Reasoning and state monitoring for the robust execution of robotic manipulation tasks*

Oriol Ruiz
Institute of Industrial and Control Eng.
Universitat Politècnica de Catalunya
Barcelona, Spain
oriol.ruiz.celada@estudiantat.upc.edu

Jan Rosell
Inst. of Industrial and Control Eng.
Universitat Politècnica de Catalunya
Barcelona, Spain
ORCID: 0000-0003-4854-2370

Mohammed Diab
Dept. of Electrical and Electronic Eng.
Imperial College London
London, UK
ORCID: 0000-0002-5743-5190

***Abstract*—The execution of robotic manipulation tasks needs to be robust in front of failures or changes in the environment, and for this purpose, Behavior Trees (BT) are a good alternative to Finite State Machines, because the ability of BTs to be edited during run time and the fact that one can design reactive systems with BTs, makes the BT executor a robust execution manager. However, the good monitoring of the system state is required in order to react to errors at either geometric or symbolic level requiring, respectively, replanning at motion or at task level. This paper make a proposal in this line and, moreover, makes task planning adaptive to the actual situations encountered by knowledge-based reasoning procedures to automatically generate the Planning Domain Definition Language (PDDL) files that define the task.**

***Index Terms*—Robotic manipulation, task planning, task monitoring, reasoning, ontology.**

I. Introduction and overview

Some of the required capabilities for mobile manipulators to make them able to autonomously move and work in semi-structured human environments are:

- Smart perception capabilities that are able to, not only perceive, but to understand the current state of the environment (including the robot itself).
- Adaptive planning capabilities that are able to: a) plan at task levels according to the current state of the environment and to the goal to be achieved, updating the initial state and choosing the actions the robot can do to solve the problem; b) plan at motion level according to the current poses of the objects, modifying grasping configurations if necessary, object placement poses or the robot base location, and choosing the most appropriate motion planner.
- Robust execution capabilities that are able to integrate both the smart perception capabilities and the adaptive planning capabilities to avoid failures or, if failures occur, be able to reason on the failed state in order to plan the best recovery strategy.

This paper partially contributes to the last two capabilities.

This work was partially supported by the Spanish Government through the project PID2020-114819GB-I00

A. Planning capabilities

On the one hand, task planning copes with the determination of the sequence of actions to be done to perform a given task. In this sense, many problems can be tackled with classical planning approaches, that assume known initial values of variables, deterministic actions, and a set of goals defined over the variables [1]. These problems are usually modeled using the Planning Domain Definition Language (PDDL, [2]), which is an action-centered language that use pre and post-conditions to describe, respectively, the applicability of actions and their effects. Planning tasks specified in PDDL are separated into two files, a domain file for predicates and actions, and a problem file for objects, initial state and goal specification. On the other hand, knowledge plays a significant role in planning, mainly to enhance the capabilities of the robots and make them able to comply with the actual situations encountered. Knowledge needs to be structured such that it is usable for reasoning tasks and, in this line, ontologies arise as hierarchical structures expressing the universe of discourse based on relations, such as *is-a* and *has-a*, between concepts and instances of classes, being these concepts, instances, and relations expressed in formal languages. Many studies have investigated the use of knowledge in planning using ontologies, like in the manipulation domain [3], in the navigation domain [4], or in the perception domain for manipulation [5]. A review of the use of ontologies to give robot autonomy is presented in [6].

This paper proposes the use of ontologies with knowledge about the objects in the environment, their features and possible manipulation constraints, and about the robots and their capabilities, to reason on the type of actions and the robot required to solve a given manipulation task. As a result, the PDDL files shall be automatically generated, which can then be used to solve the problem with any classical task planner. This way breaks the closed-world assumption that is intrinsically present in many task planning approaches.

B. Executing capabilities

In order to execute the planned manipulation tasks, Behavior Trees (BT, [7]) can be used. BTs are modular and re-usable tree structures conceived as sets of action-conditions generally called behaviors. The BT nodes are broadly classified into two

