An Ontology for Failure Interpretation in Automated Planning and Execution

Mohammed Diab[1], Mihai Pomarlan[2], Daniel Beßler[2], Aliakbar Akbari[1], Jan Rosell[1], John Bateman[2] Michael Beetz[2] ⋆

[1]Institute of Industrial and Control Engineering, Universitat Politècnica de Catalunya, Barcelona
[2]Universität Bremen, Bremen, Germany

Abstract. Autonomous indoor robots are supposed to accomplish tasks, like serve a cup, which involve manipulation actions, where task and motion planning levels are coupled. In both planning levels and execution phase, several source of failures can occur. In this paper, an interpretation ontology covering several sources of failures in automated planning and also during the execution phases is introduced with the purpose of working the planning more informed and the execution prepared for recovery. The proposed failure interpretation ontological module covers: 1) geometric failures, that may appear when e.g. the robot can not reach to grasp/place an object, there is no free-collision path or there is no feasible Inverse Kinematic (IK) solution. 2) hardware related failures that may appear when e.g. the robot in a real environment requires to be re-calibrated (gripper or arm), or it is sent to a non-reachable configuration. 3) software agent related failures, that may appear when e.g. the robot has software components that fail like when an algorithm is not able to extract the proper features. The paper describes the concepts and the implementation of failure interpretation ontology in several foundations like DUL and SUMO, and presents an example showing different situations in planning demonstrating the range of information the framework can provide for autonomous robots.

1 Introduction

Challenging robotic problems, e.g. assembly tasks in cluttered environments, require planning at task and motion levels. For both levels, the use of knowledge may enhance planning and robot capabilities, giving more autonomy to the robots [1, 2]. The enhanced robot capabilities include the capture of rich semantic descriptions of the scene, knowledge about the physical behavior of objects, and reasoning about potential manipulation actions.

Different tasks may have different grades of complexity, both at symbolic and geometric levels, as well as regarding the dependence between them. Logic states and actions have to be mapped to geometric instances, and a state transition can only occur if the action is geometrically feasible. A smart combination of task and motion planning capabilities that produces fewer failures in both levels is required to make the process efficient – e.g., the symbolic planner should not ask too many impossible queries from the motion planner. For robotic systems that need an interplay between symbolic and geometric planning, interpreting failures, diagnosing their causes, and figuring out the proper solutions

⋆ This work was partially funded by Deutsche Forschungsgemeinschaft (DFG) through the Collaborative Research Center 1320, EASE, and by the Spanish Government through the project DPI2016-80077-R. M. Diab is supported by the Spanish Government through the grants FPI 2017.

for failure recovery is essential for both automated planning and execution phases. This is true for robotic systems based on heuristic search classical planning approaches [3] or for ontology-guided assembly planning [4].

Various studies have investigated the use of knowledge in the form of ontologies for the detailed description of failures in several robotics fields such as electronic industrial applications[5] or in manufacture kitting applications [6]. Several sources of failures types have been analyzed and categorized. Geometric failures related to reachability and action feasibility are described in their relationship to their causes (missing IK solution or detected collision) [7]. Failures related to motion planners not finding collision free trajectories have also been ontologically described [8, 9]. The Unified Foundational Ontology (UFO) [10] was proposed as a reference conceptual model (domain ontology) of software defects, errors and failures, which takes into account an ecosystem of software artifacts. Although the aforementioned works describe failure types, they are very task specific, do not use any sort of interpretation mechanism to diagnose the reason of failure, and don't reuse such a mechanism in a generic way. Also, there are few works considering failures of planning and run-time phase for the automated manipulation domain and representing such failure knowledge in a generalizable, shareable ontology format.

2 Problem statement and approach overview

We are mainly interested in assembly manipulation tasks for bi-manual robots, which often encounter complexity or failures in the planning and execution phases. Planning phase failures typically refer to failures of the planner itself, but we will use planning phase failures to also refer to situations where the planner reasons that some action would be infeasible, e.g. because objects block access to what the robot should reach. A correct selection of grasps and placements must be produced in such an eventuality. Depending on the type of problem, goal order must be carefully handled especially in the assembly domain; very large search spaces are possible, requiring objects to be moved more than once for achieving the goals. Execution phase failures refer to hardware failures related to the system devices– e.g. robot or camera needs to be re-calibrated–, or software failures related to the capabilities offered by specific software components, or failures in action performance such as an unexpected occluding object, or slippage.

