5, Alg. 2), we retrieve the closest success parametrization through Alg. 1 (L-7, Alg. 2). Finally, we use the middle values of the corrected intervals as concrete parameters to retrieve a corrected variable parametrization x_{success} which is defined as a variable parametrization d, whose subsample $d_g \in \mathbf{X}_{goal}$ (MIDDLEVALFROMINTERVALS: L-8, Alg. 2). Note the algorithm only changes the parametrization for intervals that have been deemed responsible for the failure (thus variables with interval changes) and keep the current parametrization for variables that are not problematic. The output parametrization x_{success} can then be used to manipulate the environment to ensure the action will succeed.

Algorithm 2 Predict and prevent failures

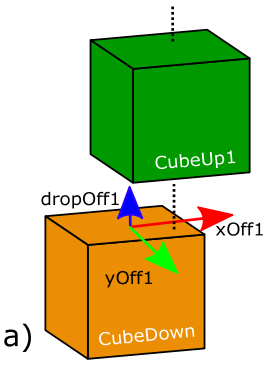
Input: current variable parametrization x_{current}, structural equations $P(X_i|\Pi_{X_i})$, discretization intervals of all model variables X_{int}, success threshold ϵ, goal parametrizations $\mathbf{X}_{goal}$
Output: Concrete success variable parametrization x_{success}

1: **procedure** PREVENTFAILURES($x_{\text{current}}, P(X_i|\Pi_{X_i}), X_{\text{int}}, \epsilon, \mathbf{X}_{goal}$)
2: $x_{\text{solution}_{\text{int}}} \leftarrow$ GETINTERVALFROMVAL($x_{\text{current}}, X_{\text{int}}$)
3: $p_{\text{solution}} = P(d_g \in \mathbf{X}_{goal}|\Pi_G = x_{\text{solution}_{\text{int}}})$
4: $x_{\text{success}} \leftarrow x_{\text{current}}$
5: **if** $p_{\text{solution}} < \epsilon$ **then**
6: $x_{\text{failure}} = x_{\text{current}}$
7: $p_{\text{solution}}, x_{\text{solution}_{\text{int}}} \leftarrow$ GETCLOSESTSUCCINTERVALS($x_{\text{failure}}, P(X_i|\Pi_{X_i}), X_{\text{int}}, \epsilon, \mathbf{X}_{goal}$)
8: $x_{\text{success}} \leftarrow$ MIDDLEVALFROMINTERVALS($x_{\text{solution}_{\text{int}}}, x_{\text{current}}, X_{\text{int}}$)
9: RETURN(x_{success})

4. Experiments

We evaluate our method to predict and prevent execution failures for the problem of stacking cubes. We conducted two different experiments:
Experiment 1: First, we evaluate our learned causal model on a simple action of stacking one cube, see Fig. 3.a. From this experiment, we assess the correction abilities of the obtained causal model (see, Section 4.1).
Experiment 2: Then, we assess our proposed method in a complex task of stacking multiple cubes to build a tower of four cubes. With this experiment, we investigate the case of performing the stacking action three times in a row (see Section 4.2).

For both experiments, we design an environment that contains two types of cubes: *CubeDown* and *CubeUp*$_i$, with i being the stacking order of the upper cubes (e.g., *CubeUp*$_1$ is the cube that is stacked first). *CubeDown* is the bottom cube of a tower that, in our case, does not need to be rearranged or requires any movement prior to the stacking actions. All cubes have a fixed size of 5 cm. To describe the stacking action, we define four types of variables: $\texttt{xOff}_i$, $\texttt{yOff}_i$, $\texttt{dropOff}_i$, $\texttt{onTop}_i$, where



b)

Variable	Meaning
$\texttt{xOff}_1$	$\text{cubeUp}_1.x - \text{cubeDown}.x$ (distance between cube centers)
$\texttt{yOff}_1$	$\text{cubeUp}_1.y - \text{cubeDown}.y$
$\texttt{dropOff}_1$	$(\text{cubeUp}_1.z - \text{cubeDown}.z) - \text{cubeLength}$ (distance between cube surfaces)
$\texttt{onTop}_1$	indicates if cubeUp_1 is on top of CubeDown after the stacking process

Fig. 3. (a) visualizes the variables that are used to describe the stacking action and (b) defines their meaning. *Source:* Figure adapted from [11].

i denotes the corresponding cube. Figs. 3.b and 4.b provide a detailed description of the variables for both experiments.

The data collection for training and evaluating the causal models is conducted in Unity3d, which employs the Nvidia PhysX engine for physics simulations. Inside Unity, we set up 400 parallel table environments to speed up the simulation process and data collection. At the beginning of every stacking experiment, the variable values for $\texttt{xOff}_i$, $\texttt{yOff}_i$, $\texttt{dropOff}_i$ are randomly sampled, and the cube positions are initialized accordingly. Note that the simulations are conducted without the existence of any robot and only involve cubes dropping from a predefined position. As a result, the simulations are less conservative than the real world. However, in [11], we have experimentally determined approximately 70% congruence in terms of stacking success between simulated stacks and stacks that were performed by a real robot.

4.1. Experiment 1 setup: Stack-1-Cube scenario

In our first experiment, the goal is to stack only one single cube on top of the bottom cube (see Fig. 3.a). As the initial step, we deploy a training phase to collect data for learning the causal Bayesian network. For this purpose, we run 40,000 simulations of randomized single-stack actions. We define the set of variables that is used for the first experiment ($E1$) with $i = \{1\}$ as $\mathbf{X}_{E1} = \{\texttt{xOff}_1, \texttt{yOff}_1, \texttt{dropOff}_1\ \texttt{onTop}_1\}$. We sample randomly values for $\texttt{xOff}_1, \texttt{yOff}_1 \sim \mathcal{U}_{[-3.0,3.0]}$ (in cm), $\texttt{dropOff}_1 \sim \mathcal{U}_{[0.5,10]}$ (in cm). $\texttt{onTop}_1 = \{\text{True}, \text{False}\}$ is not sampled but automatically determined after the stacking process. From this training data, we learn the graphical representation of the variables and fit the conditional probability distributions. For each conditional value assignment, we then determine the closest variable parametrization that would lead to a successful execution based on the procedure, which is elaborated in Section 3. This allows us to generate a 'lookup' table for the best possible corrections for each x-y-dropOffset parametrization that we could possibly encounter during a single stack action.