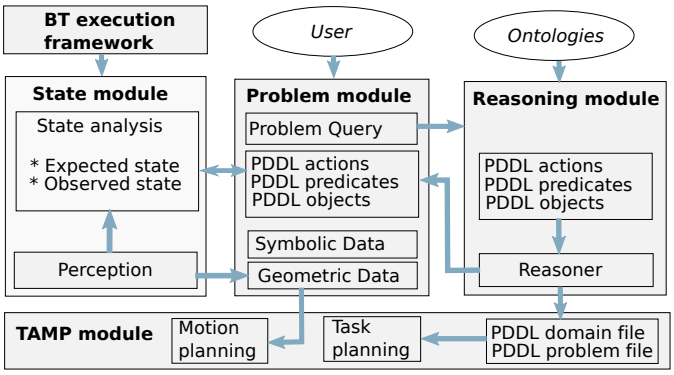

Fig. 1. Overall System Schema

classes: *Control Nodes*, that help in regulating the execution flow, and the *Execution Nodes*, which are leave nodes used to get a feedback from environment (*Condition Nodes*) or to perform a robot action (*Action Nodes*). A BT-based execution framework for task and motion planning (TAMP) was proposed in [8]. In this work in progress we propose a state monitoring module to be used by this BT execution manager to react to non-expected situations, i.e. *Condition Nodes* will query the module to test whether the conditions to execute an action hold, e.g. whether the object to be picked is located at the expected region and with the expected pose. Whenever the condition is not met, replanning at either motion or task level is required, thus giving the necessary reactive capability to cope with failures or changing environments.

C. Overview

In order to make the task planning capabilities of a robotic system adaptive and reactive to the actual situations, the PDDL domain and problem files should be automatically generated, and a state monitoring should be able to detect the situations that need replanning actions. Upon a user request of a task to be solved and once the perception module provides the current state of the environment, the system should reason on the actions to be used to solve the task, among all the possible actions that the available robots can perform.

Figure 1 shows the main modules of the proposal:

- A *Problem module*: A module used by the user to define the environment (geometric and symbolic data) and the manipulation problem to be solved. It contains the set of actions, predicates and objects provided by the Reasoning module.
- A *Reasoning module*: Assuming known all the possible actions that the robots can perform, all the predicates that may be required to define the state of the system, and the robots, regions and objects, this module reasons, using ontologies, on the subsets needed to solve the task queried by the user.
- A *State module*: Equipped by a perception submodule able to detect the objects poses, the state module is responsible to determine the symbolic state (*Observed state*). It also maintains which is the *Expected State* according to the last action executed. This module is directly connected to the BT execution manager.

The *Reasoning module* is detailed in section II and the *State module* in section III. Finally, Sec. IV sketches the conclusions and on-going and future work.

II. Reasoning Module

A standardized ontological-based reasoning framework called Perception and Manipulation Knowledge (PMK) was introduced in [5] as a tool to help task and motion planning systems (TAMP) in terms of reasoning, by providing:

a) Reasoning for perception related to sensors and algorithms, e.g. to determine which is the sensor to be used in a given situation.
b) Reasoning about the objects features, e.g. to determine if an object is pickable or not.
c) Reasoning for situation analysis to spatially evaluate the objects relations between each other, e.g. to determine if an object is behind another.
d) Reasoning for planning to reason about the preconditions of actions, action constraints and geometric reasoning related to the robot and to the environment, e.g. to determine if a grasping pose is reachable or to select a feasible placement region.

PMK has been extended here by including in the knowledge the actions the available robots can perform, and by broadening the object features related to those actions. Also reasoning predicates have been provided to help in the selection of the actions required to solve a given task and to automatically set the PDDL domain and problem files.

A PDDL parser package called *Universal-PDDL-parser* [9] has been used to write the PDDL domain file including only those actions that are relevant for the task, the decision of this relevance being made by the reasoner according to the knowledge available in the extended PMK ontology.

A list of all the possible PDDL actions is assumed to be initially available to the reasoner, as well as the PDDL predicates. The selection of the actions is handled by asking task specific questions to the knowledge layer which is formed as ontologies within Protégé interface. The appropriate questions are created as Prolog predicates and are used to check robot capabilities and to check the available objects within the task environment, e.g.: a) `find_robot(Region, Robot)` returns the available robots within the environment and the regions they are located at; b) `find_robot_capability(Robot, Capability)` returns the capabilities of a given robot; c) `find_robot_reach(Robot, Region)` checks if a given robot can reach a given region using its capabilities.

During the reasoning process, the questions (predicates) asked to the ontologies are grouped into two sections, one with the global predicates that are asked always such as the predicates related to robot capabilities, independent of the specific task, and the other with task specific predicates that are only asked depending on the answers of the global predicates, such as the features of the specific environmental entities.

III. State module

The State Module is the one which keeps track of the State of the system, and will need to detect if a change has occurred in order to trigger a replan.

