Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Automating adaptive execution behaviors for robot manipulation

O. RUIZ-CELADA[1], P. VERMA [1], M. DIAB [2] AND J. ROSELL.[1]
[1]Institute of Industrial and Control Engineering, Universitat Politècnica de Catalunya, Barcelona, Spain
[2]Personal Robotics Lab. at the Imperial College London, South Kensington Campus, London, United Kingdom

Corresponding author: P. Verma (parikshit.verma@upc.edu).

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

ABSTRACT Robotic manipulation in semi-structured and changing environments requires systems with: a) perception and reasoning capabilities able to capture and understand the state of the environment; b) planning and replanning capabilities at both symbolic and geometric levels; c) automatic and robust execution capabilities. To cope with these issues, this paper presents a framework with the following features. First, it uses perception and ontology-based reasoning procedures to obtain the Planning Description Domain Language files that describe the manipulation problem at task level. This is used in the planning stage as well as during task execution in order to adapt to new situations, if required. Second, the proposed framework is able to plan at both task and motion levels, intertwining them by incorporating geometric reasoning modules to determine some of the symbolic predicates needed to describe the states. Finally, the framework automatically generates the behavior trees required to execute the task. The proposal takes advantage of the ability of behavior trees to be edited during run time, allowing adaptation of the action plan or of the trajectories according to changes in the state of the environment. The approach allows for robot manipulation tasks to be automatically planned and robustly executed, contributing to achieve fully functional service robots.

INDEX TERMS Adaptation, Behaviour Tree, Knowledge-based Reasoning, Manipulation Planning

I. INTRODUCTION

MANY efforts in industrial and service robotics pursue making mobile manipulators able to act autonomously in semi-structured human environments. The final aim is to actually make them able to be robot co-workers at the factory floor or robot helpers at home. This poses different challenges at the perception, planning and action levels. At the perception level, there is the need to capture the state of the environment, which requires not only the detection of the objects but also the understanding of the situation. For this, deep-learning-based approaches can be used to perceive different sets of objects and to detect their poses from 2D and 3D images, and ontological-based reasoning procedures can be used to interpret the situations. At the planning level the simultaneously combination of task and motion levels is required to actually obtain geometrically feasible sequences of actions to perform a manipulation task. At the execution level, in order to successfully perform manipulation tasks, robust strategies for grasping and motion execution are required, as well as adaptive strategies to comply to sensed changes, or reactive behaviors able to recover from unforeseen situations by reasoning on failures.

Many advances have been already done in all these lines, although great efforts are still needed to make robots fully autonomous. In this direction, the paper contributes with the proposal of a reasoning-based robotic manipulation framework with robust and adaptive planning and execution capabilities.

A. PREVIOUS WORKS

Task and motion planning (TAMP) is a discipline devoted to find, for a given task, a complete sequence of actions along with feasible paths that allows to fulfill it. For robotic manipulation, this may be challenging since the actions required to perform the task may be subject to strong geometric constraints from the environment (lack of space for placing objects, occlusions) and the robot (reachability of objects, kinematic constraints of the manipulators). One of the ways of combining planning levels is to rely on classic task planning approaches, like the HTAMP [1] approach that is based on the heuristic-based Fast Forward task planner (FF, [2]), which has been modified so that its heuristic function takes geometric constraints into account through various geometric reasoning procedures. The HTAMP approach will be used in this work through ROS services (Robotic Operation System, [3]).

Classical task planning approaches, that assume known initial values of variables, deterministic actions and a set of goals defined over the variables [4], are usually modeled using the Planning Domain Definition Language (PDDL, [5]), which is an action-centered language that uses 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. The closed-world assumption is intrinsically present for classical task planning approaches modeled with PDDL. To handle PDDL this work will use the Universal PDDL Parser [6], which is a package for parsing planning problems in PDDL format.