The goal set $G = \{\texttt{onTop}_1\}$ for this experiment describes the stacking success for the cube that is stacked, and consequently, we denote the action as successful *iff* $\texttt{onTop}_1 = \text{True}$. Thus $\mathbf{X}_{goal_{E1}} = \{\texttt{onTop}_1 = \text{True}\}$. We then evaluate the impact of having the obtained correction model in terms of cube-stacking success by comparing two cases (datasets): one without corrections from the model and one including the corrected stacking positions. Both test datasets use the same sample seed, different from the train-dataset seed. However, the variable distributions for training and testing are similar. Our hypothesis for this experiment is that deploying our method to adapt the stacking position prior to dropping the cube will significantly improve the stacking success.

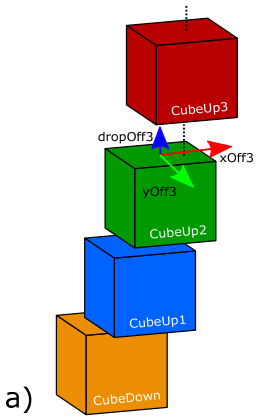
4.2. Experiment 2 setup: Stack-3-Cubes scenario

Our second experiment ($E2$) considers the more complex scenario of stacking three cubes on top of a base cube (see Fig. 4.a). Unlike experiment 1, we need three upper cube variables: $\textit{CubeUp}_1$, $\textit{CubeUp}_2$ and $\textit{CubeUp}_3$ instead of a single upper cube. Now, the goal of the robot is to build a tower of cubes by stacking $\textit{CubeUp}_1$ on top of *CubeDown*, $\textit{CubeUp}_2$ on top of $\textit{CubeUp}_1$ and, finally, $\textit{CubeUp}_3$ on top of $\textit{CubeUp}_2$. Our new set of variables is $\mathbf{X}_{E2} = \{\texttt{xOff}_i, \texttt{yOff}_i, \texttt{dropOff}_i, \texttt{onTop}_i\}$, where $i = \{1, 2, 3\}$. In this case, we expect that the success of each stacking action becomes increasingly difficult the higher the tower of cubes.

To test this second experiment, we implement some slight adaptations to the simulation environment. For example, every 3 s, the next cube is dropped until the whole tower is complete. If a previous stack has failed, the whole experiment is considered a failure, and no more cubes are stacked on top. Therefore, the experiment is terminated. Furthermore, the offset between two cubes is always calculated with respect to the previously stacked cube (e.g., between $\textit{CubeUp}_1$ and *CubeDown* or $\textit{CubeUp}_2$ and $\textit{CubeUp}_1$), as exemplified in Fig. 4.b.

Similarly to the first experiment, we begin with a learning phase. Due to the requirement of the increased samples, we conducted a total of 800,000 experiments. In addition, unlike the uniformly distributed samples in experiment 1, we sample from a Gaussian distribution for this experiment to achieve a more equally distributed ratio between failures and successful stacks. Formally, $\texttt{xOff}_i, \texttt{yOff}_i \sim \mathcal{N}_{[0.0,2.0]}$ (in cm), $\texttt{dropOff}_i \sim \mathcal{N}_{[0.1,3.0]}$ (in cm) for $i = \{1, 2, 3\}$. Again, $\texttt{onTop}_i = \{\text{True}, \text{False}\}$ is not sampled but automatically determined after each stacking action. Also note that samples are limited to the ranges of $\texttt{xOff}_i, \texttt{yOff}_i = [-3.0, 3.0]$ and $\texttt{dropOff}_i = [0.5, 10]$. This data is used to learn two different Bayesian networks: One only considers the first stacking action, thus representing a similar case as in experiment 1 ($i = 1$), only trained on normally distributed data. For this first model, we take a subset of 40,000 samples. The second model, trained on all 800,000 samples, represents all three cubes ($i = 1, 2, 3$) in one graphical representation.

We evaluate this experiment in two ways. First, we learn a BN only on the first stacking action, which we call $1 - \texttt{Stack} - \texttt{Model}_g$ (g denoting that the data was sampled from Gaussian distributions), and we want to evaluate how useful this model is for failure prediction and prevention in later stacks. We do not reuse the model from the first expriment ($1 - \texttt{Stack} - \texttt{Model}_u$, where u = uniform distribution) since we expect some differences due to the adapted sampling distributions (E1: Uniform and E2: Gaussian distributions). Then, we learn a second model that captures all three stacks in one model, which we call the $3 - \texttt{Stack} - \texttt{Model}_g$. Consequently, we



a)

b)

Variable	Meaning
$\texttt{xOff}_3$	$\text{cubeUp}_3.x - \text{cubeUp}_2.x$ (distance between cube centers)
$\texttt{yOff}_3$	$\text{cubeUp}_3.y - \text{cubeUp}_2.y$
$\texttt{dropOff}_3$	$(\text{cubeUp}_3.z - \text{cubeUp}_2.z) - \text{cubeLength}$ (distance between cube surfaces)
$\texttt{onTop}_3$	indicates if cubeUp_3 is on top of cubeUp_2 after the stacking process

Fig. 4. (a) visualizes the variables that are used to describe multiple stacks and (b) their meaning. Note the offset variables are measured always with respect to the previous cube.

created three different test datasets, each consisting of 40,000 samples, following a similar distribution but other seed as in the training dataset. The first test dataset represents the case of no corrections (our baseline), followed by applying the smaller model on each of the cubes (no history case). Finally, we used the complete $3-\texttt{Stack}-\texttt{Model}_g$ to correct the data accordingly.

The goal set $G = \{\texttt{onTop}_3\}$ for this experiment indicates the stacking success of the third cube, and we denote the action as successful *iff* $\texttt{onTop}_3$ = True. Thus $\mathbf{X}_{goal_{E2}} = \{\texttt{onTop}_3 = \text{True}\}$. Since the experiment was stopped, if one of the previous stacks has failed, a successful third stack will imply success in the other two stacks as well. We hypothesize that, while the $1-\texttt{Stack}-\texttt{Model}_g$ will improve the stacking success of the complete tower, the $3-\texttt{Stack}-\texttt{Model}_g$ will be even more helpful since it takes the entire history of all single stacking actions into account.

5. Results and discussion

In this section, we present and discuss the results of the two experiments introduced in Section 4. We set the probability threshold which distinguishes a failure from success to $\epsilon = 0.8$.

5.1. Assessing the obtained causal models

We first analyze the obtained causal models in terms of the graphical structure of the learned BN. Then, we explain the obtained conditional probabilities that were fitted around the experiment data. We validate the correctness of the model to make sure that the predictions are not based on a flawed understanding of cause–effect relations, which could result in wrong failure explanations and obstruct failure prevention.