To accurately capture, share, and reuse knowledge about failures, we propose an ontological model of failure interpretation under several foundations like SUMO [11] and DUL [12]. SUMO provides a conceptual structure that can be used and integrated with other specific ontologies developed for the robotics and automation domain, meanwhile DUL organizes concepts in a descriptive way, attempting to catch cognitive categories. It brings along modeling constraints which guide the development of domain ontologies in such a way that they may support complex reasoning tasks. The ontological formulation provides a common understanding for the robot to interpret the causes of the failures in automated planning and execution and find solutions. This paper contributes with an ontology for failures in automated planning and execution phases. The contributions are:

1. *Ontology formulation:* Introduction of an ontology to describe different types of failures in the automated manipulation planning domain.
2. *Modeling in different foundations:*
   - *For robotics domain:* Modeling the absolute abstract concepts under robotics upper-level ontologies such as CORA, which uses the SUMO ontology as an upper level.
   - *For engineering domain:* Modeling the absolute abstract concepts under very generic foundational ontologies such as DUL.

3. *Use of the failure ontology:* Description of how to use the failure ontology in a task and motion planning (TAMP) process, such as heuristic search classical approaches or the knowledge-enabled approaches, and in a static code analysis.

For the case study, we use the concept of workflow, that represent the structure of a task by subtasks linked via conditional transitions between them. In this framework, knowledge modeling for robot assembly execution includes a model of conditional workflows, software services, and robot tasks that can be automatically executed by the workflows, and a way of interfacing existing components of a robot control system.

3 Ontology driven failure interpretation

The higher level concepts in the proposed ontological module are presented. The ontological module consists of two ontologies: failure ontology and geometric one as a sub-module [1]. The combination is required when we report the geometric failure in the manipulation domain. The concepts are absolute term that cope with the modeling formalism. Moreover, some basic concepts, without being formally defined, like Agent, Plan, Task, Goal or event are used (as defined in [11, 12]). The two ontologies concepts are described below using description logic (DL).

3.1 Concepts describing failures

Task failure A situation which interprets a series of Events as the failed plan or execution of some task(s). In other words, there needs to be an Event, with an *Agent* as an effective cause; further, the *Agent* should be pursuing some *Goal*, which it does by following a *Plan* to complete a *Task*. This situation is described by a *Failure narrative*. For example, a robot has the task to serve a cup from a tray to a table. While in transit, the robot drops the cup and it shatters. The collection of events, together with the knowledge of the robot task and goal, constitutes a failure situation.

Failure Narrative A communicable description of a *Task failure* situation. It defines several roles, to be filled by an *Agent*, *Task*, and a *Goal*. It uses a *Failure symptom* to classify the *Action* that the *Agent* performed, and may provide an *Explanation* for the failure in a *Failure diagnostic*.

Failure symptom and Failure diagnostic is a simple classification label which can be applied to events or event series in order to interpret them as a failure of some sort. Failure symptoms include software error codes and signal or exception types. For example, when a robot controller raises a "hardware error signal", we would say the robot classified an event as being a hardware failure. Error status codes in query returns are another subtype of failure symptom; e.g., the status code that an IK solver would use to indicate it found no solution. Sometimes software signals or exceptions come with more data attached to them in order to identify the failure cause; for example the hardware error signal might also include which motor is thought defective. This extra information is the *Failure diagnostic*. The concepts above are defined via their relations to foundational ontologies (DUL or SUMO; see Sec. 4). Also, a taxonomy of failure symptoms is defined, as follows. One dimension of classifying failures is the "when" of happening: **Inception failures** (prevent an action from starting; e.g. *CapabilityFailure*), **Performance failures** (prevent an action from completion; e.g. *ResourceDepletionFailure*), **End state failures** (the outcome of a completed action is not conformant to the goal; e.g. *ConfigurationNotReached*). Failures

[1] ($https://github.com/MohammedDiab1/Failure\_ontology$)

are also classified by the nature of participants. Currently there are three top-level classes for this dimension: **Cognition failure**, **Communication failure**, and **Physical failure**. *CognitionFailures* classify events involving only an *Agent* and whatever internal representations it uses. *CommunicationFailures* classify an Event that involves some *Agents* exchanging information. Finally, Events with only *Physical object* or *Agent* participants can be classified as *PhysicalFailure*; e.g., the robot not being able to manipulate an object becasuse of it being too far (*ReachabilityFailure*) or because the gripper is broken (*EndEffectorFailure*).

