Using Platform Models for a Guided Explanatory Diagnosis Generation for Mobile Robots

Daniel Habering , Till Hofmann , Gerhard Lakemeyer
Knowledge-Based Systems Group, RWTH Aachen University, Germany
{habering, hofmann, lakemeyer}@kbsg.rwth-aachen.de

Abstract

Plan execution on a mobile robot is inherently error-prone, as the robot needs to act in a physical world which can never be completely controlled by the robot. If an error occurs during execution, the true world state is unknown, as a failure may have unobservable consequences. One approach to deal with such failures is diagnosis, where the true world state is determined by identifying a set of faults based on sensed observations. In this paper, we present a novel approach to explanatory diagnosis, based on the assumption that most failures occur due to some robot hardware failure. We model the robot platform components with state machines and formulate action variants for the robots' actions, modelling different fault modes. We apply diagnosis as planning with a top-k planning approach to determine possible diagnosis candidates and then use active diagnosis to find out which of those candidates is the true diagnosis. Finally, based on the platform model, we recover from the occurred failure such that the robot can continue to operate. We evaluate our approach in a logistics robots scenario by comparing it to having no diagnosis and diagnosis without platform models, showing a significant improvement to both alternatives.

1 Introduction

The execution of an action plan on a mobile robot is inherently error-prone: As a robot acts in the physical world, its actions may fail or have unexpected effects, and there may be interference by exogenous actions, e.g., by a human. Consider a mobile robot with a manipulator as shown in Figure 1 that is supposed to operate a machine, e.g., by placing an object onto its conveyor belt: If the robot is mislocalized, it may put the object next to the conveyor belt instead. If the arm is miscalibrated, it may even collide with the machine, possibly dropping the workpiece. If the robot is not equipped with a suitable sensor, it may not immediately detect that it dropped the workpiece, and proceed as if everything went as planned.

If such a fault occurs, the agent needs to determine the true world state, based on partial observations it can make. One approach is to find out *what is wrong* by applying *diagnosis* (e.g., [Reiter, 1987; de Kleer and Williams, 1987]). In diagnosis, the goal is to identify a set of faults, based on a model of the system and a set of observations. In a robotics application, it is often useful to determine *what happened* by explaining the observations with a sequence of (exogenous) actions [Cordier and Thiébaux, 1994; Mcllraith, 1999]. *History-based diagnosis* [Iwan, 2001] determines the true sequence of events based on an expected action history, e.g., the actions that the robot was supposed to execute. Unexpected events are modeled by action variants and exogenous actions, e.g., an action `put-drop` that drops the object instead of putting it onto the conveyor belt. During diagnosis, those action variants may occur randomly, i.e., there is no modeled underlying cause. Any adapted sequence of events that is consistent with the observations form a possible explanation about what might have happened.

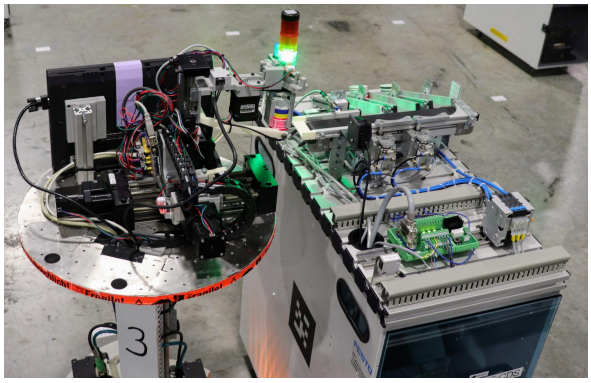


Figure 1: A robot placing a workpiece onto a machine.

However, on a mobile robot, such action variants often do not occur randomly, but have an underlying cause. As an example, a failed `put` action is often caused by a miscalibrated gripper. If the agent tries the action again, it will just fail again with the same error. Thus, we propose to combine diagnosis as fault analysis in the sense of Reiter [1987] with history-based diagnosis. Instead of letting action variants occur randomly, we propose to explicitly model the robot platform as state machines and determine the state of the platform during diagnosis. Action variants then depend on the state of a platform component, e.g., `put-drop` may only oc-

cur if the gripper is miscalibrated. This approach has two advantages: First, the robot no longer re-tries actions that are bound to fail; instead, it can avoid using the faulty component altogether. Second, the robot can try to recover from the underlying cause, e.g., by recalibrating the arm.