5.1.1. Obtained causal model for Experiment 1

Fig. 5 visualizes the obtained causal relations between the subset of variables $\mathbf{X}_{E1} = \{\texttt{xOff}_1, \texttt{yOff}_1, \texttt{dropOff}_1\ \texttt{onTop}_1\}$, by applying the PC structure learning algorithm to the collected data. The results exhibit dependencies of all the cube positioning variables ($\texttt{dropOff}_1, \texttt{xOff}_1, \texttt{yOff}_1$) on the stacking outcome $\texttt{onTop}_1$. The variables from $\mathbf{X}_{E1}$ were discretized according to Table 1. For example, the variable of $\texttt{dropOff}_1$ has three possible intervals (z_1, z_2, and z_3). Fig. 6 visualizes the obtained probabilities for the stacking success of *CubeUp*$_1$ conditional on the analyzed variables. Generally, for all three drop-offset intervals, the causal model showed large stacking success probabilities centered around smaller x/y-offsets and decreases the larger the x/y-offsets become. This decrease in probability is faster for higher drop-offset positions (e.g., compare the two drop-offset intervals of z_1 and z_3). For the most extreme x/y-offset intervals, the model displays a stacking success of below 0.2 (red areas in Fig. 6), which is credible considering that the center of gravity of the stacked cubes in these x/y-offset intervals is close to the limits or outside the surface of the bottom cube. We conclude that the probability distributions trained on simulated data are plausible.

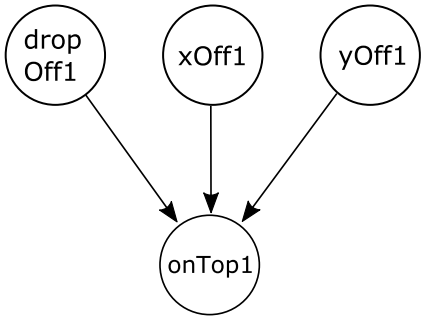


Fig. 5. Obtained $1-\texttt{Stack}-\texttt{Model}_u$ Bayesian network structure for the Stack-1-Cube scenario.

Table 1
Enlists the obtained discretization intervals for the variables $\mathbf{X}_{E1}$ (in cm).

$\texttt{dropOff}_1$	$\texttt{xOff}_1$	$\texttt{yOff}_1$
z_1 : [0.5, 3.7]	x_1 : [−3.0, −1.8]	y_1 : [−3.0, −1.8]
z_2 : (3.7, 6.8]	x_2 : (−1.8, −0.6]	y_2 : (−1.8, −0.6]
z_3 : (6.8, 10.0]	x_3 : (−0.6, 0.58]	y_3 : (−0.6, 0.6]
	x_4 : (0.58, 1.78]	y_4 : (0.6, 1.8]
	x_5 : (1.78, 3.0]	y_5 : (1.8, 3.0]

5.1.2. Learned causal model for Experiment 2

The obtained DAGs for the two evaluation cases ($1-\texttt{Stack}-\texttt{Model}_g$ and $3-\texttt{Stack}-\texttt{Model}_g$), are displayed in Fig. 7. For both cases, we used the PC structure learning algorithm and discretized the used variables according to Table 2. In the case of the $1-\texttt{Stack}-\texttt{Model}_g$ ($i = 1$), we obtained a slightly different dependency structure than in the $1-\texttt{Stack}-\texttt{Model}_u$ that we obtained from experiment 1. In particular, we notice from Fig. 7 that the drop-offset variable $\texttt{dropOff}_1$ is now independent of the stacking outcome $\texttt{onTop}_i$ compared to the obtained model shown in Fig. 5 due to the different sampling distributions. In the new model, the Gaussian distribution for $\texttt{dropOff}_1$ samples created more values around the mean of 1 cm, and smaller drop-offsets are shown not to

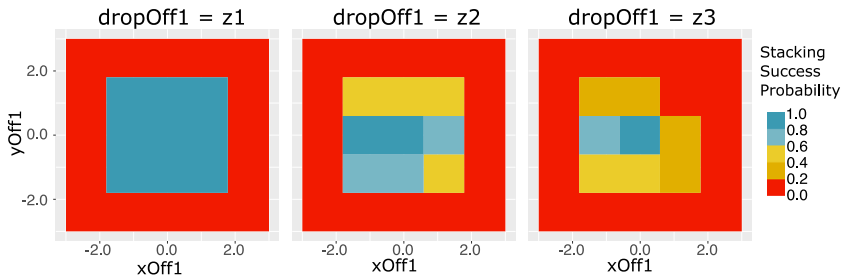


Fig. 6. Visualization of the conditional probability table for $P(\texttt{onTop}_1 = 1 | \Pi_{\texttt{onTop}_1})$. $\texttt{xOff}_1$, $\texttt{yOff}_1$ are discretized into 5 intervals and $\texttt{dropOff}_1$. Values for $\texttt{xOff}_1$, $\texttt{yOff}_1$ are in cm.

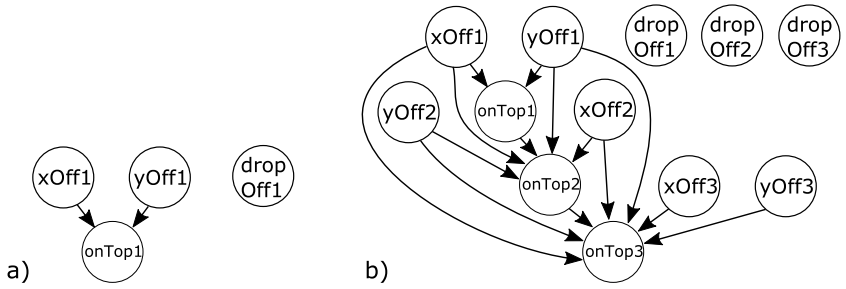


Fig. 7. Obtained Bayesian network structure for the 3-Stack-Cube scenario. In (a) the causal model obtained from only the first stacking action is compared with (b) the complete causal model covering all three stacks.

Table 2
Enlists the obtained discretization intervals for the variables $\mathbf{X}_{E2}$ (in cm).

