FailRecOnt – An Ontology-Based Framework for Failure Interpretation and Recovery in Planning and Execution

Mohammed Diab[1], Mihai Pomarlan[2], Stefano Borgo[3], Daniel Beßler[4], Jan Rossel[1], John Bateman[2] and Michael Beetz[4]

[1]*Institute of Industrial and Control Engineering, Universitat Politècnica de Catalunya*
[2]*Faculty of Linguistics and Literature, University of Bremen*
[3]*Laboratory for Applied Ontology (LOA), ISTC CNR*
[4]*Institute for Artificial Intelligence, University of Bremen*

Abstract

Autonomous mobile robot manipulators have the potential to act as robot helpers at home to improve quality of life for various user populations, such as elderly or handicapped people, or to act as robot co-workers on factory floors, helping in assembly applications where collaborating with other operators may be required. However, robotic systems do not show robust performance when placed in environments that are not tightly controlled. An important cause of this is that failure handling often consists of scripted responses to foreseen complications, which leaves the robot vulnerable to new situations and ill-equipped to reason about failure and recovery strategies. Instead of libraries of hard-coded reactions that are expensive to develop and maintain, more sophisticated reasoning mechanisms are needed to handle failure. This requires an ontological characterization of what failure is, what concepts are useful to formulate causal explanations of failure, and integration with knowledge of available resources including the capabilities of the robot as well as those of other potential cooperative agents in the environment, e.g. a human user. We propose the FailRecOnt framework as a step in this direction. We have integrated an ontology for failure interpretation and recovery with a contingency-based task and motion planning framework such that a robot can deal with uncertainty, recover from failures, and deal with human-robot interactions. A motivating example has been introduced to justify this proposal. The proposal has been tested with a challenging scenario.

1. Introduction

Robotic manipulators have established themselves in industry as reliable tools for some complex but repetitive tasks, but this reliability depends on a carefully designed and controlled environment. Domestic settings do not have this property, and despite interest in service robots working in the home, today's robots have too brittle performance because of insufficiently robust failure handling.

RobOntics 2021: 2nd International Workshop on Ontologies for Autonomous Robotics, held at JOWO 2021: Episode VII The Bolzano Summer of Knowledge, September 11–18, 2021

mohammed.diab@upc.edu (M. Diab); mihai.pomarlan@uni-bremen.de (M. Pomarlan); stefano.borgo@cnr.it (S. Borgo); danielb@uni-bremen.de (D. Beßler); jan.rosell@upc.edu (J. Rossel); bateman@uni-bremen.de (J. Bateman); beetz@uni-bremen.de (M. Beetz)

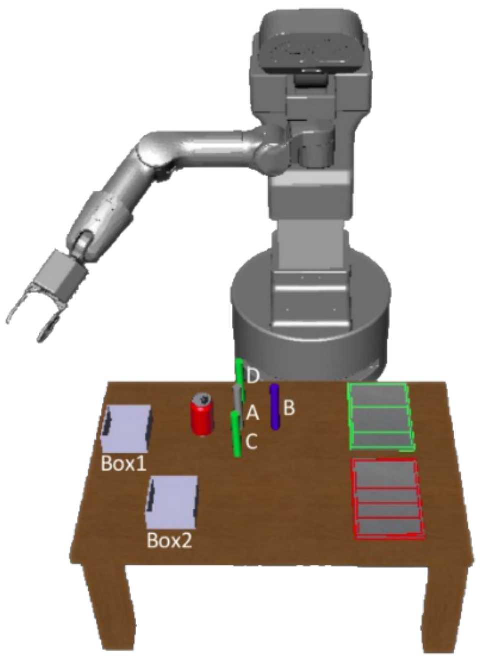

Figure 1: A motivating example to justify our approach.

This paper presents a framework for autonomous behavior which is a step towards a more automated, reasoning and knowledge-driven approach to failure handling. Such an approach needs a concept of what "failure" means, what kinds of failures might happen and why, and what an appropriate response might be. The reasoning must also be integrated into the perception-action loop of the robot, and able to guide a plan repair process to resume or repeat a task after a failure. The main **contributions** of this paper are:

- *Ontology formalization:*[1] Formal definition of concepts describing failures in terms of causal mechanism, location, time of performance, and functional considerations including concepts to describe recovery strategies according to the available resources and the required plan repair operations.
- *State interpretation based on observation:* Integration of sensing action and reasoning action with failure knowledge. Perception flags abnormal events and describes their nature in terms of failure conceptualizations.
- *Plan repair guided by reasoning:* Failure descriptions inform reasoning processes that propose possible recovery strategies via classification, which allows posing repair/replanning queries to a contingency task planner.