To determine a diagnosis efficiently, we make use of diagnosis as planning [Mcllraith, 1999; Sohrabi *et al.*, 2010] and combine it with history-based diagnosis. We do this by generating a PDDL domain that enforces the agent's action history and that allows for action variants based on the platform models. We apply a top-k planning approach [Katz *et al.*, 2018] to determine k cost-optimal diagnoses to focus on diagnosis candidates with a minimal number of assumed faults.

Sometimes, the observations are not sufficient to provide a unique explanation. In this case, *active diagnosis* [Sampath *et al.*, 1998] aims to distinguish the explanation candidates with sensing actions. We combine active diagnosis with our platform models. We assign probabilities to exogenous events in the platform model (e.g., the arm getting miscalibrated), which allows us to choose the next sensing action with the maximal information gain.

In the following, we start by summarizing related work in Section 2. We continue with the approach by describing how to model the robot hardware platform (Section 3.1) and how to generate a PDDL domain for history-based diagnosis (Section 3.2), which can then be used with a top-k planner (Section 3.3). In Section 3.4, we explain how to incorporate knowledge about hardware faults into active diagnosis. We evaluate our approach in Section 4 and conclude in Section 5.

2 Related Work

Model-based diagnosis [de Kleer and Williams, 1987] describes the process of determining a set of faults that explain a set of observations, given a model of the system behavior. In *consistency-based diagnosis*, a diagnosis is a minimal set of faults that are entailed by the observations [Reiter, 1987]. In contrast, in abductive diagnosis, the assumptions must entail the observations [Poole, 1989a; Poole, 1989b]. Sampath *et al.* [1995] describe a first diagnosis system based on discrete event systems, where the system is modeled as finite automaton. If multiple explanation candidates are viable, *active diagnosis* [Sampath *et al.*, 1998; Baral *et al.*, 2000; Chanthery and Pencolé, 2009] determines the correct explanation by actively controlling the system to gather missing information. Mühlbacher and Steinbauer [2014] distinguish active diagnosis candidates based on entropy and noise sensors to find the least costly diagnosis plan. In *pervasive diagnosis* [Kuhn *et al.*, 2008], the system controller is instead directly adapted to maximize diagnostic information during execution.

Similar to the proposed approach, the tool `COMPASS` [Bozzano *et al.*, 2009; Bozzano *et al.*, 2019] allows to extend nominal component specification with error models. The extended model can be used for *fault tree analysis* to determine the cause of observed faults [Bittner *et al.*, 2016].

When diagnosing a robot agent, one is often interested in what happened rather than what is wrong. In *history-based diagnosis* [Baral *et al.*, 2000; Gspandl *et al.*, 2011; Delgrande and Levesque, 2012], the explanation is determined based on a history of actions and events. A possible explanation may contain faulty variants of the robot's actions (e.g., picking up the wrong object) and exogenous events (e.g., someone placing a new object on the table). A simplification for consistency-based diagnoses is to assume that a failed action may have any unknown effects [Witteveen *et al.*, 2005; Roos and Witteveen, 2009; Mühlbacher and Steinbauer, 2016]. Iwan [2001] describes an abductive diagnosis approach that uses diagnosis templates for compact representation to speed up the diagnosis process.

Listing 1: PDDL definition of a move action

```
(:action move
 :parameters (?r - robot
              ?from - location ?to - location)
 :precondition (at ?r ?from)
 :effect (and (not (at ?r ?from)) (at ?r ?to))))
```

While the above approaches determine diagnosis candidates with algorithms specific to the diagnosis problem, Mcllraith [1999] shows that the search for an explanatory diagnosis can also be modeled as a planning problem. Based on regression in the Situation Calculus, they determine an explanation by planning for the observations made by the system, which has also been extended to determine not one but k cost-optimal candidates [Sohrabi *et al.*, 2016]. By formulating the diagnosis problem as a PDDL planning problem, one can make use of the well-developed heuristics for planning [Sohrabi *et al.*, 2010; Katz *et al.*, 2018].

3 Approach

The goal is to find a unique explanation for observed faults for two reasons: (a) to regain a consistent belief state, and (b) to recover from any faults that may have occurred. An overview of the approach is shown in Figure 2.