When executing the planning tasks, difficulties may arise if the states encountered differ from the ones expected. To deal with this, a recent work [7] interleaves the symbolic and geometric search processes by calling a motion planner at every step of the symbolic search and by tentatively assigning geometric parameters to the current symbolic state before advancing to the next action. The resulting plan remains valid even if the objects are moving and can be executed by reactive controllers that adapt to the environmental changes. Nevertheless, this adaptation is done at the action execution level (i.e., controller adaptation), not at a plan generation level (i.e., adding actions to the plan). The adaptation at a plan generation level requires of a reasoning mechanism to infer the actions to be added to recover from the current unexpected situation. In this line, this work will introduce a methodology towards automating adaptive behaviors, demonstrating its ability to adapt to changing environments both in simulation and real experiments.

Also, knowledge plays a significant role in enhancing the capabilities of the robots and make them able to comply with the actual situations encountered, like e.g. the hierarchical representation composed of geometric and symbolic scene graphs used in [8] as a structured, object-centric abstraction of manipulation scenes that can be quickly processed by graph neural networks to plan the task. However, in order to structure knowledge for reasoning purposes, 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 [9], like KnowRob [10] or PMK [11]. This work will extend the PMK ontology to reason on the required actions to solve a task, thus breaking the closed-world assumption by flexibly configuring the task planning PDDL domain and problem files.

A TAMP framework was proposed in [12] that used FF and The Kautham Project (TKP, [13]), a motion planning tool based on the Open Motion Planning Library suite of sampling-based motion planners (OMPL, [14]) that offers motion planning and geometric reasoning ROS services. The framework basically defines an interface layer as an XML file where to include the geometric description of each symbolic action, and implements a TAMP manager as a ROS client that calls the task and motion planning services. This TAMP framework has been updated with the use of HTAMP instead of FF, allowing to always obtain sequences of actions that are feasible (although for simple manipulation tasks where the task and motion planning levels are not very tightly coupled, the use of FF may be enough).

Looking for the real execution of the sequence of motions required to fulfill a manipulation task, a preliminary work [15] introduced the idea of extending this TAMP framework by making the TAMP manager to automatically write an output XML file that represents the Behavior Tree (BT, [16]) that may allow to execute the task with a real robot using a BT executor. Behavior Trees are a good alternative to Finite State Machines (FSMs, [17]), or to more ad hoc methods like the automatic generation and parameterization of skill primitives for maintenance automation tasks [18]. Since BTs can be represented in an XML format, to execute the TAMP problems in a real robot, the framework simply needs to generate the behavior tree XML file once the TAMP problem is solved and prior to the real execution. Moreover, the ability of BTs to be edited during run time and the fact that one can design reactive systems with BTs, makes BT executor a robust execution manager.

Other studies have also investigated the use of BTs with an adaptive perspective at different levels, like a semi-autonomous BT framework for the automation of sorting-based industrial applications [19], a reactive mobile manipulation system [20] where the adaptability comes from a proposed robust and reactive motion controller allowing the robots to achieve a desired end-effector pose taking into account several constraints, or as an efficient general reactive planning tool to adapt to dynamic environments [21]. Also, with the aim of further increasing the adaptability, a recent approach [22] proposed the combination of planning and learning techniques to generate BTs. In a similar direction, the combination of planning with AI techniques is proposed here, using reasoning instead of learning.

B. PROBLEM STATEMENT AND SOLUTION OVERVIEW

Let us consider a working scenario with: a) one or more robots able to perform some predefined (possibly big) set of manipulation actions like Move, Pick, Place, Stack or Unstack; b) a set of objects to be manipulated with known geometric models and grasping transforms; c) a set of locations (defined by some geometry and symbolic name) where robots or objects can be placed; d) a perception system able to identify and locate the pose of the objects and their locations. Then, given a final state of the environment determined by the user, a robotic system must be able to plan and execute a manipulation task to change the state of the environment from the initial to the final desired one, despite possible changes in the state of the environment encountered while executing the task.