The integration of our proposed ontology-based reasoning mechanism and contingent-based

[1]https://github.com/ease-crc/failrecont

planning has been implemented for a robot and demonstrated in a test scenario motivated by previous work [1]. The task is to store an object (a cylinder labeled A) in a given tray according to its color. Cylinder A may be among other objects, each of which have to be manipulated according to its nature. Figure 1 shows the initial belief state of the mobile manipulation problem, where cylinder A is painted in gray because its color is uncertain (it could be red or green), it is not known whether the can is filled or not (and hence whether the can should be pushed or picked instead), nor if the containers are open or closed (and hence if the placement of an object inside the box can be directly performed or not). We enhance our previous work by providing an ontological way for failure interpretation and recovery strategies to allow the robot to fix its plan whether a human is available to assist or not.

2. Related Work

An influential, though informal, conceptualization of system dependability is due to Laprie [2]. It defines failures as a system being unable to operate according to nominal specifications; failures are caused by error states, in turn caused by faults due to the system, its environment, or its design. Carlson and Murphy [3] enrich this conceptualization with more human-caused faults and study failures in deployments of search-and-rescue unmanned ground vehicles. Honig and Oron-Gilad [4] further enrich the human-caused fault side of the conceptualization and survey existing failure classifications for robotic systems, especially those due to faults in human-robot interaction. Ross [5] builds upon this framework and defines errors in terms of recovery strategies. Unlike our approach, Laprie's conceptualization is informal, i.e., meant for use by human researchers and engineers, and treats failures as disruptive and rare events in the life of a system. What we propose is a formalized account of task failure, i.e., an ontology that a machine can reason with and use in the frequent eventuality of a particular task – not a system – failing.

Early detection of erroneous robot behavior has been pursued by model-free methods with classifiers trained on data from regular operation [6], hidden Markov models [7], predictive models [8], and models trained from simulation [9, 10].

An early example of fault-tolerant planning is provided by Jensen et al. [11], in which actions may have likely/unlikely side-effects, some of which lead to failure, and investigate algorithms for finding plans that can recover from some number of such failures.

Our first trial toward failure interpretation and recovery was presented in [1] and [12]. The former is a contingent-based task and motion planning approach; the basic Contingent-FF planner [13] has been modified to include human-robot collaboration and perception-based state observation. The latter work describes several sources of failures in automated planning and execution – geometric, hardware-related, software-related. This ontology has been formalized based on foundations such as DUL [14] and SUMO [15] in order to generalize the approach and allow it to be widely used in robotic systems. However, the ontology did not describe recovery strategies that would let the plan automatically repair itself.

Other investigations of interpreting failures have been done, e.g. [16], in which three layers of knowledge are proposed. Although that study shows how assumptive planning provides a single mechanism for interpretation under uncertainty, it is less applicable to robot manipulation planning in cluttered environments. We compare this approach with ours in Sec. 7.

3. Ontological Modelling of Failures

3.1. Conceptualization

Ontologically speaking, the labels “success” and “failure” are perspectival. They do not apply to actions or events per se but are used by an agent to evaluate the result of its action based on its expectations. The robot acts with the aim to bring about a desired state of the world, therefore it needs to establish whether the performed action and the resulting state are as expected. A failure is a lack of correspondence between the expectation and the actual state of the world.

In this paper we concentrate on an actual robot platform, and present both robot-specific and generic knowledge about actions to derive failure detection and recovery decisions. This illustrates how the approach is applicable to a broad class of cases and how it can work for other robots or scenarios provided adaptations to the specifics of a robot platform are implemented.

3.2. Formalization

We use the DUL ontology as a foundation[2]. However, we need to enrich the DUL terminology to cover our scenario. In particular, in DUL the notion of *Situation* is agent- and language-independent. In robotics one need to talk of epistemic situations (“e-situations”), i.e., situations as accessible to a specific robot, relative to its capabilities to sense and describe the world.

We write *Situation*, capitalized and italicized, for the general DUL concept, and “e-situation” to refer to the robot’s specialized concept. Thus, we call “e-situation” a fragment of physical reality (a “state of the world”) as it can be sensed and processed by the robot.

By construction, to each e-situation corresponds a different situation description: an expression in the robot’s language giving the maximal knowledge the robot may have about the e-situation. An e-situation S satisfies a description D when S makes the description D true.

We need to introduce another concept, that of expected transformation. An expected transformation is a triple <S_i; Act_j; S_f> stating that the execution of action Act_j at state S_i is expected to terminate in state S_f. Transformations are robot-dependent and normative. They may be regularly updated as a result of experience in learning robots.

The execution of an action Act_j at state S_i is deemed successful if it is an instantiation of the expected transformation <S_i; Act_j; S_f>, i.e., if the ending state satisfies S_f. A failure occurs when an action instantiates a triple <S_i; Act_j; S'> such that (a) the triple is not an expected transformation, and (b) for any expected transformation <S_i; Act_j; S_f>, S' and S_f cannot both hold in any possible world state.

We will next present how the above discussion is formalized in description logic axioms. We introduce a relation, *manifestsIn*, of domain *Situation* and range *Event*. In our model, *State* is a subclass of *Event*. A DUL *Situation satisfies* some *Description*. The correspondences between a situation description and an e-situation as state of the world are represented by the chain $isSatisfiedBy \circ manifestsIn$.