$\texttt{dropOff}_{1,2,3}$	$\texttt{x/yOff}_{1,2}$	$\texttt{x/yOff}_3$
$z_1 : [0.5, 1.9]$	$x_1 : [-3.0, -1.4]$	$y_1 : [-3.0, -0.7]$
$z_2 : (1.9, 3.6]$	$x_2 : (-1.4, -0.4]$	$y_2 : (-0.7, 0.7]$
$z_3 : (3.6, 10]$	$x_3 : (-0.4, 0.4]$	$y_3 : (0.7, 3.0]$
	$x_4 : (0.4, 1.4]$	
	$x_5 : (1.4, 3.0]$	

Table 3
Percentage of failed stacking actions in Experiment 1 (40,000 data samples). The displayed results were obtained with ($\epsilon = 0.8$).

Stack-1-Cube	Failure percentage	Corrected failures
no model	74%	0%
$1 - \texttt{Stack} - \texttt{Model}_u$	1.9%	97%

have any measurable impact on the stacking success. Structure learning methods find causal models that most accurately fit the underlying data. If one of the variables, such as $\texttt{dropOff}_1$, is not recognized to have a causal impact on the outcome of the action, this variable simply will not play a role in the failure explanations and preventions. In the particular case of Experiment 2, failures will not be associated with too high drop-offset positions but only with the x/y-offsets of the three stacking actions.

A similar graph dependency structure between the variables is observed in the $3 - \texttt{Stack} - \texttt{Model}_g$ ($i = 1, 2, 3$), where the $\texttt{dropOff}_i$ variables are independent of the stacking success. From Fig. 7.b, it becomes evident that the success of each stack depends on an increasing number of parent nodes. Interestingly, not only the x/y-offset variables but also previous $\texttt{onTop}_i$ impact the stacking success (e.g., consider the arrow from $\texttt{onTop}_2$ to $\texttt{onTop}_3$). The reason for these causal links is the termination of the stacking experiments in cases where the previous stack has already failed. We conclude that the $3 - \texttt{Stack} - \texttt{Model}_g$ successfully captures the dependence of earlier stacks on the outcome of later stacking actions, thus encoding the action history in one causal model.[3]

5.2. Evaluation of prediction and failure prevention capabilities

Next, we analyze the ability of our obtained causal models to predict and avoid potential future failures by correcting the cubes' stacking position to the closest variable parametrization (as explained in Section 3.4) that is predicted to lead to successful execution. Practically, when the robot predicts a failure, it will modify its movements to the closest position in which the robot is likely going to succeed.

5.2.1. Failure prediction and prevention in Experiment 1

The results from the test datasets of experiment 1 are displayed in Table 3. Out of 40,000 collected samples of the ground-truth dataset, almost 30,000 stacking experiments failed (around 74%) without corrective actions. Our model could fix and prevent 97%[4] of these failures. Given the obtained failure prevention of 97%, we conclude that our model is highly beneficial for predicting and avoiding future errors.

5.2.2. Failure prediction and prevention in Experiment 2

The failure prediction and prevention capabilities for experiment 2 are analyzed in Table 4. Without any corrective actions (Table 4 - no correction), we sampled around 81% stacking failures. 22% of the failures happened during the first stacking action, 42% during the second, and 36% during the third. Thus the failure probability increases with the height of the tower. The decreasing number of errors for the third stacking action can be attributed to the limited number of data samples, as less than 50% of all experiments reach the third stack. As the second row of Table 4 shows, applying the $1 - \texttt{Stack} - \texttt{Model}_g$ on all three stacking actions reduced the number of stacking failures significantly by

[3] Note that we do not visualize the conditional probability table for this experiment due to the increased number of variables that the stacking outcome is conditioned on.

[4] The failure reduction achieved by the $1 - \texttt{Stack} - \texttt{Model}_u$ can be computed from the failure percentages in Table 3 as $1 - \frac{1.9\%}{74\%}$.

Table 4
Percentage of failed stacking tasks in Experiment 2 (40,000 data samples). The displayed results were obtained with ($\epsilon = 0.8$).

Stack-3-Cubes	Fail. perc.	Fail in 1	Fail in 2	Fail in 3
no correction	81%	22%	42%	36%
$1-\texttt{Stacks}-\texttt{Model}_g$	18%	5%	12%	83%
$3-\texttt{Stack}-\texttt{Model}_g$	**4.3**%	0%	0.5%	99.5%

Table 5
Two examples for failure prevention. Corresponding real-world experiments are visualized in Figs. 8 and 9.

Input	Input interval	Curr. succ. prob.	Closest solution interval	Exp. succ. prob.
Example 1:				
xOff1 = 0.02	$\boldsymbol{x_5}$		$\boldsymbol{x_3}$	
yOff1 = 0.0	y_3		y_3	
dropOff1 = 0.01	z_1		z_1	
xOff2 = 0.01	x_4		x_4	
yOff2 = 0.0	y_3	0.05	y_3	0.96
dropOff2 = 0.01	z_1		z_1	
xOff3 = 0.0	x_2		x_2	
yOff3 = 0.0	y_2		y_2	
dropOff3 = 0.01	z_1		z_1	
Explanation: The first cube was stacked too far to the right.				
Example 2:				
xOff1 = 0.01	$\boldsymbol{x_4}$		$\boldsymbol{x_3}$	
yOff1 = −0.01	y_2		y_2	
dropOff1 = 0.01	z_1		z_1	
xOff2 = 0.01	x_4		x_4	
yOff2 = −0.01	y_2	0.04	y_2	0.82
dropOff2 = 0.01	z_1		z_1	
xOff3 = 0.01	$\boldsymbol{x_3}$		$\boldsymbol{x_2}$	
yOff3 = −0.01	$\boldsymbol{y_1}$		$\boldsymbol{y_2}$	
dropOff3 = 0.01	z_1		z_1	
Explanation: The first cube was stacked too far to the right and the third cube was stacked too far to the right and back.				

78%.[5] Applying the $3-\texttt{Stack}-\texttt{Model}_g$ even prevents 95%[6] of the failures. We, therefore, conclude that both models prevent failures well. As expected, the $3-\texttt{Stack}-\texttt{Model}_g$ outperforms the $1-\texttt{Stack}-\texttt{Model}_g$, due to capturing the causal effects over all three stacking actions in one single model.