After executing an action sequence and making any observation that contradicts the expected world state, we use the history of executed actions and a model of the underlying hardware platform to generate alternated histories that explain the observations and thus result in a consistent belief. In order to do so, we first model the robot platform using finite state machines, which we then use to model action variants that depend on an unexpected state of a platform component. We assume that faults are independent and at least some of them are recoverable. Next, we generate a PDDL domain from a previously prepared template (containing all actions, possible variants, and the platform model) that enforces the executed action history (or a variant thereof), which can then be used with a planner to generate explanation candidates for the observation. Note that any intermediate observations can be incorporated into the executed action history as exogenous actions which only assert the observed fact as effect. This includes any self-diagnostic information gathered by the platform components, which are modeled as exogenous component state transitions. Once we have obtained a set of explanation candidates, we use active diagnosis to determine the most informative sensing action. That sensing action is executed and any explanation candidates that contradict its sens-

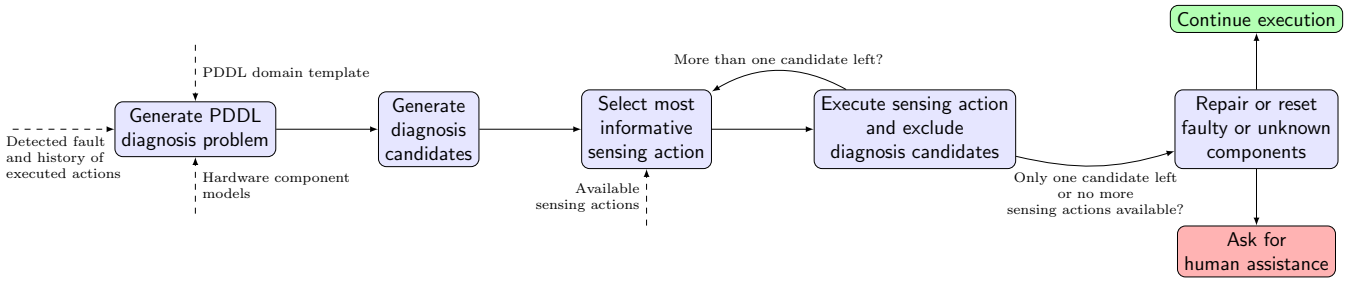
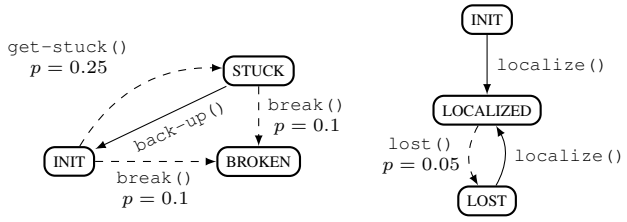


Figure 2: The diagnosis pipeline, starting with a detected fault and ending with a repair attempt.



(a) Navigation (b) Localization

Figure 3: Models of the robot hardware components

Listing 2: An exogenous state transition action in PDDL

```
(:action get-stuck
 :parameters ()
 :precondition (state NAVIGATION INIT)
 :effect (and (not (state NAVIGATION INIT))
              (state NAVIGATION STUCK))
              (increase (total-cost) 1)))
```

Listing 3: An action with hardware component constraints

```
(:action move
 :parameters (?r - robot
              ?from - location ?to - location)
 :precondition (and (at ?r ?from)
                    (state NAVIGATION INIT)
                    (state LOCALIZATION LOCALIZED))
 :effect (and (not (at ?r ?from)) (at ?r ?to)))
```

ing result are removed. This is a useful approach in scenarios where we can assume that the execution of sensing actions does not lead to more serious failures. We continue with the next sensing action until we have a unique explanation candidate or no other sensing action is possible. In this work, we assume perfect sensing in the context of active diagnosis, however in general active diagnosis is able to cope with sensing noise as well. Finally, based on that explanation, we update the agent's world model and try to recover from the fault. If this is not possible, the robot disables itself and asks a human for help.

3.1 Hardware Platform Model

We model the components of the robot platform, e.g., its navigation unit, based on finite state machines, similar to [Hofmann *et al.*, 2018]. We use two types of edges: (a) action edges that the agent can decide to execute, e.g., `localize()` to trigger the localization unit to re-initialize, and (b) exogenous event edges that are triggered by the environment, e.g., `lost()`, which causes the localization unit to lose its known pose. We also attach a probability to each exogenous edge which will be used during active diagnosis to assign probabilities to the explanation candidates and to determine the most promising sensing action.

Two example components that are used by the `move` action are shown in Figure 3. Figure 3a shows the navigation unit of the robot. By default, it is in the state `INIT` (we simplify the navigation and do not distinguish whether it is currently idle or moving). However, it may get stuck, e.g., because it hits an obstacle. If that is the case, the agent can abort the current navigation goal and decide to back up, which sets the navigation back to the `INIT` state. Similarly, the navigation may suffer from a hardware failure, which completely blocks the movement and from which the agent cannot recover. The localization unit in Figure 3b first needs to be initialized with the `localize()` action. While it is in use, it sporadically gets `lost()`, in which case it needs to be re-initialized with another `localize()` action.