To cope with this problem the paper proposes a planning and execution framework with (see Fig. 1): a) a perception module able to detect the poses of the objects using fiducial markers and to determine their locations, generating a list of objects in the environment with geometric and symbolic information; b) geometric and symbolic ontology-based reasoning modules able to reason on the locations of the objects and on the actions required to solve a manipulation task according to the current and desired goal states and the available robots, filling up the required information and, hence, generating the PDDL domain and problem files as well as the motion planning problem file; c) a TAMP generator which combines the symbolic and geometric information and configures the planning problem in a TAMP configuration file; d) a task planner server able to find a sequence of actions to solve a task, assuming a PDDL task description; e) a motion planning server able to find collision-free paths and to solve geometric queries, using geometric models of the objects and the robots; f) a TAMP manager able to coordinate the planning levels using the TAMP configuration file to link symbolic and geometric information of the actions, and able to write a behavior tree XML file; g) a behavior tree executor able to execute a task in a flexible way, and to adapt to changes at different levels, replanning at motion or task level as required, or even reasoning on the required actions to recover from an unexpected state. The recovery process entails calling the perception module to observe changes in the environment, calling the reasoning module to make changes in the problem files, calling the TAMP generator to reproduce the TAMP configuration file accordingly and, finally, querying the TAMP manager to produce a favorable BT XML file which the BT executor adapts to.

The contributions of the paper are:

- A smart manipulation system able to perceive and understand the current state of the environment, and to reason on the actions required to solve the task, modeling the task with the appropriate PDDL domain and problem files.
- A robust execution mechanism that can automatically generate and update BTs, giving rise to a behavior reactive to changes at geometric and symbolic levels.

After this introduction, Sec. II presents the reasoning module and Sec. III the behavior tree structure and its automatic generation and update. Then, Sec. IV shows some demonstration examples and, finally, Sec. V presents the conclusions and future works.

II. REASONING FOR TASK PLANNING

This section describes a reasoning module conceived to help increasing the robot autonomy by automatically generating the PDDL domain and problem files needed to solve a task. Assuming the availability of a perception module able to detect the objects and agents in the environment, the following two challenges need to be addressed: 1) the interpretation of the perceived environment through a reasoning framework to understand the current scene; 2) the generation of the task specific PDDL files, based on the current state of the environment and assuming known all the possible actions that can be executed in the domain and provided by a given global domain PDDL file. This paper suggests the overall schema shown in Fig. 2 to cope with the above mentioned challenges.

A. ONTOLOGY-BASED REASONING FOR ROBOT MANIPULATION

Robotic manipulation involves the planning at task level (determining which is the sequence of actions to be done to perform a given task) and at motion level (finding the sequence of collision-free motions that allow to safely move the robot from one configuration to another). In this scope, a standardized ontological-based reasoning framework, called Perception and Manipulation Knowledge (PMK), was introduced in [11] as a tool to help task and motion planning systems (TAMP) in terms of reasoning. PMK provides:

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 is extended here by including the knowledge about the actions the available robots can perform, and by broadening the object features related to those actions. Also reasoning predicates from PMK are extended to help in the selection of the actions required to solve a given task and to automatically set the PDDL domain and problem files. These predicates allow robot-centered and state-centered reasoning:

1) Robot-centered reasoning

It includes the following predicates: a) `find_robot (Region, Robot)` to return the available robots within the environment and the regions they are located at; b) `find_robot_capability (Robot, Capability)` to return the actions a given robot can do; c) `find_robot_reach (Robot, Region)` to check if a given robot can reach a given region using its capabilities.

2) State-centered reasoning

State-centered reasoning is required to reason on the initial and goal state of the world. Even though the perception module provides the poses of the detected ob-