5.2.3. Explanation & prevention of timely shifted action errors: Evaluation on the physical robot

We demonstrate the ability of the $3-\texttt{Stack}-\texttt{Model}_g$ to explain and prevent timely shifted action errors in two physical cube stacking tasks (see Table 5). For each example, we performed six real-world experiments; three for the initial variable parametrization and three for the correction proposed by the $3-\texttt{Stack}-\texttt{Model}_g$. The examples showcase scenarios where the tower did not fall directly when the error had been committed but only became evident at a later stage. In example 1 (see Table 5, Fig. 8), the first cube has been stacked far to the right. This did not lead to a failure immediately, but after stacking the second (1 out of 3 times) or third cube (2 out of 3 times), the tower fell. Both models ($3-\texttt{Stack}-\texttt{Model}_g$ and $1-\texttt{Stack}-\texttt{Model}_u$) detected that the culprit was the first stacking action and not the second or third. Taking each stack individually would have also yielded a correction of the first

[5] The failure reduction achieved by the $1-\texttt{Stack}-\texttt{Model}_g$ can be computed from the failure percentages in Table 4 as $1-\frac{18\%}{81\%}$.

[6] The failure reduction achieved by the $3-\texttt{Stack}-\texttt{Model}_g$ can be computed from the failure percentages in Table 4 as $1-\frac{4.3\%}{81\%}$.

Table 6
Confusion matrix for success prediction with the $1-\texttt{Stack}-\texttt{Model}_u$.

	Predicted as correct	Predicted as failure	
actually correct	15.1%	11.1%	26.2%
actually a failure	0.7%	73.1%	73.8%
	15.8%	84.2%	100%

Table 7
Confusion matrix for success prediction of $3-\texttt{Stack}-\texttt{Model}_g$ for $\epsilon = 0.8$.

	Predicted as correct	Predicted as failure	
actually correct	0.2%	18.9%	19.1%
actually a failure	0.0%	80.9%	80.9%
	0.2%	99.8%	100%

stacking action since the $1-\texttt{Stack}-\texttt{Model}_u$[7] predicts a success probability of < 0.2 (Fig. 6), despite the tower not failing in reality. However, in example 2 (see Table 5, Fig. 9), we investigated a case where the $1-\texttt{Stack}-\texttt{Model}_u$ did not predict any errors since each stack looks perfectly fine on its own. Only due to the cumulative effect of several offsets did we get a failure at the last stack of cube 3 (3 out of 3 cases). Without the history of the $3-\texttt{Stack}-\texttt{Model}_g$, we would not be able to correct this example. In both examples, the corrected version by the $3-\texttt{Stack}-\texttt{Model}_g$ succeeded in all three trials.[8]

6. Discussion of the obtained results

Despite the superiority of the $3-\texttt{Stack}-\texttt{Model}_g$ when it comes to failure prevention, the $1-\texttt{Stack}-\texttt{Models}$ perform remarkably well. One indicator for this success can be found in the confusion matrix (see Table 6). Table 6 shows that the $1-\texttt{Stack}-\texttt{Model}_u$ predicts almost all actual failures as failures (true negatives). At the same time, it predicts around 42%[9] of the actually correct cases as failures (false negatives). Consider the first stacking action of the previously discussed example 1 in Section 5.2.3. This action was corrected by the $1-\texttt{Stack}-\texttt{Model}_u$ because it was predicted to have a low success chance, even though, in reality, it did not fail. However, also the $3-\texttt{Stack}-\texttt{Model}_g$ has large true negative and false negative ratios (see Table 7). Therefore, our models are conservative in the sense that they try to avoid failures at the cost of misclassifying a large number of successful stacking executions as a failure. For the discussed application, this is beneficial as we want to avoid stacking failures, and an additional corrective action is more acceptable than a failed tower.

6.1. Impact of the success threshold ϵ on failure prevention

To better understand the reason for this large number of false negatives, we analyzed the role of ϵ. As expected, larger ϵ-values lead to fewer stacking failures. Table 8 shows that the $3-\texttt{Stack}-\texttt{Model}_g$ with $\epsilon = 0.6$ has a lower failure prevention success than the $1-\texttt{Stack}-\texttt{Model}_g$ with $\epsilon = 0.8$. The $3-\texttt{Stack}-\texttt{Model}_g$ with $\epsilon = 0.95$ is able to prevent almost all stacking failures from happening. The impact of ϵ can be concretely demonstrated on the two examples that are discussed

[7] In reality, the underlying distribution of variables like $\texttt{xOff}_i$ might be unknown. Therefore we cannot know if the model trained on uniform or Gaussian data is more appropriate.

[8] The robot executions can be seen in https://youtu.be/VT55y15lmt4.

[9] calculated as $\frac{11.1\%}{26.2\%} = 42\%$

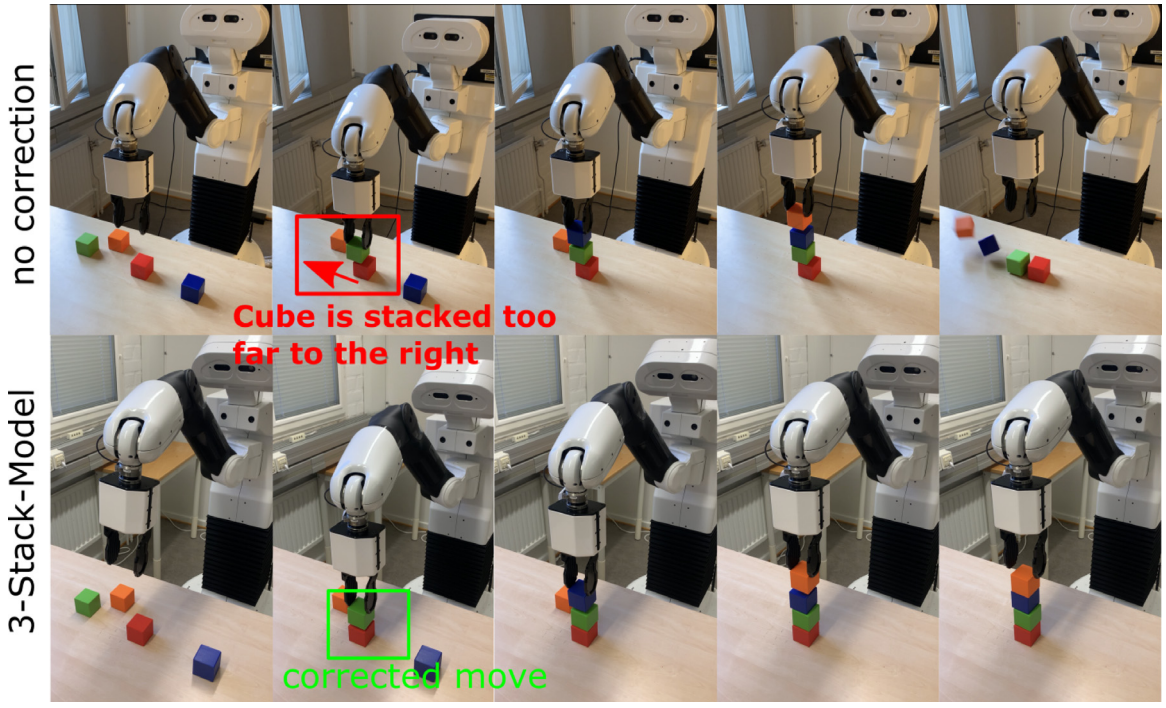


Fig. 8. Displays the stacking execution of example 1 performed on the real robot. In the first row, the $\texttt{cubeUp}_1$ (green) is stacked too far to the right, which leads to a failure in the third stack. In the corrected sequence, as proposed by the $3-\texttt{Stack}-\texttt{Model}_g$, $\texttt{cubeUp}_1$ (green) is stacked more to the left, which allows the robot to successfully stack all three cubes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