3.2 Creating a PDDL Diagnosis Domain

With the goal to take a *diagnosis as planning* approach, we generate a PDDL domain that completely describes the diagnosis problem. We do this by (1) adding the nominal platform state to the precondition of actions, (2) adding actions for the exogenous events, following the platform model, (3) providing action variants of the original actions, depending on faulty states of the platform, and (4) enforcing the history, i.e., any solution of the problem must include each action (or one of its variants) of the originally executed plan.

In order to incorporate the platform model into the PDDL domain, we model the component states as PDDL domain objects and the current states as PDDL predicates `(state ?c - component ?s - state)`. Furthermore, we add exogenous events as PDDL actions, as shown in Listing 2 for the `get-stuck` transition of the navigation component. Using these PDDL platforms models, we can model the dependencies of the available actions on the hardware components. For example, the move action as defined in Listing 1 depends on the navigation component as well as the localization component, as defined in Figure 3. Therefore, we add preconditions for the required states of the component models to the PDDL model of the move action, as shown in Listing 3.

Listing 4: A faulty variant of the move action

```
(:action move-stuck
 :parameters (?r - robot ?from - location)
 :precondition (and (at ?r ?from)
                    (state NAVIGATION STUCK)
                    (state LOCALIZATION LOCALIZED))
 :effect ())
```

Listing 5: Order action template

```
(:action order-i
 :parameters ()
 :precondition (last-ACT_i PARAM_i)
 :effect (and (not (last-ACT_i PARAM_i))
              (next-ACT_i+1 PARAM_i+1)))
```

Creating Action Variants

In general, an action variant models a fault mode of an action in the original domain. While these are typically written ad-hoc based on the experience with failed execution runs, our platform models allow a more principled approach: As we have already added the expected state of a platform component to the original action, we can now also define an action variant for each state that the platform component may have reached by exogenous events. For example, the navigation component in Figure 3a contains an exogenous transition from the `INIT` state to the `STUCK` state, which represents the situation that the robot is not able to move. Therefore, as shown in Listing 4, we add an action variant `move-stuck` of the action `move` that models the behavior of the action if the navigation is stuck. In `move-stuck`, the precondition is adapted to require the navigation to be stuck. The effect models what happens if the robot executes a `move` action while it is stuck, which is nothing in this case. In order to ensure that the diagnosis process is able to determine the effects of all hardware faults that might have happened, it is beneficial to use the hardware models as a guide. Although different hardware faults may result in the same action variant, using the possible platform states as a guideline to create all variants will result in a complete description of the faulty behaviour.

Enforcing the History

We are not merely interested in any sequence of actions that is an explanation for the observation but rather only in action sequences that represent an adapted version of the original action sequence. For example, suppose an agent that executed the actions A and B, starting from a world state S, and observes the contradicting fact F. A planner might determine a plan consisting of the single action C as possible diagnosis. However, this does not take into account that the agent executed A and B. The only valid explanations are therefore sequences of actions that consist of variants of A and B and insertions of exogenous actions. Thus, we need to forbid solutions that do not result from possible variants of the original action sequence. We do this with *order actions*, which force the planner to insert variants of the executed actions in the right order.

For a given action sequence $h = \{a_1, \ldots, a_n\}$, we add an action `order-i` for each a_i, as shown in Listing 5, where `ACT_i` is replaced with the action type of a_i and `PARAM_i` is replaced with parameters of a_i. We replace `ACT_0` by `BEGIN` and `ACT_n+1` by `FINISHED`, where `(last-BEGIN)` is true initially and `(next-FINISHED)` is part of the goal. An example for such an order action is shown in Listing 6. We also add a new precondition and a new effect to each action and its variants to enforce the order actions, as shown exemplarily for the `move` action in Listing 7.