For failures where it makes sense to identify a physical location, there is a classification along the "where" of occurrence. So far, this is only seen in the taxonomy for embodied *Physical agents*, and failures may be classified as **Body part failure** (example, *TorsoFailure*). There is a dimension of classifying failures along the "what" of the relevant interaction. So far, this is done only for *PhysicalFailures*, which can be **Mechanical failures** or **Electrical failures**.

3.2 Concepts describing geometric queries

Since one of the main focus on the proposed failure interpretation ontology is the geometric failures, an ontological module is defined to cover geometric notions used in robotics such as collision, placement feasibility, etc. The concepts in this ontology also describe geometric querying, query status, and status diagnosis. A more specific geometric ontology module contains terms for particular queries such as IK (inverse kinematics) for a specific robot. To describe geometric failure, the following terms in the ontology are included.

Geometric querying An *Event* in which some (software) spatial reasoner– e.g. a collision checker– participates, and which is classified by/executes a **GeometricReasoningTask**. It is defined as $GeometricQuerying \equiv \sqsubseteq (\exists isClassifiedBy.GeometricReasoningTask) \sqcap (\exists hasParticipant.SpatialReasoner) \sqcap (=1 hasStatus.QueryStatus)$. We also say that $GeometricReasoningTask \sqsubseteq \forall\ isExecutedBy.GeometricQuerying$.

Spatial reasoner A software component used to answer geometric queries. For example, for checking reachability, an IK solver can be used as a *SpatialReasoner*. The spatial reasoner is represented as a $SpatialReasoner \sqsubseteq ComputationalAgent$. Several types of *SpatialReasoner: CollisionChecker, IKSolver, MotionPlanner* are proposed in this ontology.

Query status The outcome of a *GeometricQuerying*. If failure, it means the query was not answered at all, and the reason is given. Such reasons include the geometric component responsible for the query not answering in time or being unavailable. If success, it means the query has been answered, and the answer can further be interpreted to ascertain what it implies for an actions feasibility. Note that "query failure" is used for situations where no answer at all is given; "geometric failure" for situations where an answer is given, but it is not satisfactory for some constraint. E.g., an IK solver returning it failed to find a solution is not a query failure (the IK solver works well) but it is a geometric failure (a *ReachabilityFailure*).

Status diagnosis Information to support the analysis of geometric query answers.

4 Modeling of failure ontology under different upper level foundations

Aiming at making the failure ontology sharable and widely used, it has been formalized under SUMO[11] and DUL[12] foundations, as shown in Table 1. The upper-level ontologies SUMO and DUL foundations are used because of their abilities to cover several

concepts related to robotics but also more general domains. Several ontology frameworks are defined under those upper-levels, such as [1], to facilitate the incorporation/importing of different ontologies with the same common vocabularies while avoiding semantic conflicts. This way of modeling becomes essential in large-scale research and development projects.

Table 1. Modeling the failure ontology under the DUL and SUMO foundations.