The expected transformation <S_i; Act_j; S_f> becomes in DUL terms an *Action* that *isEventIncludedIn* some *Situation* which *hasPostcondition* a *Situation* corresponding to S_f.

[2]DUL is formulated in a fragment of description logic which we therefore also use for our work here; see appendix https://sir.upc.edu/projects/ontologies/ for a short introduction to its syntax and semantics.

We derive our *Failure* concept from DUL's *EventType*. A *Failure classifies* only *Actions* (*Events* with an *Agent* enacting a *Plan* pursuing a *Goal*). Further, *Failure classifies* only *Actions* for which the *Agent* knows of inconsistencies between expected and actual postconditions. The mismatch is represented by a new concept we defined, *NonrealizedSituation*, i.e. *Situations* without *manifestsIn* links to any *Events*:
$PlanExecution \sqsubseteq Situation$ (DUL axiom)
$NonrealizedSituation \sqsubseteq Situation$
$NonrealizedSituation \sqcap (\exists manifestsIn.Event) \sqsubseteq \bot$
$FailedPlanExecution \equiv PlanExecution \sqcap$
$\quad \exists hasPostcondition.NonrealizedSituation$
$Failure \equiv EventType \sqcap \forall classifies.(Action \sqcap$
$\quad \exists isEventIncludedIn.FailedPlanExecution)$

We use a feature of open-world semantics: facts not asserted are not false, but unknown. A robot may know of a *Situation* without any *manifestsIn* links to *Events* in the robot's history but, unless that *Situation* is explicitly asserted to be a *NonrealizedSituation*, the robot maintains the possibility that the *Situation* could manifest into an *Event*, e.g. one in the future. Decisions on whether a *Situation* corresponds to some observed *Event* should come from a module such as perception. What our ontology describes is how the decision from perception should be formalized so that it becomes accessible to further reasoning.

The input to reasoning about failure recovery is, in our approach, a knowledge graph containing a snapshot of the robot's history up to a *Failure* – the past observed *Events*, their participants, and any *Situation* they might relate to.

4. From Failure Understanding to Recovery Strategies

4.1. Causal Explanations for Failure

The main question of an *Agent* when detecting a *Failure* is what to do about it. The answer is often simple: repeat the last action. Such "reflex" answers are often not good enough, but necessary for an *Agent* to act fluidly.

However, simple reflexes will not always work. E.g., a robot wants to turn on a blender, so it pushes a button; the blender doesn't start. If the blender is not plugged in, repeated pushing does nothing. To infer how to act, the *Agent* needs a causal explanation for why the blender fails to start. One of the goals of our failure ontology is to define concepts for creating such causal explanations. Our approach is inspired by Galton's model of causal relations [17].

However, because of the nature of *Failures*, we need to push beyond this model. We are interested in prevention, and causal explanations of why something did not happen – why the actual state of the world does not match the *Agent*'s expectations. The *NonrealizedSituation* concept is our way around one difficulty identified by Galton [17]: a prevented *Event* did not occur, so there should be no token associated with it in the robot's history. The expectation for a state however did exist, modelled as a *Situation*, and as a *NonrealizedSituation* when it matches no actual *Events*. We define *prevents* as a relation from a *Situation* which manifests in some *Event* to a *NonrealizedSituation*:

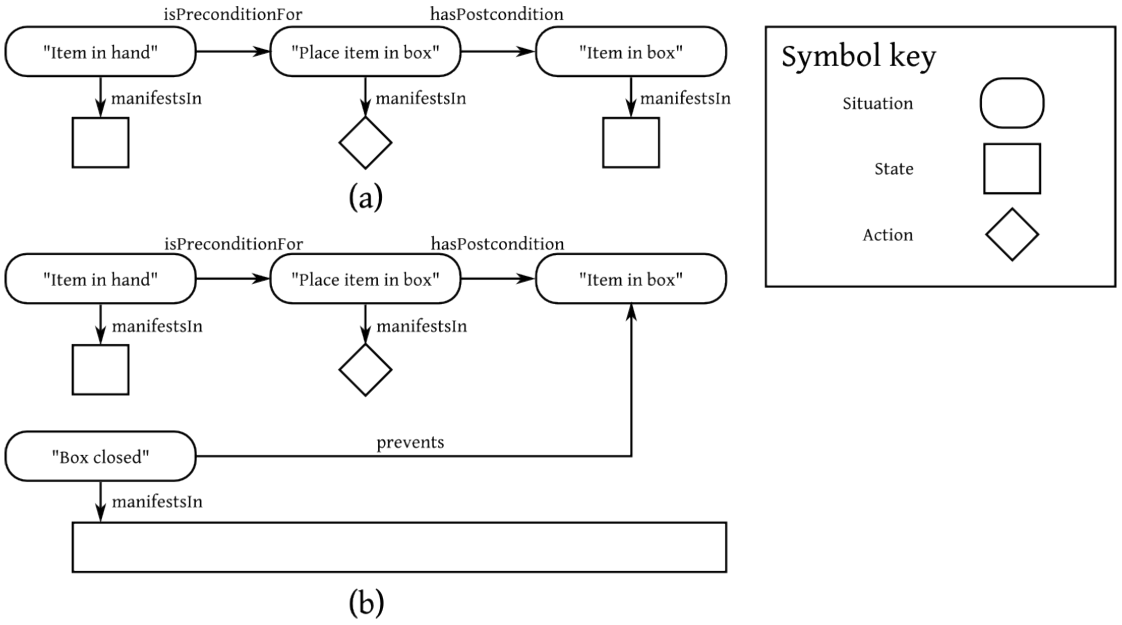

Figure 2: Examples illustrating our approach to represent task executions and failures: (a) a successful task execution results in its postconditions manifesting in a state of the world; (b) a failed task execution has unmanifested postconditions.

$prevents^{-} \equiv isPreventedBy$
$\exists prevents.\top \sqsubseteq Situation \sqcap (\exists manifestsIn.Event)$
$\exists isPreventedBy.\top \sqsubseteq NonrealizedSituation$

4.2. Recovery Strategies

A recovery strategy is a method a robot can apply to repair or reconstruct a plan that resulted in a failure. Replanning is a time-consuming operation, so several heuristics might be used by the robot in different cases. It is relevant to decide, based on knowledge about the failure, which heuristic may be appropriate. We will say a *RecoveryStrategy recoversFrom* some *Action* classified as a *Failure*:
$RecoveryStrategy \sqsubseteq \exists recoversFrom.(Action \sqcap$
$(\exists isClassifiedBy.Failure))$

We can axiomatically encode that “repeat last action”is not appropriate when the failure has a sustained cause:
$RepeatLastAction \sqsubseteq RecoveryStrategy$
$RepeatLastAction \sqcap \exists recoversFrom.$
$(\exists isClassifiedBy.SustainedFailure) \sqsubseteq \bot$

Depending on the nature of a sustained failure’s cause, we can use the ontology to formulate new subgoals for a *RemoveFailureCause* strategy by checking which of the robot-known situations get classified as *UnrealizedPrecondition*:

$$\begin{aligned}&UnrealizedPrecondition \equiv NonrealizedSituation \sqcap\\&\quad \exists isPreconditionOf.(\exists isSettingFor.\\&\qquad (\exists isClassifiedBy.SustainedFailure))\end{aligned}$$

and hence the unrealized preconditions become goals for new planning queries.

Our ontology defines several other recovery strategies to organize a robot's response. These concepts are a common vocabulary between robot modules, even if not all reasoning required to decide on a strategy's applicability can be captured in OWL-DL.

In order to check whether the *prevents* relations hold or not – so as to ascertain whether *impedes* holds or not – we would have to defer to other reasoning modules.

4.3. Competency Questions

Our ontology plays two roles. First, it provides a common vocabulary for a robot's subsystems to assert facts, enabling an integrated understanding of an event. Second, it enables reasoning with those facts and selecting appropriate recovery strategies. To guide development towards achieving these roles, we use a standard technique based on competency questions [18]. Answering the competency questions requires a collaboration between several robot subsystems and an OWL DL reasoner operating on the ontology axioms and the facts asserted by the modules. The DL reasoning queries are classification or consistency queries: given what is known, what is entailed about the categorization of a failure or its recovery? What cannot be ruled out? We will go through these competency questions below.

Q1: What kind of Failure resulted from an Action? The robot system as a whole needs an integrated understanding of the error it detects, which is enabled by our ontology through a classification query returning which types of failure are entailed by the information coming from subsystems.

Q2: Is a Failure causing a problem for subsequent activities? Our ontology provides the vocabulary to organize situations and goals in terms of precondition relations, and for robot subsystems to assert facts about what situations prevent which others.

Q3: Why did the Failure happen? We are interested in whether the failure is of type *SustainedFailure*, indicating a recovery strategy addressing its cause is necessary. This is a consistency query asking whether, given prevention relations asserted by the robot's subsystems while addressing the previous question, it is possible the observed failure is not a *SustainedFailure*.

Q4: What recovery strategy is appropriate to a failure? A DL classification query can be used to find out what is entailed about the recovery strategy given the nature of the failure as identified in the previous competency questions. For a more permissive alternative, consistency queries can be used to find out what recovery strategies are not ruled out.

5. The Failure-Module in the Robot Architecture

Autonomous manipulation tasks require cooperation between perception, knowledge representation and reasoning, and planning at both symbolic and geometric levels.