Listing 6: Grounded order action to enforce `grab` after `move`

```
(:action order-1
 :parameters ()
 :precondition (last-move R1 C-BS C-CS1)
 :effect (and (not (last-move R1 C-BS C-CS1))
              (next-grab R1 WP1 C-CS1))
```

Listing 7: A non-faulty action with order predicates.

```
(:action move
 :parameters (?r - robot
              ?from - location ?to - location)
 :precondition (and (next-move ?r ?from)
                    (at ?r ?from))
 :effect (and (not (at ?r ?from)) (at ?r ?to)
              (not (next-move ?r ?from))
                   (last-move ?r ?from))))
```

To reach a goal state, in which `(last-FINISH)` holds, the planner has to generate a sequence of the form $h' = \{o_0, a'_1, o_1, \ldots, a'_n, o_n\}$, where o_i is an order action and a'_j is either $a_j \in h$ or one of its faulty variants. Additional exogenous actions may be inserted anywhere in the sequence.

Action Variants Based on Changing Parameters

With the construction above, we can force any diagnosis produced by a planner to form a variant of the original action sequence, including the actions' parameters. However, when executing an action, the action's parameters may change without being noticed by the executive, thereby producing another variant of the action. Instead of specifying those variants explicitly, we obtain them by only fixing some of the parameters in the `(last-...)` and `(next-...)` predicates. In order to do so, we have to differentiate between parameters that would have caused the execution to fail in the first place if changed, and parameters that may change unknowingly.

Consider a `move` action that takes as argument the current position `?from` of the robot and the target machine `?to`. Assuming that the execution will fail if the robot is not actually starting from where it expects to be, then the starting position cannot have changed unnoticed. Therefore, `?from` has to be enforced by the `order` predicates. On the other hand, the robot might not end up where it expected, e.g., if the target machine was moved from its original location. In order to enable the diagnosis process to consider this possibility, the `?to` parameter should be replacable and not be part of the `order` predicates, as shown in Listing 7. The decision about which parameters have to be enforced in the history depends on the specific implementation and execution of the actions and can differ from platform to platform.

Action Variants Based on Conditional Effects

When designing the PDDL definition of an action with the intention to use it for planning, it is sufficient to model the

Listing 8: A put action with conditional effect. Hardware dependencies and order predicates are removed for visibility purposes.

```
(:action put
 :parameters (?r - robot
              ?mps - location ?wp - workpiece)
 :precondition (at ?r ?mps)
 :effect (and (when (holds ?r ?wp)
                    (and (not (holds ?r ?wp))
                         (at ?wp ?mps)))))
```

effects and preconditions of an action without any faults in mind. However, when executed on a real robot, an action's precondition may not be satisfied and still execute successfully, but only achieving its effects partially. As an example, if the robot executes `put` while not actually holding a workpiece, the execution may succeed (assuming it is not equipped with an appropriate sensor), but without the expected effects, as the workpiece will not be on the machine. Therefore, for diagnosis purposes, the effect needs to be modelled as conditional effect, depending on whether the robot is holding the workpiece. The corresponding PDDL action with this conditional effect is shown in Listing 8. Additionally, an action may also have undesired side effects, e.g., if there was already a workpiece on the machine before the robot executed `put`, it may knock that workpiece from the machine. Modelling those conditional and side effects is crucial for diagnosis, as only then the diagnosis process will be able to determine all possible outcomes of an action. Based on our experiences, we have identified some principles for modelling actions for diagnosis purposes: (1) The precondition must express the necessary condition for the action not to fail completely during execution. (2) Any other condition, which changes the effect of the action but does not affect its executability, must be used as a condition of a conditional effect. (3) An action may have additional side effects on other objects. Such effects need to be added as conditional effect and possibly require the other objects as additional parameters of the action.

Generating the PDDL Domain and Problem Description

We conclude this section by summarizing the process of generating the PDDL domain and problem description for a particular diagnosis problem. We start with a PDDL domain augmented by exogenous actions, action variants, and modified action preconditions that require the respective platform states, as described above. For a given initial state, an action history $h = \{a_1, \ldots, a_n\}$, and a set of observations, we generate the PDDL description as follows:

(1) For each a_i, insert an order action o_i into the domain.
(2) As initial state of the problem, use the state in which h started (i.e., the initial state of the original plan).
(3) Add `(next-BEGIN)` to the initial state to fulfill the precondition of the first order action o_0.
(4) Create the goal formula as a conjunction of the observed fault and `(last-FINISH)` to enforce the complete history h.

The resulting PDDL domain and problem description can then be used to determine possible diagnoses.