Concept	DUL	DL description	SUMO	DL description
Failure symptom	Event type: A Concept that classifies an Event. An event type describes how an Event should be interpreted, executed, expected, seen, etc., according to the Description that the EventType isDefinedIn (or used in)	EventType $\sqsubseteq$ Concept EventType $\sqsubseteq$ ($\forall$classifies.Event) FailureSymptom $\sqsubseteq$ EventType FailureSymptom $\sqsubseteq$ $\forall$classifies. ($\exists$hasParticipant.Agent)	Class: similar to Sets, but not assumed to be extensional, i.e. distinct classes may have the same members. Membership decided by some condition.	FailureSymptom $\sqsubseteq$ Class FailureSymptom $\sqsubseteq$ $\forall$ instance^{-1}.AgentPatientProcess
Failure Narrative Role	Role: A Concept that classifies an Object	Role $\sqsubseteq$ Concept Role $\sqsubseteq$ $\forall$classifies.Object FailureNarrativeRole $\sqsubseteq$Role	Class: similar to Sets, but not assumed to be extensional, i.e. distinct classes may have the same members. Membership decided by some condition.	FailureNarrativeRole $\sqsubseteq$ Class FailureNarrativeRole $\sqsubseteq$ $\forall$ instance^{-1}.Object
Failure Narrative	Narrative: A descriptive context of situations	Narrative$\sqsubseteq$Description FailureNarrative$\sqsubseteq$ Narrative	Proposition: An abstract entities that express a complete thought or a set of such thoughts.	FailureNarrative $\sqsubseteq$ Proposition
Task Failure	Situation: A view, consistent with ('satisfying') a Description, on a set of entities.	Situation $\sqsubseteq$ $\exists$satisfies.Description TaskFailure $\sqsubseteq$ Situation TaskFailure $\sqsubseteq$ $\exists$satisfies.FailureNarrative	Propositional attitude: An IntentionalRelation in which an agent is aware of a proposition.	TaskFailure $\sqsubseteq$ PropositionalAttitude TaskFailure $\sqsubseteq$ $\exists$satisfies.FailureNarrative
Failure diagnosis	Diagnosis: A Description of the Situation of a system, usually applied in order to control a normal behavior, or to explain a notable behavior (e.g. a functional breakdown).	Diagnosis $\sqsubseteq$ Description FailureDiagnosis $\sqsubseteq$ Diagnosis	Proposition: An abstract entities that express a complete thought or a set of such thoughts.	FailureDiagnosis $\sqsubseteq$ Proposition

5 Case study: Use of failure ontology in robotic assembly domain

5.1 In planning phase

To illustrate our proposal some simulation examples are performed using Rviz [13] for visualization and The Kautham Project for geometric planning [14], which has the ability to report the potential geometric failures. Ontologies are encoded using the Web Ontology Language (OWL) [15]. The ontologies are designed using the Protégé [2] editor.

A classical task planning approach and a knowledge-enabled approach have been used. With the former a sequence of actions is computed using a combined heuristic task and motion planning [3] (actions like transit, to move to grasping configurations, and transfer, to pick and place objects have been used). For latter a workflow is proposed that describes task and actions required in task execution. Some geometric situations are used for testing. The sequence to plan the assembly operations is:

1. call IK module to check reachability for grasping the objects;
2. call a collision checker to validate a path to grasp an object.

Some failures can happen through calling these modules. The failures get reported to the planner, where the failures are interpreted and a decision on the next action is made.

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

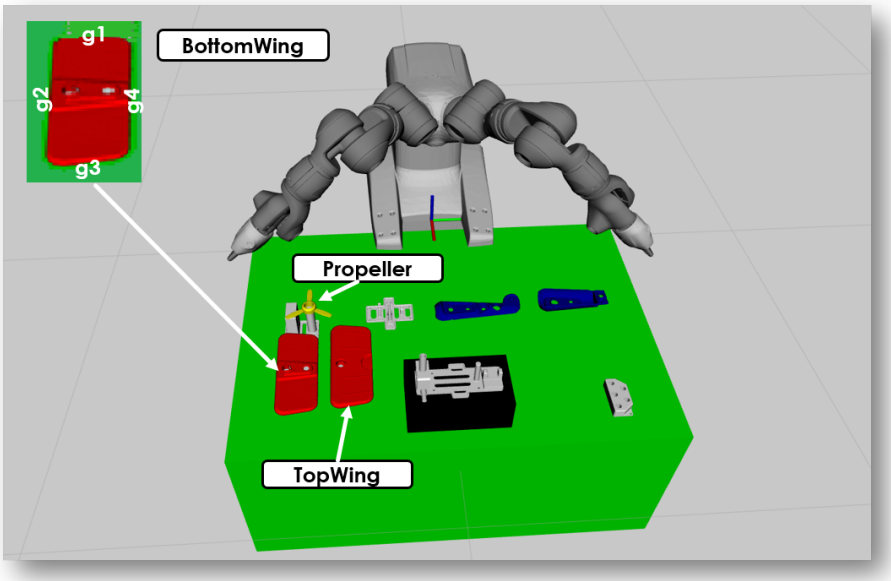

Fig. 1. Motivating example in assembly domain showing cases that need the planner to use the failure ontology to interpret query and action results. Description of reachability and collision problems. The sequence of assemble the Battat toy can be found in https://sir.upc.es/projects/ontologies/