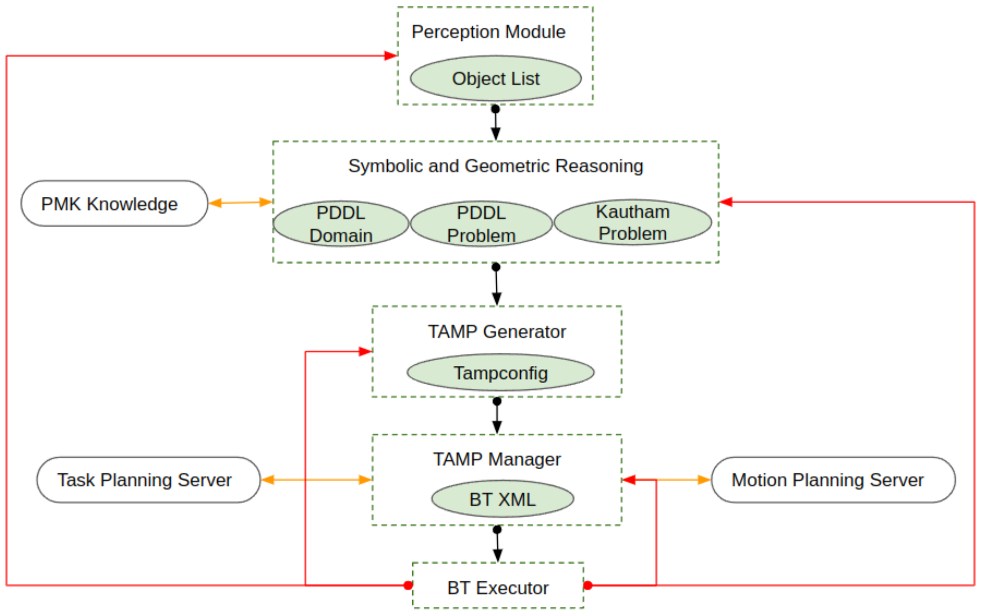

FIGURE 1: Schema of the proposed TAMP framework. The Reasoning is described in Section II, and the BT Generation and Execution in Section III.

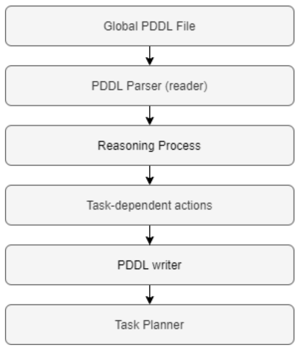

FIGURE 2: Flowchart of the overall symbolic reasoning operation.

jects and agents, it does not provide the symbolic regions where they are located, nor the spatial relation information between the detected objects (i.e. in, on, left, right). For this purpose, spatial evaluation predicates from PMK, which convert raw perceived environment information into spatial locations and relations, are used to reason on the state of the environment based on the perceived objects, e.g. `retrieve_symbolic_region_init (objPose, symbolicRgn)` predicate returns the symbolic region of an object given its pose, and `on (obj1, obj2)` predicate evaluates to true if object `obj1` is on top of `obj2`. These calculations are based on comparing perceived coordinates of the objects with specific values depending on the spatial relation definition. For instance, to infer the *on* relation between two objects, the x and y coordinates should be similar for the two objects, meanwhile the z coordinates

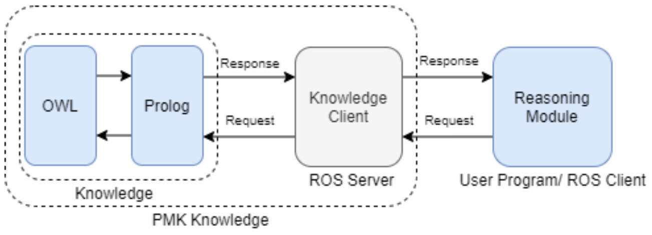

FIGURE 3: Knowledge management schema.

should be apart within a certain threshold.

In addition to finding object locations and the spatial relations between the objects, the initial and goal states can be retrieved with the following predicates inherited from PMK: 1) `retrieve_symb_init (Task)`; 2) `retrieve_symb_goal (Task)`, which gather all the state predicates that evaluate to true in the initial and goal state, respectively.

The overall reasoning process through the above-mentioned predicates queried over knowledge is presented in Section II-C.

In order to reason over the knowledge, an intermediate layer between the knowledge and the user program is required to organize the sequence of the predicates to be queried with the relevant information coming from the perception module. For this purpose, in this paper, the overall knowledge management schema in Figure 3 is implemented through a ROS interface.