3.3 Top-k Planning for Diagnosis Generation

With the generated PDDL domain and problem description, we can use a PDDL planner to determine a possible history alternative that explains the observations. As we are interested in multiple explanation candidates, we will use a top-k planner [Katz *et al.*, 2018] that generates k cost-optimal solutions. As metric, we apply Ockham's razor and assume that diagnoses with fewer exogenous actions are more likely. We achieve this by applying a cost of 1 to each exogenous action, while each plan action (and its variant) have a cost of 0. By doing so, we will obtain explanations with the least possible amount of faults. It is possible that the real explanation contains more faults than other possible explanations and might not be included in the first k plans. However, since there might be an infinite number of possible explanations, we fix k such that the risk of missing the real explanation is negligible.

3.4 Active Diagnosis

Starting with a set of possible explanations generated by the top-k planner, we now determine which of those is the correct one. However, before we continue with excluding possible diagnosis candidates with active diagnosis, we filter this set of k possible solutions by excluding one of any two diagnosis candidates that would result in the same world state. To result in the same world state, two diagnosis candidates have to contain the same component faults. For example, if a possible solution for the given problem would be the insertion of an exogenous action e into the sequence, then the planner would treat all solutions that has e inserted at any place in the sequence as distinct solutions, even though some of them might result in exactly the same state. As we cannot distinguish those explanations, we only keep one of them.

Next, we apply *active diagnosis* to filter the explanation candidates. For a given set of candidates, we determine the common set of facts, the so-called *shared knowledge*. For every candidate that can be excluded by a sensing action, the shared knowledge will increase, as at least one disagreement on a fact will be resolved. Following Mühlbacher and Steinbauer [2014], we select the sensing action with the highest impact on the set of diagnosis candidates.

For a set of diagnosis candidates D and a grounded sensing action a that senses whether a predicate f is true, the action a splits D into D_+ and D_-. The subset D_+ contains all diagnosis candidates that are consistent with a positive outcome of $a(\vec{x})$ and D_- all candidates that are consistent with a negative outcome. Using the combined probabilities $p_{D_+} = \sum_{d \in D_+} p_d$ (similarly for p_{D_-}), we can calculate the entropy of a, which is a metric for the impact of a onto D:

$$E(D, a) = -\left(p_{D_p} \log_2 p_{D_p} + p_{D_n} \log_2 p_{D_n}\right)$$

To determine the probability p_d of a single diagnosis candidate d, we combine the probability of all exogenous actions F_d that are part of the diagnosis candidate d: $p_d = \prod_{f \in F_d} p_f$. Since exogenous actions represent exogenous event edges of a hardware component model, we can directly use the probabilities assigned to these exogenous edges. Therefore, the engineering knowledge encoded in the platform model directly affects the selection of the sensing action.

p_{failure}	0.12 %			0.19 %			0.27 %		
	∅	D	DP	∅	D	DP	∅	D	DP
# Plans	1874	1136	2102	1527	683	3312	2019	542	3766
# Failed	527	288	115	621	165	121	757	219	379
% Failed	28.1	25.3	5.5	40.6	24.1	3.6	37.5	40.4	10.1
# Diagnoses	0	56	49	0	20	66	0	22	111
# Repairs	0	0	31	0	0	38	0	0	69
# Disabled	0	0	26	0	0	32	0	0	50

Table 1: The number of plans, diagnoses, repairs, and robot disengagements in the simulated RCLL games, where ∅ stands for *no diagnosis*, D for *diagnosis without platform models*, and DP for *diagnosis with platform models*.

	Game Score w/o Diagnosis			Game Score w/ Diagnosis w/o platform			w/ platform		
p_{failure}	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
0.12%	71.31	6	260	76.66	12	217	160.94	93	229
0.19%	58	4	142	68.44	12	151	156.6	91	295
0.27%	32.75	0	98	37.88	0	144	109.5	10	174

Table 2: Game scores in simulated RCLL games of a team of robots without diagnosis, with diagnosis but without platform models, and with diagnosis including platform models. Failure was simulated, p_{failure} is the probability of a failure in an interval of 1 s.

After repeatedly using active diagnosis to determine the sensing action with the highest impact, executing the action, and excluding mismatching diagnosis candidates, the agent ends up with one or more remaining diagnosis candidates. Finally, the agent can integrate all facts that are true in all remaining diagnosis candidates into its world model. Similarly, it can remove all facts that are inconsistent with all remaining diagnosis candidates. Next, it can try to repair any hardware component that was determined to be faulty. If this is not possible, it may decide to call for human intervention or disable itself, since the robot is not operable any more.