A. Expected State and Observed State

The way the State Module works is by having two versions of the State:

- **Expected State**: it is the state that the system *should* be in if the tasks that have been carried up to a point have been successful. At the start of the execution it is the state that is read and interpreted directly from the PDDL files. When an action is performed, the Expected State needs to be updated with the new stage of the planned execution.
- **Observed State**: it is the state the state that the system *is* in, observed using the Perception Module (which is a submodule within the State Module). During each loop of the State Module, the environment is observed, meaning that the appropriate sensors are called and the Observed values in the state are updated according to the output.

These two versions of the system State are required as there needs to be a way to compare the states according to the original plan and the actual states, i.e., a replan is necessary if there is a mismatch between the Expected and the Observed states, as the original plan built with the Expected State may no longer be valid.

B. Observing the State

The State Module also handles the conversion between data from the sensors (geometric poses, joint states, state of the gripper, etc.) and symbolic data for the task planning process. Some criteria to translate from the output of the perception services to the binary conditions of the predicates is required. A distinction is made between Observable and Not Observable predicates.

- **Not Observable predicates**: they contain those predicates that will always match their Expected value due to their nature as intrinsic properties of the state defined at the start of the execution. For instance, the predicate `REACHABLE ROBOT REGION` indicates that the robot arm can reach the location designated as `REGION`. This property will not change during the course of the execution, it is immutable.
  They also contain to properties that cannot be observed using the current Perception Module, even if they are not intrinsic properties and are prone to change during the execution. Without the ability to sense them, there is no alternative other than trusting that they are set to the value they are Expected to be. For instance, if in a scenario in a kitchen there is a predicate that controls the state of the oven, `OVEN ON`, but the oven itself has no sensor, the system has to trust that it has been turned ON when the appropriate action has been performed. Not Observable predicates are not considered when comparing the Observable and Expected states, as there is no risk of mismatch.
- **Observable predicates**: they contain those predicates that can be observed using the Perception module and have the potential to cause a replan. For instance, the predicate `IN OBJA ZONEL` is observable, and requires that the perception module outputs the pose of Object A, detected by e.g. a fidducial marker, and then assigns a location to that object if the object pose lies within the limits of a location region.

As an example, the implemented State Module has been tested with the capabilities to observe and interpret the logic of the following predicates:

- **Location predicates**: those related to where the object is observed.
  - `IN OBJ ZONE`: Set to true when the object `OBJ` is in the region `ZONE`.
  - `CLEAR OBJ`: Set to true when there is nothing on top of the object `OBJ`.
  - `ONSTACK OBJ1 OBJ2` Set to true when two objects are stacked, with `OBJ1` on top of `OBJ2`.
  - `VISITED ZONE`: Set to true when an object has occupied region `ZONE` at some point in the execution.
- **Gripper predicates**: those related to the gripper according to whether it is holding any object.
  - `HOLDING ROBOT OBJ`: Set to true when the robot `ROBOT` is holding the object `OBJ`.
  - `HANDEMPTY ROBOT`: Set to true when the robot `ROBOT` has the gripper free to grab something. There is overlap with `HOLDING ROBOT OBJ` but both are needed to facilitate the description of the actions preconditions.

C. State definition

The State Module loads the Expected State from the initial files and then constantly observes the environment. States are defined with the following data structure:

- Obstacles: list of all the obstacles in the problem, each with the following properties:
  - Name
  - Expected Pose
  - Expected Location
  - Observed Pose
  - Observed Location
- Robots: list of all the robots in the problem, each with the following properties:
  - Name
  - Joint States
- Locations: list of all the locations in the problem. Each location is defined with the following properties:
  - Name
  - Location x-y-z limits
- Expected Predicates: list of all the predicates that are True for the expected state. At the start they are read from

the PDDL file and then, each time an action in the task plan is executed, it is updated by adding and deleting the appropriate predicates following, respectively, the positive and negative effects of the action taken.
- Observed Predicates: list of all the predicates that are True based on the data from the Perception module.

D. Associated BT Nodes

The BT Nodes related to the State Module are:

- `CompareStates`: Condition Node which will perform the comparison between the Expected State and the Observed State by comparing the predicates that are observable and thus have the possibility to be different. If there is a mismatch it returns the Observed State in order to rewrite the appropiate files to begin a new execution (the old Observed State will become the new Expected State for the next execution).
- `LoadState`: Condition node that is called at the start of any BT execution to load the Expected State from the indicated files.
- `UpdateExpected`: Condition Node which is called after an action is executed and that computes the state the system should be in when that action is performed, following the rules established by the PDDL domain file.
- `UpdateExpectedPose`: Condition Node which is used to modify the Expected Pose of an object (this is needed in cases a Place action is performed).
- `CheckObjPose`: Condition node which is called to check if the expected pose and the observed pose of an object match (within a tolerance), used to trigger motion replans that will not change the task plan.