Some situations in manipulation domain may happen often, such as the case where an object is blocking the chosen configuration to grap/place an object. This situation requires the selection of alternative feasible (or reachable) grasping poses and/or placements.

For example, as shown in Fig. 1, if the object *BottomWing* has four grasping poses from each side g1-g4, the g1 and g4 are occluded by the holders of *PropellerHolder* and *TopWingHolder* respectively, that means g1 and g4 are not feasible and the robot is not able to reach object *BottomWing* through them. Meanwhile, the robot may not be able to grasp the object *BottomWing* through g2 and g3 because of the infeasibility of IK configurations. By querying over the proposed ontology, the robot will be able to analyze the cause and report to the planner (infer a prober solution will be included in our upcoming work). Fig. 2, shows how the assembly process of the *battat* toy is done.

The failure symptom produced when planning to use g2 and g3 for grasping is a *ReachabilityFailure*, which is a *CapabilityFailure*, whereas a failure produced when planning to grasp using g1 or g4 is an *OcclusionFailure*, which is an *AffordanceFailure*. Both of these are *InceptionFailures* which prevent the planned task from even being undertaken. *CapabilityFailure* can be addressed by generating new capabilities, e.g. selecting new grasping poses that are reachable. Affordance failures, meanwhile, can be addressed by manipulating the environment to better expose its affordances, so by generating intermediary goals of moving the occluders out of the way, the robot can eventually grab the *BottomWing*.

To interpret the causes of those situations presented in Fig. 1, the geometric ontology is integrated with the failure ontology as described in Fig. 3. This ontology describes the failure symptoms, as well as the *FailureNarratives* which make use of these symptoms to classify failures. The narratives may include other information to enhance the diagnosis process, such as the initial goal and participating objects in the agent's task, and a diagnos-

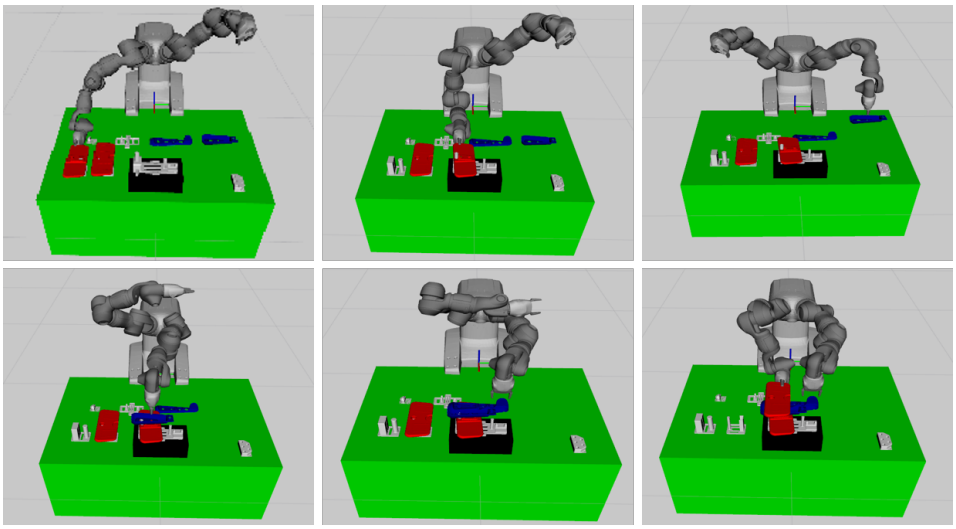

Fig. 2. Motivating example in assembly domain showing how to assemble the battat toy