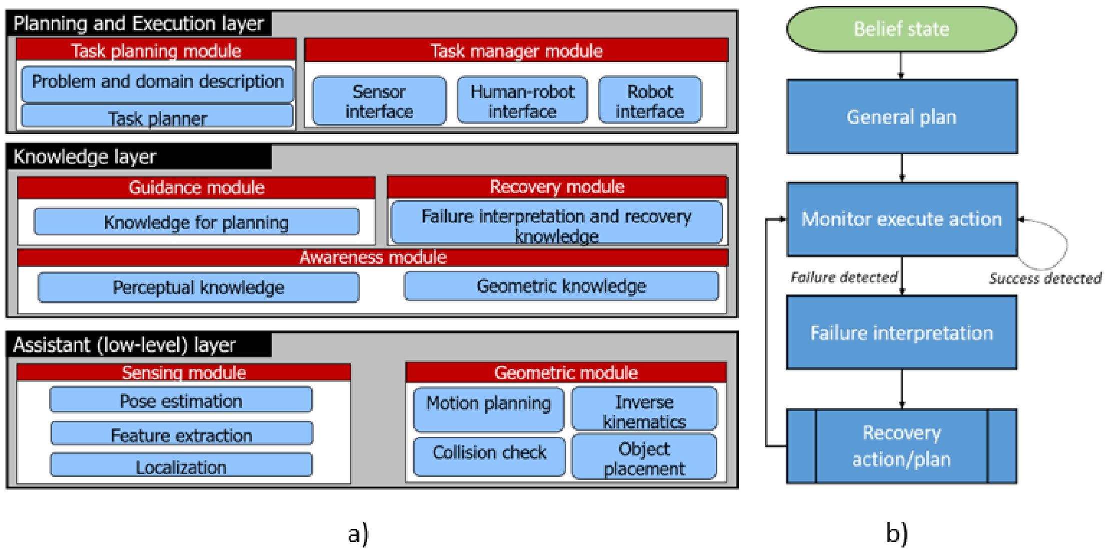

Figure 3: a) FailRecOnt - The proposed framework components. b) Flowchart of the proposed framework.

5.1. Framework Overview

The proposed framework – FailRecOnt – is composed of three main layers, as shown in Fig. 3-a): planning and execution, knowledge, and assistant (low-level) layer.

The planning and execution layer contains two modules, the task planning and the task manager modules. The former includes a task planner to compute a sequence of manipulation actions to be done, which requires a problem and domain description to set the initial scene, including world entities' state, and the goal state. The latter provides interfaces to communicate with the agents/operators (e.g., robots, humans or sensors). It also keeps monitoring the executed actions and it returns a failure signal to the recovery module if an error occurs. Moreover, it has a procedural structure for each step of an action. This structure is formally defined as a workflow that can be automatically executed through interfacing existing robot control software components. Workflows can be automatically executed through interfacing existing software components, e.g., executing a *pickUp* requires a call to the Inverse Kinematics (IK) module to check reachability for grasping the objects then finding a collision-free path towards a grasping configuration. A *putDown* action must also search for available placement room.

The knowledge layer contains a set of knowledge to guide the planning and execution layer.

1. Guidance module, containing planning information, e.g., related to geometric skills, tuned based on the robot's experience. It answers questions such as *How to grasp an object? What are the constraints of a task?* If the robot must store an item in a box, the robot must reason about which type of grasp is feasible (side or top-grasp).
2. Recovery module, providing knowledge to interpret failures and propose recovery strategies like: 1) asking a human for assistance for unsolvable tasks by the robot, 2) guiding the robot to autonomously recover itself, for instance by calling a sensing module to sense

the current state of the world or repeat the same action with another parameter (repeat a grasping action with different angle).

3. Awareness module, containing geometric knowledge for reasoning to check action feasibility, and perceptual knowledge to assist the robot to figure out its workspace. It assists the robot to figure out which are the proper algorithms and parameters to be used for the available sensors in order to extract data. An example for this knowledge is the Perception and Manipulation Knowledge (PMK) presented in detail in [19].

Finally, the assistant layer provides the low-level modules that allow dealing with:

1. perception issues, e.g. finding out which sensors can be used for a sensing action in a given situation, which are dealt by the sensing module.
2. geometric issues, e.g. determining if a configuration is collision-free or if an inverse kinematic solution exists for a gripper pose, which are dealt by the geometric module.

5.2. Framework Flowchart

The flowchart shown in Fig. 3-b) illustrates how the modules in the framework are integrated for such tasks. First, the robot needs to describe the initial and goal states. The initial state can be obtained from perception informed by perceptual knowledge [19, 20]. Then, the task planner computes the sequence of actions to be executed, which is a general plan. This sequence is obtained at a symbolic level, without any geometric considerations.

The generated plan is associated with geometric reasoning to generate a feasible path for each action. When these paths are executed, failures during execution may occur due to, for instance, uncertainties in motion planning or sensory extraction. The robot needs to monitor the outcome of each manipulation action in the execution phase. For the monitoring process, sensing and reasoning actions are required. The sensing action is required to detect the failures without any consideration of interpretation process. The reasoning action is required to interpret potential failures, their possible causes, and potential recovery strategies.

The recovery strategies let the robot react to changes in the environment that lead to failures. Moreover, the general plan generated based on the initial state is maintained as-is, to avoid re-planning based on the environmental changes.