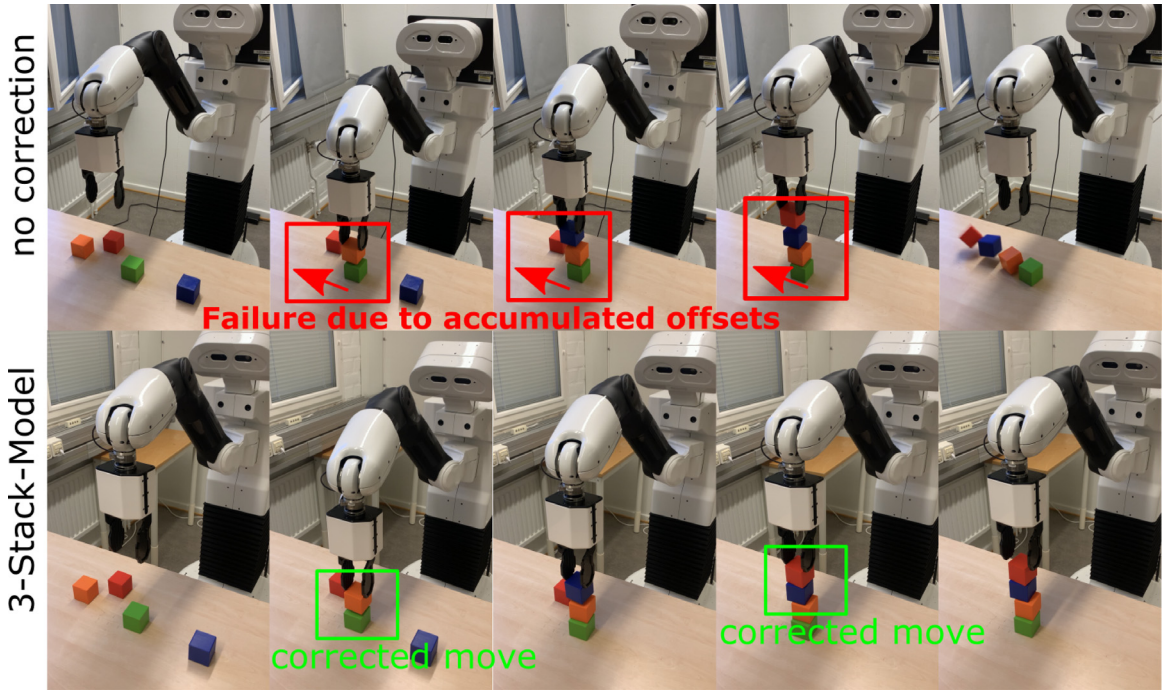


Fig. 9. Displays the stacking execution of example 2 performed on the real robot. In the first row, each cube is stacked a little to the right and down. Each stack on itself is not found to be problematic by $1-\texttt{Stack}-\texttt{Model}_u$, and thus no corrective actions are found. However, with the third cube, the cumulative error is too large, and the tower falls. In the corrected sequence, as proposed by the $3-\texttt{Stack}-\texttt{Model}_g$, $\texttt{cubeUp}_1$ (orange) is stacked a little more to the left and $\texttt{cubeUp}_3$ (red) is also stacked more centered, which allows the robot to successfully stack all three cubes, despite further offsets the second stacking action. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

in Section 5.2.3. The preventive actions based on the closest solution intervals in Table 5 are based on $\epsilon = 0.8$. In example 1, $\texttt{xOff}_1$ will only change to x_4 with $\epsilon = 0.6$, resulting in a success chance of 0.65. With $\epsilon = 0.95$, our method finds similar preventive actions as with $\epsilon = 0.8$ for example 1. However, for example 2, the closest solution interval has an additional change

Table 8
Percentage of failed stacking tasks including failure preventive actions based on the $3-\texttt{Stack}-\texttt{Model}_g$ with different choices of ϵ.

Stack-3-Cubes	fail. perc.	Fail in 1	Fail in 2	Fail in 3
$3-\texttt{Stack}-\texttt{Model}_g$ ($\epsilon = 0.6$)	26.8%	14.9%	8.7%	84.3%
$3-\texttt{Stack}-\texttt{Model}_g$ ($\epsilon = 0.8$)	4.3%	0%	0.5%	99.5%
$3-\texttt{Stack}-\texttt{Model}_g$ ($\epsilon = 0.95$)	**0.02**%	0%	16.7%	83.3%

Table 9
Confusion matrix for success prediction of $3-\texttt{Stack}-\texttt{Model}_g$ for $\epsilon = 0.6$.

	Predicted as correct	Predicted as failure	
actually correct	0.8%	18.2%	19.0%
actually a failure	0.3%	80.7%	81.0%
	1.1%	98.9%	100%

Table 10
Confusion matrix for success prediction of $3-\texttt{Stack}-\texttt{Model}_g$ for $\epsilon = 0.95$.

	Predicted as correct	Predicted as failure	
actually correct	0.1%	18.9%	19.0%
actually a failure	0.0%	80.9%	80.9%
	0.1%	99.8%	100%

in the $\texttt{yOff}_2$ variable from y_2 to y_3, resulting in a success chance of 0.97. Nevertheless, the number of false negatives is over 18% for all three ϵ (as seen in Tables 7, 9 and 10). We conclude that it is important to choose a large ϵ value to achieve better failure prevention probabilities. At the same time, we conclude that the choice of ϵ only has a small effect on the number of false negatives in our models. Instead, we suspect that the number of discretization intervals and the much larger number of action samples with negative stacking outcomes are important factors for the large number of false negatives.