4 Evaluation

We implemented the proposed diagnosis approach[1] and integrated it into the goal reasoning system CLIPS Executive [Niemueller *et al.*, 2019]. We evaluated our approach in a scenario from the RoboCup Logistics League (RCLL) [Niemueller *et al.*, 2014; Hofmann *et al.*, 2021]. In the RCLL, a team of three robots needs to operate machines to manufacture products. In order to do so, a robot needs to put workpieces on and grab workpieces from machines, and instruct the machines to process the workpiece. A team scores points upon successful delivery of a product.

We modeled the gripper, RGB/D camera, and RGB camera as hardware components. The gripper may become decalibrated or break completely. Also, the gripper's fingers may get stuck. Each camera might crash, which sometimes can be repaired by performing a reset. We extended an implementation of the RCLL in the Gazebo simulator [Zwilling *et al.*, 2014] with simulated failures of each of those components.

We simulated full RCLL games with simulated failures with diagnosis including platform models and compared it to diagnosis without platform models and no diagnosis at all. Both approaches try to recover a valid world model after observing an action failure. In cases where this is successful, the agents will try to continue the operation. Table 1 shows a summary of the results. We can see that with diagnosis, the percentage of successful plans increases. We can also see that diagnosis with platform models considerably outperforms diagnosis without platform models, for several reasons: First, even with a successful diagnosis, the system without platform models cannot find a repair action, because there is no known underlying cause; the action variant seems to have occurred randomly. In contrast, with platform models, the underlying reason (a failed platform component) can be diagnosed and repaired. Second, with platform models, the robot is able to detect when it is no longer operational, e.g., because its gripper is broken. In this case, it can disengage and stop interfering with the other robots. Table 2 shows the game scores in each configuration. As we can see, diagnosis with platform models increases the team's score significantly, especially in configurations where faults occur more often. Considering the running time t, both approaches perform similarly, with $t = (4.00 \pm 1.80)\,\text{s}$ for diagnosis without platform models and $t = (4.17 \pm 1.76)\,\text{s}$ for diagnosis with platform models.

5 Conclusion

We presented a novel approach to explanatory diagnosis for a mobile robot, based on the assumption that many execution failures are caused by issues of the robot hardware platform. To diagnose such a system, we model each robot component as a finite state machine and use them to design a diagnosis domain consisting of faulty action variants and exogenous actions that model state transitions in the platform model. We take a history-based diagnosis approach by forcing the PDDL planner to only consider variants of the original action sequence for explanation candidates, and we use a top-k planner to determine the k cost-optimal explanations. Next, we apply active diagnosis to decide which explanation is correct by determining sensing actions that differentiate between the possible explanations. As a result, we obtain an explanation for the observed failure. As this explanation is based on the model of the robot, the underlying hardware fault is part of the explanation. Therefore, the robot can try to repair itself by taking an appropriate action. We evaluated the approach in the context of the RoboCup Logistics League, which showed that using platform models results in more successful plans and ultimately in a better-performing team.

Acknowledgements

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grants *GL-747/23-1* and *GRK 2236/1*. This work is partially supported by the EU ICT-48 2020 project TAILOR (No. 952215).

[1]The code is open source and available at https://github.com/Sagre/GuidedExplanatoryDiagnosis.

References

[Baral *et al.*, 2000] Chitta Baral, Sheila McIlraith, and Tran Cao Son. Formulating diagnostic problem solving using an action language with narratives and sensing. In *Proc. of KR*, 2000.

[Bittner *et al.*, 2016] Benjamin Bittner, Marco Bozzano, Roberto Cavada, Alessandro Cimatti, Marco Gario, Alberto Griggio, Cristian Mattarei, Andrea Micheli, and Gianni Zampedri. The xSAP Safety Analysis Platform. In *Proc. of TACAS*, 2016.

[Bozzano *et al.*, 2009] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. The COMPASS Approach: Correctness, Modelling and Performability of Aerospace Systems. In *Proc. of SAFECOMP*, 2009.

[Bozzano *et al.*, 2019] Marco Bozzano, Harold Bruintjes, Alessandro Cimatti, Joost-Pieter Katoen, Thomas Noll, and Stefano Tonetta. COMPASS 3.0. In *Proc. of TACAS*, 2019.

[Chanthery and Pencolé, 2009] Elodie Chanthery and Yannick Pencolé. Principles of self-maintenance in an onboard architecture including active diagnosis. In *IJCAI SAS Workshop*, 2009.

[Cordier and Thiébaux, 1994] Marie-Odile Cordier and Sylvie Thiébaux. Event-Based Diagnosis for Evolutive Systems. In *Proc. of DX*, 1994.