6. Case Study

To illustrate our proposal some simulation examples are performed using The Kautham Project for geometric planning [21], which has the ability to report geometric failures. Ontologies are encoded using the Web Ontology Language (OWL) [22]. The ontologies are designed using the Protégé [3] editor. The modified contingent-based planning presented in [1] is used to generate a plan. Reasoning for failure interpretation and the potential recovery action(s) is integrated with the planner through ROS (Robot Operating System) [4] as a service-client communication.

[3](http://protege.stanford.edu/)

[4]https://www.ros.org

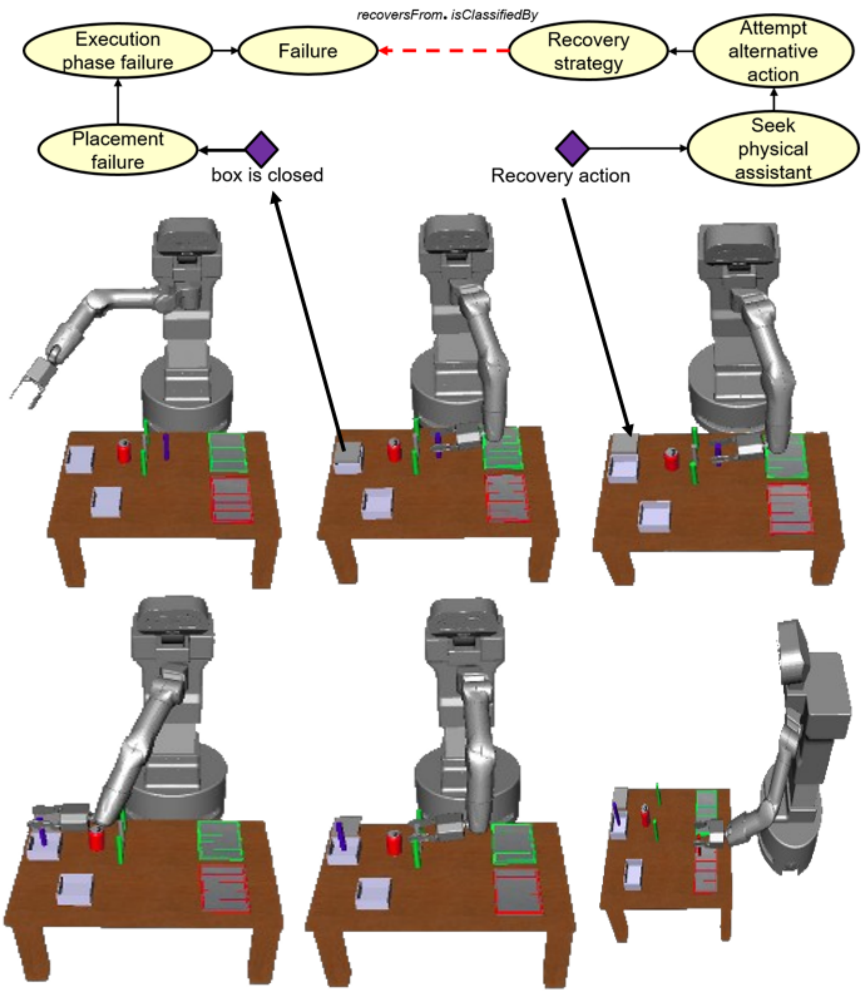

Figure 4: The sequence of snapshots from the planning process and its relation with FailRecOnt. The upper image depicts how the failure *box is closed* is classified in the simulated example. The bottom image shows the manipulation example where the goal is to transfer the gray cylinder (labeled as A) to one of the trays w.r.t its color.

6.1. Action Description

The following actions types are considered: manipulation, sensing, and reasoning.

Manipulation actions – pushing, transporting objects, opening – require motion, and can be done by either the robot or a person.

Sensing actions – checking color, pose, status of a container – do not involve motion, and are devoted to observing object status. The observation is done at run-time.

The reasoning actions are devoted to interpreting execution phase failures and providing a recovery strategy. The reasoning process is done at run-time.

The contingent Fast-Forward planner uses manipulation actions to find the conditional tree of

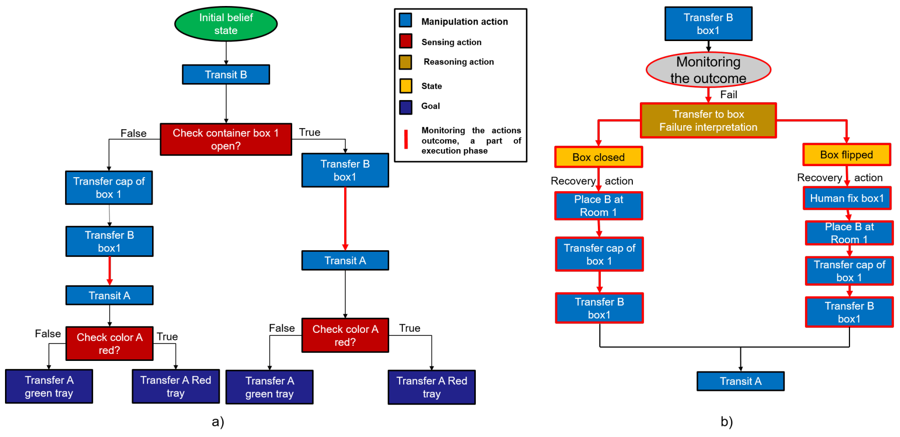

Figure 5: The conditional plan results from the planning process. a) flowchart describing the plan obtained by the contingent FF. b) flowchart describing the monitoring process of the manipulation action *Transfer-B-box1*.