6.2. Data efficiency and transferability of the causal models

While the causal structure that is represented through a BN can often be transferred, the conditional probability distributions typically need to be relearned. For example, our results showed that the causal structure of the 1-Stack-Model is applicable to all the individual stacking actions of the Stack-3-Cubes scenario. Potentially it might be applicable even beyond to other tasks, such as stacking cups or plates, as the x/y-offset plays a very crucial role in the stacking success. The conditional probability distributions are transferable between tasks that are close, as demonstrated through the excellent failure prevention capabilities of the $1-\texttt{Stack}-\texttt{Model}_g$ for the Stack-3-Cubes scenario. However, generally, the precise parameters are situational and need to be relearned, e.g., applying the cube stacking models for the action of stacking plates.

One limitation of our approach is the data required for learning large causal models. We conducted tests with the two different structure learning algorithms of the PC algorithm [27] and grow-shrink [30] and found that at least 200,000 and 600,000 samples were respectively required to learn the causal structure of the $3-\texttt{Stack}-\texttt{Model}_g$. To understand the data requirements of the parameter learning step, we investigated how quickly the estimation converges towards the final parameters after 800,000 samples (see Fig. 10). Our findings show that for a subset of

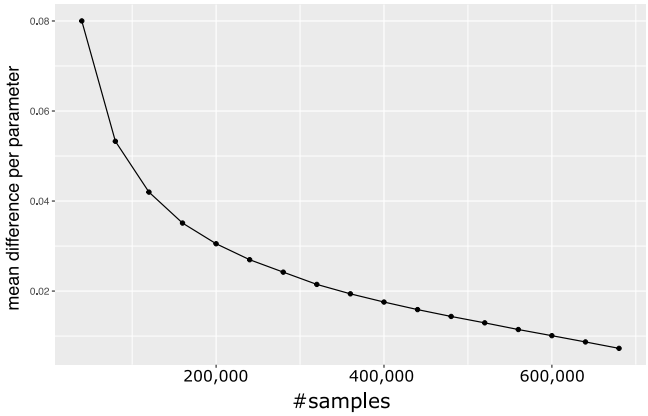


Fig. 10. Mean difference per parameter, comparing the parameter estimation based on an increasing number of samples with the parameter estimation based on all 800,000 samples. For each investigated number of samples, we have randomly chosen five different subsets of samples (out of the complete dataset of 800,000 samples) and calculated the mean of the per parameter difference between the parameter estimation given the subset and the complete dataset.

400,000 the mean per parameter difference between the subset and the full data set of 800,000 samples is less than 0.02. Such a difference will hardly matter as our algorithm proposes corrective actions based on the ϵ threshold. If the actual success chance is slightly lower than predicted (e.g., 0.79 instead of 0.81), our algorithm would, in the worst case, choose a different corrective action than it would if provided with the actual success chance. However, with a reasonably large ϵ, such as $\epsilon = 0.8$, that would likely not lead to any action failures as the success chance is still quite large. If the actual success chance is slightly higher than predicted, our model might propose a different corrective action, which, however, is even more conservative and thus more likely to succeed. To conclude, our causal model will perform equally well even with less number of samples. However, for the time being, we assume that the collected number of samples is enough to obtain good estimates of the causal structure and parameters, and we leave it for future work to find more automated strategies for deciding how much data is enough.

7. Conclusion

In this paper, we propose a causal-based method that allows robots to understand possible causes for execution errors and predict how likely an action will succeed. We then introduce a novel method that utilizes these prediction capabilities to find corrective actions which will allow the robot to prevent failures from happening. Our algorithm proposes a solution to the complex challenge of timely shifted action effects. By detecting causal links over the history of several actions, the robot can effectively predict and prevent failures, even if the root of a failure lies in a previous action. We have shown the success of our approach for the problem of stacking cubes in two cases; (a) single stacks (stacking one cube) and; (b) multiple stacks (stacking three cubes). In the single-stack case, our method was able to reduce the error rate by 97%. We also show that our approach can scale to capture various actions in one model, allowing us to measure timely shifted action effects, such as the impact of an imprecise stack of the first cube on the stacking success of the third cube. For these complex situations, our model was able to prevent around 95% of the failures.

Despite being able to capture action histories in one model, one challenge of obtaining such large models is data efficiency. The more parents a BN node has, the more samples are required

to learn its graphical structure and the conditional probabilities. In the future, we will investigate intelligent ways of pre-initializing the model, e.g., by utilizing the single-action models as prior for structure and probabilities, with the goal of reducing the number of new samples.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

The research reported in this paper has been supported by Chalmers AI Research Centre (CHAIR) through grant 32631004, 32631021-22.

References

[1] A. Kuestenmacher, N. Akhtar, P. Plöger, G. Lakemeyer, Towards robust task execution for domestic service robots, Proceedings of the International Conference on Automated Planning and Scheduling 24 (1) (2014) 528–531, http://dx.doi.org/10.1609/icaps.v24i1.13653.

[2] E. Khalastchi, M. Kalech, On fault detection and diagnosis in robotic systems, ACM Comput. Surv. 51 (1) (2018) http://dx.doi.org/10.1145/3146389.

[3] J. Pearl, D. Mackenzie, The Book of Why: The New Science of Cause and Effect, first ed., Basic Books, Inc. USA, 2018.

[4] T. Miller, Explanation in artificial intelligence: Insights from the social sciences, Artificial Intelligence 267 (2019) 1–38.

[5] T. Hellström, The relevance of causation in robotics: A review, categorization, and analysis, Paladyn, J. Behav. Robot. 12 (1) (2021) 238–255, http://dx.doi.org/10.1515/pjbr-2021-0017.

[6] A. Mitrevski, A.F. Abdelrahman, A. Narasimamurthy, P.G. Plöger, On the diagnosability of actions performed by contemporary robotic systems, in: 31th International Workshop on Principles of Diagnosis, DX, 2020, URL http://dx-2020.org/papers/DX-2020_paper_6.pdf.

[7] A. Mitrevski, P.G. Plöger, G. Lakemeyer, Robot action diagnosis and experience correction by falsifying parameterised execution models, in: 2021 IEEE International Conference on Robotics and Automation, ICRA, 2021, pp. 11025–11031, http://dx.doi.org/10.1109/ICRA48506.2021.9561710.

[8] D. Lewis, Causal explanation, in: D. Lewis (Ed.), Philosophical Papers Vol. II, Oxford University Press, 1986, pp. 214–240.