E. Perception

In order to provide the necessary knowledge to the reasoning process for both the domain and problem PDDL generation, perception module should transfer the observed environment where the task will take place. Various methodologies can be followed for this purpose such as the use of fiducial markers placed on the objects to be detected. Another solution is to use Deep Neural Networks as in the Deep Object Pose Estimation project [10]. The latter solution requires a training session for the objects that are desired to be detected. For the objects that are placed in non-line-of-sight regions, by using RFID sensors [11], the detection of these objects can also be accomplished. In addition to the available objects, human operators in the environment can be detected using computer vision and deep learning techniques [12]. All of the selected perception submodules, independent of the methodology they follow, will provide information of the detected aspects within the task environment to the main perception module through a ROS interface.

IV. Conclusions and Future Work

This paper has presented a new proposal to make the task planning capabilities of a robotic system adaptive to the current situation of the environment (which object are there and in which locations), to the available robots and their features (which actions can be performed), and to the task to be solved as queried by a human operator. The contribution of automatically generating the PDDL domain and problem files breaks the closed-world assumption that is intrinsically present in many task planning approaches. Also, the definition of a State Module to encapsulate the evaluation of the state at each step in the sequence of actions to solve a manipulation task, allows the Behavioral Tree implementation of a task and motion planning framework to be focused on the robot side. The on-going work is centered in validating the proposal with several case studies using the YuMi robot in table-top manipulation tasks, and comprising examples regarding the need of: a) replanning at motion level; b) replanning at task level; c) reasoning to replan at task level with an extended set of actions and resources.

References

[1] M. Ghallab, D. Nau, and P. Traverso, *Automated planning: theory & practice*. Elsevier, 2004.

[2] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL—The Planning Domain Definition Language," 1998.

[3] M. Beetz, D. Beßler, A.Haidu, M. Pomarlan, Bozcuoglu, and G. Bartels, "Know Rob 2.0 — a 2nd generation knowledge processing framework for cognition-enabled robotic agents," pp. 512–519, 2018.

[4] J. Ruiz-Sarmiento, C. Galindo, and J. Gonzalez-Jimenez, "Building multiversal semantic maps for mobile robot operation," *Knowledge-Based Systems*, vol. 119, p. 257–272, 2017.

[5] M. Diab, A. Akbari, M. U. Din, and J. Rosell, "PMK - A knowledge processing framework for autonomous robotics perception and manipulation," *Knowledge-Based Systems*, vol. 19, p. 1166, 2019.

[6] A. Olivares-Alarcos, D. Beßler, A. Khamis, P. Goncalves, M. Habib, J. Bermejo-Alonso, M. Barreto, M. Diab, J. Rosell, J. Quintas, J. Olszewska, H. Nakawala, E. Pignaton, A. Gyrard, S. Borgo, G. Alenyà, M. Beetz, and H. Li, "A review and comparison of ontology-based approaches to robot autonomy," *The Knowledge Engineering Review*, vol. 34, 2019.

[7] M. Colledanchise and P. Ögren, "Behavior trees in robotics and AI," Jul 2018. [Online]. Available: http://dx.doi.org/10.1201/9780429489105

[8] P. Verma, M. Diab, and J. Rosell, "Automatic generation of behavior trees for the execution of robotic manipulation tasks," in *IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 2021.

[9] Universal PDDL parser multiagent. [Online]. Available: https://github.com/aig-upf/universal-pddl-parser-multiagent

[10] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," 2018.

[11] M. Diab, M. Pomarlan, D. Beßler, A. Akbari, J. Rosell, J. Bateman, and M. Beetz, "Skillman — a skill-based robotic manipulation framework based on perception and reasoning," *Robotics and Autonomous Systems*, vol. 134, p. 103653, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889020304930

[12] A. Brunetti, D. Buongiorno, G. F. Trotta, and V. Bevilacqua, "Computer vision and deep learning techniques for pedestrian detection and tracking: A survey," *Neurocomputing*, vol. 300, p. 17–33, Jul 2018.