B. PDDL PARSER

In order to deal with the challenge to generate task specific PDDL files, the PDDL parser and writer tool called Universal-PDDL-parser [6] is used. With the help of this package, a global PDDL domain file with all the possible actions that can be done with the robots available is first parsed into its elements (e.g. the actions and the predicates defining their preconditions and effects) and the contents is stored using the classes within the package (with the help of this hierarchical class structure, the links between the elements in the global file are also preserved).

After reading and storing the global domain file, the task specific PDDL files for the given task is automatically generated by using the same parser tool to write into a new PDDL file including only the actions that are relevant for the task. The decision of this relevance is made by the reasoner. In order to rewrite the task specific domain file, the feasible actions list determined by the reasoning framework is forwarded into *print(std::ostream& os, std::vector<std::string> actionList)* function in the parser package where *os* input is the new task specific PDDL file to be generated and the *actionList* input is the feasible action set coming from the reasoner.

C. AUTOMATIC PDDL GENERATION

PDDL files are automatically generated with the help of the aforementioned perception module, reasoning predicates, and PDDL parser. The reasoning process requires the combination of the robot and state-centered reasoning predicates discussed above to answer the query: *Which are the actions to be included in the PDDL files to solve this problem?*. Particularly, for a given task and situation, the following information is obtained from the predicates introduced in Sections II-A1 and II-A2:

- Spatial relations between the objects.
- Initial and goal states.
- Available robots and their capabilities.

Finally, based on this information: a) The robot assigned to the task is selected; b) The PDDL problem file is written with the actual initial state; c) The PDDL domain file is written by the PDDL parser with the required actions to solve the given task. The required reasoning predicates can be extended according to the complexity of the task.

As an example, imagine that in a table-top manipulation task to be done by a fixed robot, one of the objects is in another counter out of the robot reach. In this case, the reasoning module should determine that the Move action is required, as well as a robot with navigation capabilities able to perform that action.

III. BEHAVIOR TREE-BASED EXECUTION

Behavior Trees (BTs) is a useful mechanism to implement an execution manager for a robotic system, which provides modularity and re-usability features. The building blocks of BTs are known as BT-nodes, which may be either Execution Nodes or Control Nodes. The Execution Nodes are leaf nodes, i.e. they do not have child nodes, and are used to query the robotic system hardware, either to get a feedback from the sensors or to perform a robot action. They can be Action Nodes that return `SUCCESS`, `FAILURE` or `RUNNING` and that can be preempted, or Condition Nodes that return just `SUCCESS` or `FAILURE` and that cannot be preempted. Control Nodes, on the other hand, are used to control the execution flow by regulating a periodic signal, called tick, amongst its multiple child nodes, one child node at a time, in a given sequence. They can be Sequence Nodes that return `SUCCESS` only if all of the child nodes return `SUCCESS`, or Fallback Nodes that return `SUCCESS` as soon as one of the child nodes returns `SUCCESS`. A highly reactive behavior can be achieved by the variants of these nodes, called Reactive Sequence Nodes and Reactive Fallback Nodes, which while a given current node is `RUNNING`, the previous nodes in the sequence or fallback are continuously ticked so as to monitor a change in their state that shall abort the execution of the running node.

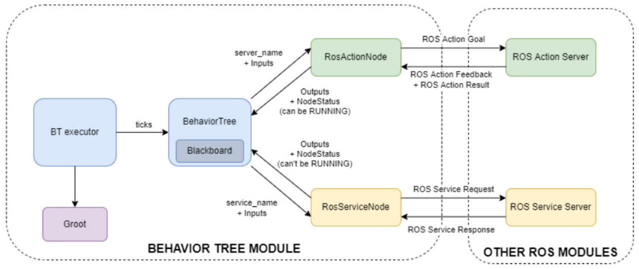

FIGURE 4: Schema of the Behavior Tree ROS implementation.