[9] A. Mitrevski, P.G. Plöger, G. Lakemeyer, Representation and experience-based learning of explainable models for robot action execution, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2020, pp. 5641–5647, http://dx.doi.org/10.1109/IROS45743.2020.9341470.

[10] A.S. Bauer, P. Schmaus, F. Stulp, D. Leidner, Probabilistic effect prediction through semantic augmentation and physical simulation, in: 2020 IEEE International Conference on Robotics and Automation, ICRA, 2020, pp. 9278–9284, http://dx.doi.org/10.1109/ICRA40945.2020.9197477.

[11] M. Diehl, K. Ramirez-Amaro, Why did I fail? A causal-based method to find explanations for robot failures, IEEE Robot. Autom. Lett. 7 (4) (2022) 8925–8932, http://dx.doi.org/10.1109/LRA.2022.3188889.

[12] D. Altan, S. Sariel, Probabilistic failure isolation for cognitive robots, in: 27th International FLAIRS (Florida Artificial Intelligence Research Society) Conference, 2014.

[13] J. Brawer, M. Qin, B. Scassellati, A causal approach to tool affordance learning, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2020, pp. 8394–8399, http://dx.doi.org/10.1109/IROS45743.2020.9341262.

[14] T.E. Lee, J.A. Zhao, A.S. Sawhney, S. Girdhar, O. Kroemer, Causal reasoning in simulation for structure and transfer learning of robot manipulation policies, in: 2021 IEEE International Conference on Robotics and Automation, ICRA, 2021, pp. 4776–4782, http://dx.doi.org/10.1109/ICRA48506.2021.9561439.

[15] A.A. Bhat, V. Mohan, G. Sandini, P.G. Morasso, Humanoid infers Archimedes' principle: understanding physical relations and object affordances through cumulative learning experiences, J. R. Soc. Interface 13 (2016).

[16] C. Uhde, N. Berberich, K. Ramirez-Amaro, G. Cheng, The robot as scientist: Using mental simulation to test causal hypotheses extracted from human activities in virtual reality, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2020, pp. 8081–8086, http://dx.doi.org/10.1109/IROS45743.2020.9341505.

[17] D. Song, K. Huebner, V. Kyrki, D. Kragic, Learning task constraints for robot grasping using graphical models, in: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010, pp. 1579–1585, http://dx.doi.org/10.1109/IROS.2010.5649406.

[18] M. Diehl, C. Paxton, K. Ramirez-Amaro, Automated generation of robotic planning domains from observations, in: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2021, pp. 6732–6738, http://dx.doi.org/10.1109/IROS51168.2021.9636781.

[19] T. Chakraborti, S. Sreedharan, S. Kambhampati, The emerging landscape of explainable automated planning & decision making, in: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI '20, 2021, pp. 4803–4811.

[20] B. Seegebarth, F. Müller, B. Schattenberg, S. Biundo, Making hybrid plans more clear to human users — a formal approach for generating sound explanations, Proceedings of the International Conference on Automated Planning and Scheduling (1) (2012) 225–233, http://dx.doi.org/10.1609/icaps.v22i1.13503.

[21] D. Das, S. Banerjee, S. Chernova, Explainable AI for robot failures: Generating explanations that improve user assistance in fault recovery, in: HRI '21: ACM/IEEE International Conference on Human-Robot Interaction, Boulder, CO, USA, March 8-11, 2021, ACM, 2021, pp. 351–360, http://dx.doi.org/10.1145/3434073.3444657.

[22] M.J. Vowels, N.C. Camgoz, R. Bowden, D'Ya Like DAGs? A survey on structure learning and causal discovery, ACM Comput. Surv. (2022).

[23] M. Scutari, Learning Bayesian networks with the bnlearn R package, J. Stat. Softw. 35 (3) (2010) 1–22, http://dx.doi.org/10.18637/jss.v035.i03.

[24] Y.-C. Chen, T.A. Wheeler, M.J. Kochenderfer, Learning discrete Bayesian networks from continuous data, J. Artif. Int. Res. 59 (1) (2017) 103–132.

[25] R. Nagarajan, M. Scutari, Bayesian Networks in R with Applications in Systems Biology, Springer, New York, 2013, http://dx.doi.org/10.1007/978-1-4614-6446-4, ISBN 978-1-4614-6445-7, 978-1-4614-6446-4.

[26] J.L. Lustgarten, S. Visweswaran, V. Gopalakrishnan, G.F. Cooper, Application of an efficient Bayesian discretization method to biomedical data, BMC Bioinformatics 12 (1) (2011) 1–15.

[27] D. Colombo, M.H. Maathuis, Order-independent constraint-based causal structure learning, J. Mach. Learn. Res. 15 (1) (2014) 3741–3782.

[28] A. Sharma, V. Syrgkanis, C. Zhang, E. Kiciman, DoWhy: Addressing challenges in expressing and validating causal assumptions, in: ICML Workshop: The Neglected Assumptions in Causal Inference, 2021.

[29] J. Peters, D. Janzing, B. Schlkopf, Elements of Causal Inference: Foundations and Learning Algorithms, The MIT Press, 2017.

[30] D. Margaritis, Learning Bayesian Network Model Structure From Data (Ph.D. dissertation), Carnegie Mellon University Pittsburgh PA School of Comput. Sci., 2003.

M. Sc. Maximilian Diehl is a Ph.D. student with the Division of Systems and Control (SYSCON), Department of Electrical Engineering (E2) at the Chalmers University of Technology since January 2020. Previously, he received his Master and Bachelor degree in Electrical Engineering and Computer Science at the Technical University of Munich (TUM), Germany in 2019 and 2018 respectively. His research interests include Artificial Intelligence and Robotics, in particular Causality, Planning and explainable Decision Making for robots.

Dr. Karinne Ramirez-Amaro is an Associate Professor at the Department of Electrical Engineering at the Chalmers University of Technology since 2022. In 2019, she became an Assistant Professor at Chalmers in the research group of Mechatronics. Previously, she was a post-doctoral researcher at the Chair for Cognitive Systems at the Technical University of Munich (TUM). She completed her Ph.D. (summa cum laude) at the Department of Electrical and Computer Engineering at the TUM, Germany in 2015. She received the Laura Bassi award granted by TUM and the Bavarian government in 2015, also in that year she received the price of excellent Doctoral degree for female engineering students, granted by the state of Bavaria, Germany. In 2011, she received the Google Anita Borg scholarship. Her research interests include Semantic Representations, Interpretable methods, and Human Activity Recognition and Understanding.