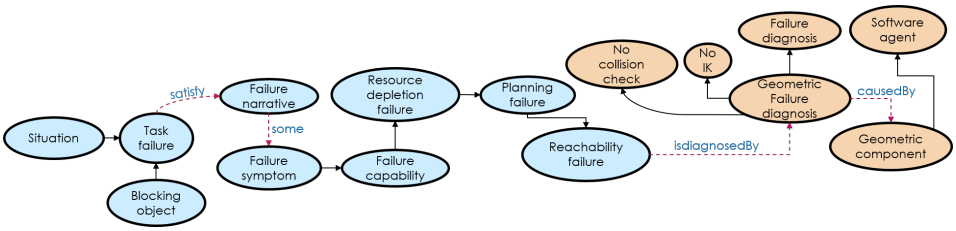

Fig. 3. An interpretation of blocking object using the proposed failure ontology. The concepts in blue belong to failure ontology, meanwhile the ones in yellow are from the geometric ontology.

tic to indicate which component failed. Failure symptoms are ontologically characterized also in terms of what failure diagnostics they are compatible with; for example, a *ReachabilityFailure* can only be used by a failure narrative where the explanation role is played by a failure diagnostic that names some IK component as the failure cause. These modules (i.e, IK and collision check) are low-level modules that the symbolic level considers to be spatial reasoners.

5.2 Static program analysis in the workflow execution

When writing programs for robots, developing the failure handling branches often takes considerable time and is a complex, error prone process in itself. An ontology of failures, as we have presented here, enables reasoning for the analysis of such programs even before they are run, and may improve the development process by identifying what parts of a program need strengthening against failure eventualities.

Such reasoning also needs a representation of programs or workflows, and upper ontology axioms. Our technique for static analysis works if the tasks in a workflow are characterized properly, i.e., each task belongs to a task concept that has sufficient axioms to

define it. Definitions of such task concepts, as well as object concepts restricting what may participate in a task and filling which roles, are not part of the failure ontology module but should be part of the knowledge model of the system as a whole. The reasoning process can only be done within some models of the world. That is, there is a classification of potential failures (this is the failure ontology), a classification of possible tasks, their roles, and possible fillers for them. Within this model, we can ask whether certain combinations of tasks and failures make sense, and this is what our static code analysis answers. From previous work, we take a workflow to be a transition system where the nodes correspond to tasks, linked by transitions that are conditional on, among other things, task outcomes including failure signals, if any. Tasks can be "atomic", but they can also be described by workflows. When executing a workflow and encountering a non-atomic task, the execution process will begin following a path through the workflow describing the non-atomic task, in a pattern similar to invoking a piece of a program called a subroutine via a shorthand name for it.

The kinds of reasoning questions we will focus on here are the following:

1. Are there possible failures not included among the outcomes of the workflow describing how to perform a particular task?,
2. Are the failure outcomes of a workflow actually possible outcomes of the task described by the workflow?,
3. Are there enough branches in a workflow to account for the possible failure outcomes of a subtask of the workflow?

For all of these queries, we assume there exists some ontological characterization of a task, and a workflow that describes how to perform this task. An ontological characterization of a task means indicating what kind of task it is, and what restrictions there are on the participants in an event classifiable by this task. The ontological characterization of failure symptoms is from our ontology. In effect, the above queries involve answering one question: for a particular Task, what are the possible failures that can happen? This can be analyzed using the Distributed Ontology, Modeling and Specification Language (DOL) [16] pattern below. Here, *ExampleTask* and *ExampleFailure* are some named subconcepts of *Task* and *FailureSymptom* respectively, while *Ot*, *Of* are an ontology of tasks and our ontology of failures, respectively.

```
ontology PossibleFailure[Class: ExampleTask SubClassOf: Task]
                        [Class: ExampleFailure SubClassOf: FailureSymptom]
                        [Class: ExampleEvent SubClassOf: Event]
 given Of, Ot =
 ObjectProperty: isClassifiedBy InverseOf: classifies
 Class: ExampleEvent
  SubClassOf: (isClassifiedBy some ExampleTask) and
              (isClassifiedBy some ExampleFailure)
end
```

A DL reasoner would be presented with the ontology resulting from applying the pattern above, which combines an ontology of tasks and other relevant concepts such as robot parts and objects, with our ontology of failures and the axioms above. The reasoner would be required to find a model for *ExampleEvent*. If no such model exists, then the failure symptom is impossible for the task. This query would be run for every pair of named task/failure symptom concepts in an ontology.