Mathematically, BTs can be represented as directed acyclic graphs and hence easily be described in an XML file. Moreover, each node in a behavior tree can have input and output data ports which provides flexibility in the exchange of data between nodes and between different behavior trees. To implement BTs for task and motion planning problems, the *behaviorTree.ROS* [23] library can be used, which provide classes to initialize BT nodes as ROS nodes, allowing them to act as clients to ROS Action Servers and Service Servers (Fig. 4). As the BT action nodes can be preempted, they are implemented as clients to ROS Action Servers, while BT condition nodes are implemented as clients to ROS Service Servers. The exchange of information from a BT client to a ROS server is done via input-output ports of BTs, using a variable storage system called Blackboard which can be accessed using a key/value system.

This section proposes a BTs structure to execute the sequence of actions of a manipulation task generated by the TAMP framework described above, and how to automatically generate the XML files that describe them. The proposal seeks to achieve robustness in the task execution and, with this aim, the main BT is defined as shown in

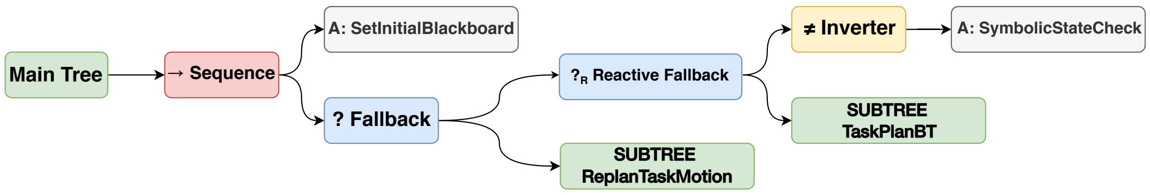

FIGURE 5: The main Behavior Tree.

Fig. 5, where a Reactive Fallback node is used which, while the `TaskPlanBT` responsible of the task execution is `RUNNING` (meaning it is performing the different actions in the plan), a node called `SymbolicStateCheck` keeps monitoring those variables of the state which are always observable (e.g., an object being held by the gripper), so as to interrupt the execution of the `TaskPlanBT` and launch a `ReplanTaskMotion` process if an erroneous state is detected in order to replan the task from the current state. Those state variables which are not always observable (e.g., the pose of an object may be occluded by the robot while it is moving) will be monitored just before the execution of each robot action, as shown in the next section.

A. TASK-LEVEL BEHAVIOR

The `TaskPlanBT` is a simple BT sequence, as that shown in Fig. 6(left), which will execute the actions of the task plan, as provided by the task planner. The actions (Pick, Move, Place, etc.) are BT subtrees named after the actions name plus a task index (e.g., MOVE1, PICK2, MOVE3, PLACE4,...), and all have the same general sequence structure shown in Fig. 6(right), that includes:

- First, a symbolic check: the preconditions of the actions are checked and if there is a mismatch between the expected and observed state variables, a full replan is triggered (details on the state module monitoring can be found in [24]).
- Second, a geometric check: the relevant poses of the objects involved in the action are checked, so as to verify whether they are equal to those that have been used to plan the motions (within a given tolerance), e.g. a Pick action will look that the object to be picked at position x=0.3 m, y=0.3 m and z=0 m is actually at that pose or, otherwise, will replan a motion to comply to the actual pose. This is different to the symbolic check because as long as the predicates remain the same, this only has an effect on the motion to be executed, not the task plan. The objective is to fix small deviations from the expected object poses, like for instance in the case a Place action leaves the object slightly moved of where it was supposed to be left, the next Pick will need a correction.
- Then, after the preconditions are checked, the action itself is executed following a BT subtree according to the type of action (see next section).
- Finally, once the task has been performed, the expected state is updated in order to be used in the next action symbolic check.

B. ACTION-LEVEL BEHAVIORS

Each action of a task sequence is modeled with a BT composed of different action nodes to control the robot motions. For instance, a Pick task is carried out with a sequence composed of: a) a `MoveTraj` action node responsible to move the robot along a trajectory towards the grasp position next to the object; b) a `Gripper` action node to close the gripper; and c) another `MoveTraj` action node to go back to the safe home position.