[de Kleer and Williams, 1987] Johan de Kleer and Brian Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1), 1987.

[Delgrande and Levesque, 2012] James Delgrande and Hector Levesque. Belief revision with sensing and fallible actions. In *Proc. of KR*, 2012.

[Gspandl *et al.*, 2011] Stephan Gspandl, Ingo Pill, Michael Reip, Gerald Steinbauer, and Alexander Ferrein. Belief management for high-level robot programs. In *Proc. of IJCAI*, 2011.

[Hofmann *et al.*, 2018] Till Hofmann, Victor Mataré, Stefan Schiffer, Alexander Ferrein, and Gerhard Lakemeyer. Constraint-based online transformation of abstract plans into executable robot actions. In *AAAI Spring Symp.*, 2018.

[Hofmann *et al.*, 2021] Till Hofmann, Tarik Viehmann, Mostafa Gomaa, Daniel Habering, Tim Niemueller, and Gerhard Lakemeyer. Multi-agent goal reasoning with the CLIPS Executive in the Robocup Logistics League. In *Proc. of ICAART*, 2021.

[Iwan, 2001] Gero Iwan. History-Based Diagnosis Templates in the Framework of the Situation Calculus. In *KI 2001: Advances in Artificial Intelligence*. Springer, 2001.

[Katz *et al.*, 2018] Michael Katz, Shirin Sohrabi, Octavian Udrea, and Dominik Winterer. A Novel Iterative Approach to Top-k Planning. *Proc. of ICAPS*, 2018.

[Kuhn *et al.*, 2008] Lukas Kuhn, Bob Price, Johan De Kleer, Minh Do, and Rong Zhou. Pervasive diagnosis: The integration of active diagnosis into production plans. In *Proc. of DX*, 2008.

[Mcllraith, 1999] Sheila Mcllraith. Explanatory Diagnosis: Conjecturing Actions to Explain Observations. In *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*. Springer, 1999.

[Mühlbacher and Steinbauer, 2014] Clemens Mühlbacher and Gerald Steinbauer. Active Diagnosis for Agents with Belief Management. In *Proc. of DX*, 2014.

[Mühlbacher and Steinbauer, 2016] Clemens Mühlbacher and Gerald Steinbauer. Belief management using the action history and consistency-based-diagnosis. In *Proc. of DX*, 2016.

[Niemueller *et al.*, 2014] Tim Niemueller, Daniel Ewert, Sebastian Reuter, Alexander Ferrein, Sabina Jeschke, and Gerhard Lakemeyer. RoboCup Logistics League sponsored by Festo: A competitive factory automation testbed. In *RoboCup 2013: Robot World Cup XVII*, 2014.

[Niemueller *et al.*, 2019] Tim Niemueller, Till Hofmann, and Gerhard Lakemeyer. Goal reasoning in the CLIPS Executive for integrated planning and execution. In *Proc. of ICAPS*, 2019.

[Poole, 1989a] David Poole. Explanation and prediction: An architecture for default and abductive reasoning. *Computational Intelligence*, 5(2), 1989.

[Poole, 1989b] David Poole. Normality and faults in logic-based diagnosis. In *Proc. of IJCAI*, 1989.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1), 1987.

[Roos and Witteveen, 2009] Nico Roos and Cees Witteveen. Models and methods for plan diagnosis. *Autonomous Agents and Multi-Agent Systems*, 19(1), 2009.

[Sampath *et al.*, 1995] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9), 1995.

[Sampath *et al.*, 1998] M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event systems. *IEEE Transactions on Automatic Control*, 43(7), 1998.

[Sohrabi *et al.*, 2010] Shirin Sohrabi, Jorge Baier, and Sheila McIlraith. Diagnosis as planning revisited. In *Proc. of KR*, 2010.

[Sohrabi *et al.*, 2016] Shirin Sohrabi, Anton Riabov, Octavian Udrea, and Oktie Hassanzadeh. Finding diverse high-quality plans for hypothesis generation. In *Proc. of ECAI*, 2016.

[Witteveen *et al.*, 2005] Cees Witteveen, Nico Roos, Roman van der Krogt, and Mathijs de Weerdt. Diagnosis of single and multi-agent plans. In *Proc. of AAMAS*, 2005.

[Zwilling *et al.*, 2014] Frederik Zwilling, Tim Niemueller, and Gerhard Lakemeyer. Simulation for the RoboCup Logistics League with real-world environment agency and multi-level abstraction. In *RoboCup 2014: Robot World Cup XVIII*, 2014.