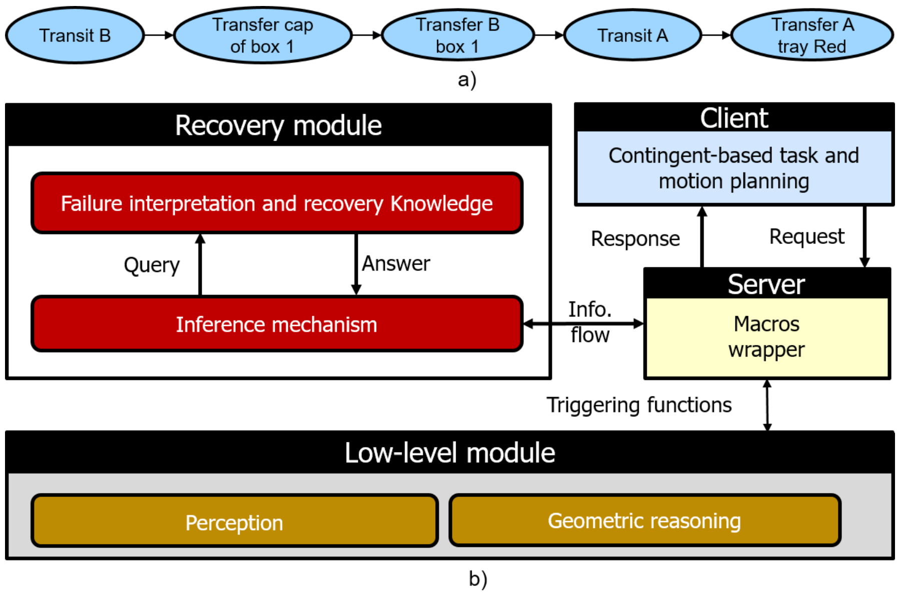

Figure 6: a) The executable plan. b) The integration of the FailRecOnt. PS: Macros wrapper is used to ground symbolic information from knowledge.

plans to execute the task, with reasoning actions associated to monitoring the execution of the manipulation actions.

6.2. The Integration of the Recovery Module with a Conditional Plan

The complete conditional tree of plans is shown in Fig. 5. The plan contains branching nodes in which sensing and/or reasoning actions are assigned to monitor manipulation action outcomes in a semi-automatic way. To reduce computational cost we only assign sensing/reasoning actions to manipulations expected to have a high probability of failure.

The recovery strategies have been provided based on the current status of the box as shown in Fig. 5-b). Recovery strategies could be to ask for help, or the robot recovers the failure by itself.

7. Discussion

The advantages of using FailRecOnt framework are: 1) Generality: FailRecOnt provides common vocabularies for failure interpretation and recovery strategies in the robotics domain. 2) Shareability: FailrecOnt can be exploited in shared tasks with other operators, e.g. robots or humans, and using heuristic-based or logic-based planning approaches. 3) Interoperability: FailRecOnt can be used widely in the robotics domain at both planning and execution phases.

In an alternative approach ([16]), a new state of the world is assumed when a failure occurs, then the assumption is used by a task planner. This approach works well for tasks that involve testing options in some order of expected value and where being wrong is safe. However, such assumptions about the world state are not always necessary to proceed.

Our approach allows reasoning to more carefully characterize what can be proven about a failure and what constraints may be imposed on recovery strategies, without making assumptions. Further, we have shown, using a manipulation task in a cluttered environment, how to integrate our failure modelling with plan repair techniques rather than relying only on replanning and/or asking for human assistance (although these are also available recovery strategies).

8. Conclusion and Future Work

Robots, like any other agent, sometimes fail. Knowledge-based robots can recover from failure by reasoning whether to try again, try something else, or move to other tasks. We argued that the choice must integrate the conceptual (ontology), the planning (task) and the execution (feasibility) levels. This work raises many issues that we aim to expand in the future: to deepen the ontological module, enrich the causal explanatory module, improve the search for an optimal match between what is known about detected failures and recovery strategies, optimize the interconnections among FailRecOnt submodules, and find a proper way to handle unknown/unmodeled failures.

Acknowledgments

This work was partially funded by the Spanish Government through the project PID2020-114819GB-I00, by Deutsche Forschungsgemeinschaft (DFG) through the Collaborative Research Center 1320, EASE, and by European CSA project OntoCommons (GA 958371). M. Diab is supported by the Spanish Government through the grants FPI 2017.