6 Conclusion

This paper proposes the formalization and implementation of the standardized failure ontology to extend the capabilities of autonomous robots related to manipulation tasks that require task and motion planning, including actions description, along with execution. This combination require the integration of geometric ontology that also proposed here. In the modeling level, the absolute concepts of both ontologies are modeled under DUL and SUMO foundations to facilitate the usability for roboticists community. A case study is introduced to illustrate the use of failure ontology in automated planning and workflow execution phases by proposing the common situations that could be encountered by such planner. Moreover, some static code analysis is proposed to analyze the possible failures while executing the tasks.

References

1. Diab, M., Akbari, A., Ud Din, M., Rosell, J.: PMK - a knowledge processing framework for autonomous robotics perception and manipulation. Sensors **19**(5) (2019)
2. Tenorth, M., Beetz, M.: Representations for robot knowledge in the knowrob framework. Artificial Intelligence **247** (2017) 151 – 169 Special Issue on AI and Robotics.
3. Akbari, A., Lagriffoul, F., Rosell, J.: Combined heuristic task and motion planning for bimanual robots. Autonomous robots (2018) 1–16
4. Beßler, D., Pomarlan, M., Akbari, A., Muhayyuddin, Diab, M., Rosell, J., Bateman, J., Beetz, M.: Assembly planning in cluttered environments through heterogeneous reasoning. In Trollmann, F., Turhan, A.Y., eds.: KI 2018: Advances in Artificial Intelligence, Cham, Springer International Publishing (2018) 201–214
5. Zhou, X., Ren, Y.: Failure ontology of board-level electronic product for reliability design. In: The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety. (June 2011) 1086–1091
6. Kootbally, Z., Schlenoff, C., Antonishek, B., Proctor, F., Kramer, T., Harrison, W., Downs, A., Gupta, S.: Enabling robot agility in manufacturing kitting applications. Integrated Computer-Aided Engineering (Preprint) (2018) 1–20
7. Srivastava, S., Riano, L., Russell, S., Abbeel, P.: Using classical planners for tasks with continuous operators in robotics. In: Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence. (2013)
8. Caldiran, O., Haspalamutgil, K., Ok, A., Palaz, C., Erdem, E., Patoglu, V.: Bridging the gap between high-level reasoning and low-level control. In: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer (2009) 342–354
9. Caldiran, O., Haspalamutgil, K., Ok, A., Palaz, C., Erdem, E., Patoglu, V.: From discrete task plans to continuous trajectories. Proc. of BTAMP (2009)
10. Duarte, B.B., Falbo, R.A., Guizzardi, G., Guizzardi, R.S., Souza, V.E.: Towards an ontology of software defects, errors and failures. In: International Conference on Conceptual Modeling, Springer (2018) 349–362
11. Niles, I., Pease, A.: Towards a standard upper ontology. In: Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001, ACM (2001) 2–9
12. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: Wonderweb deliverable d18 ontology library. Technical report, IST Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web (08 2003)
13. Kam, H.R., Lee, S.H., Park, T., Kim, C.H.: Rviz: A toolkit for real domain data visualization. Telecommun. Syst. **60**(2) (October 2015) 337–345
14. Rosell, J., Pérez, A., Aliakbar, A., Muhayyuddin, Palomo, L., García, N.: The kautham project: A teaching and research tool for robot motion planning. In: Proceedings of the IEEE Emerging Technology and Factory Automation (ETFA). (Sept 2014) 1–8
15. Antoniou, G., van Harmelen, F. In: Web Ontology Language: OWL. Springer Berlin Heidelberg, Berlin, Heidelberg (2004) 67–92

16. Mossakowski, T., Codescu, M., Neuhaus, F., Kutz, O.: The distributed ontology, modeling and specification language–dol. In: The Road to Universal Logic. Springer (2015) 489–520