References

[1] A. Akbari, M. Diab, J. Rosell, Contingent task and motion planning under uncertainty for human–robot interactions, Applied Sciences 10 (2020) 1665.

[2] J. Laprie, Dependable computing and fault tolerance : Concepts and terminology, in: Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'., 1995, pp. 2–. doi:10.1109/FTCSH.1995.532603.

[3] J. Carlson, R. Murphy, How ugvs physically fail in the field, Robotics, IEEE Transactions on 21 (2005) 423 – 437. doi:10.1109/TRO.2004.838027.

[4] S. Honig, T. Oron-Gilad, Understanding and resolving failures in human-robot interaction: Literature review and model development, Frontiers in Psychology 9 (2018) 861. URL: https://www.frontiersin.org/article/10.3389/fpsyg.2018.00861. doi:10.3389/fpsyg.2018.00861.

[5] R. Ross, R. Collier, G. O'Hare, Demonstrating social error recovery with agentfactory, in: International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2004, pp. 1424–1425. doi:10.1109/AAMAS.2004.103.

[6] R. Hornung, H. Urbanek, J. Klodmann, C. Osendorfer, P. van der Smagt, Model-free robot anomaly detection, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2014, pp. 3676–3683. doi:10.1109/IROS.2014.6943078.

[7] D. Azzalini, A. Castellini, M. Luperto, A. Farinelli, F. Amigoni, Hmms for anomaly detection in autonomous robots, in: International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2020.

[8] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, S. Schaal, Skill learning and task outcome prediction for manipulation, in: 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 3828–3834. doi:10.1109/ICRA.2011.5980200.

[9] A. Haidu, D. Kohlsdorf, M. Beetz, Learning action failure models from interactive physics-based simulations, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 5370–5375. doi:10.1109/IROS.2015.7354136.

[10] A. S. Bauer, P. Schmaus, F. Stulp, D. Leidner, Probabilistic effect prediction through semantic augmentation and physical simulation, in: IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 9278–9284. URL: https://elib.dlr.de/134290/.

[11] R. M. Jensen, M. Veloso, R. E. Bryant, Fault tolerant planning: Toward probabilistic uncertainty models in symbolic non-deterministic planning, in: ICAPS, 2004.

[12] M. Diab, M. Pomarlan, D. Beßler, A. Akbari, J. Rosell, J. Bateman, M. Beetz, An ontology for failure interpretation in automated planning and execution, in: Iberian Robotics conference, Springer, 2019, pp. 381–390.

[13] J. Hoffmann, R. Brafman, Contingent planning via heuristic forward search with implicit belief states, in: Proc. ICAPS, volume 2005, 2005.

[14] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, WonderWeb Deliverable D18 Ontology Library, Technical Report, IST Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web, 2003.

[15] I. Niles, A. Pease, Towards a standard upper ontology, in: Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001, ACM, 2001, pp. 2–9.

[16] M. Hanheide, M. Göbelbecker, G. S. Horn, A. Pronobis, K. Sjöö, A. Aydemir, P. Jensfelt, C. Gretton, R. Dearden, M. Janicek, H. Zender, G.-J. Kruijff, N. Hawes, J. L. Wyatt, Robot task planning and explanation in open and uncertain worlds, Artificial Intelligence 247 (2017) 119–150. URL: https://www.sciencedirect.com/science/article/pii/S000437021500123X. doi:https://doi.org/10.1016/j.artint.2015.08.008, special Issue on AI and Robotics.

[17] A. Galton, States, processes and events, and the ontology of causal relations, in: Formal Ontology in Information Systems - Proceedings of the Seventh International Conference, FOIS 2012, Gray, Austria, July 24-27, 2012, 2012, pp. 279–292.

[18] M. Uschold, M. Gruninger, Ontologies: principles, methods and applications, The Knowledge Engineering Review 11 (1996) 93–136. doi:10.1017/S0269888900007797.

[19] M. Diab, A. Akbari, M. Ud Din, J. Rosell, PMK - a knowledge processing framework for autonomous robotics perception and manipulation, Sensors 19 (2019). URL: http://www.mdpi.com/1424-8220/19/5/1166. doi:10.3390/s19051166.

[20] M. Diab, M. Pomarlan, D. Beßler, A. Akbari, J. Rosell, J. Bateman, M. Beetz, Skillman - a skill-based robotic manipulation framework based on perception and reasoning, Robotics and Autonomous Systems (2020) 103653. URL: http://www.sciencedirect.com/science/article/pii/S0921889020304930. doi:https://doi.org/10.1016/j.robot.2020.103653.

[21] J. Rosell, A. Pérez, A. Aliakbar, Muhayyuddin, L. Palomo, N. García, The kautham project: A teaching and research tool for robot motion planning, in: Proceedings of the IEEE Emerging Technology and Factory Automation (ETFA), 2014, pp. 1–8.

[22] G. Antoniou, F. van Harmelen, Web Ontology Language: OWL, 2004, pp. 67–92.