C. REPLANNING BEHAVIORS

To do a replan at task and motion levels, the following files need to be updated: a) the PDDL problem file required by the task planner needs to be updated with the initial state according to the new observed predicates; b) the motion planning problem file required by the motion planner and the TAMP configuration file required by the TAMP manager need to be updated with the new poses of the objects. This is done following the `ReplanTaskMotionBT` shown in Fig. 7. After its execution, a new BT results and, once the BT executor is aware of it, it switches the BT to the new one to be executed. On the other hand, if there are no symbolic mismatches but only geometric differences, a behavior called `ReplanMotionBT` is triggered which changes the geometric files and calls the motion planner with the updated information.

D. AUTOMATIC GENERATION OF BEHAVIOR TREES

The BTs are automatically generated by the TAMP manager. First, the main BT is generated and, once the task planner service is called, the manager writes the XML file for the `TaskPlanBT` with the particular task plan, hence completing the first instance of Task-Level Behavior.

Then, the TAMP manager manages each of the actions in the plan using the information of the TAMP configuration file and calling the motion planning service when necessary. For each action, it writes the XML file corresponding to the Action-Level Behavior tree of the action. This includes the coding of the trajectory generated by the motion plan as required by the ROS Action Service of the robot. Once all the actions have been managed, the first instance of all the Action-Level Behaviors is complete.

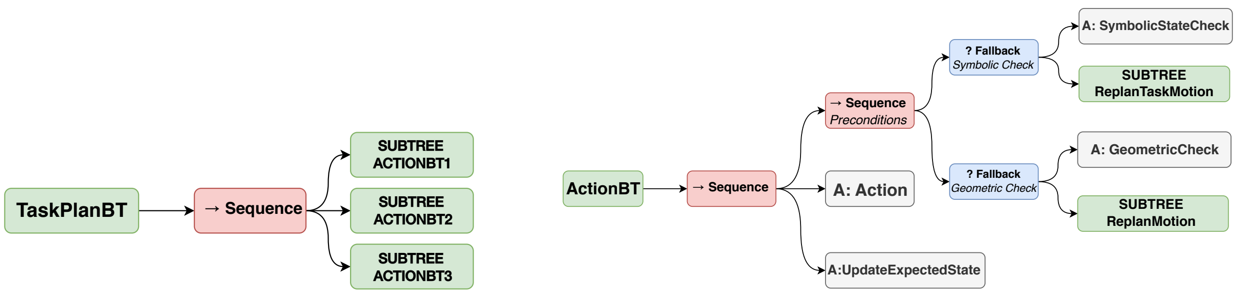

FIGURE 6: (left) a `TaskPlanBT` with 3 generic actions; (right) the `ActionBT` general sequence structure for the actions, where a symbolic check and a geometric check are done before the actual execution of the action.

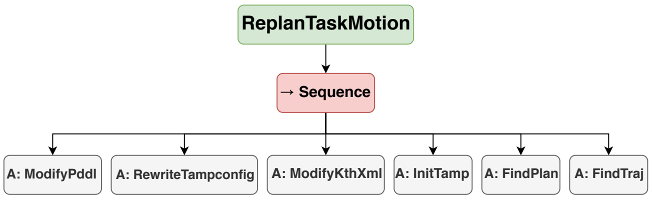

FIGURE 7: `ReplanTaskMotionBT`: the sequence of actions to be performed to replan the whole task from the current state, i.e. to plan the sequence of actions and their motions.

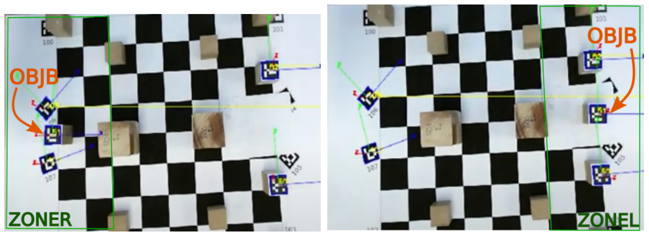

FIGURE 9: Scenario 1 example 1: Goal (left) and init (right) configurations.

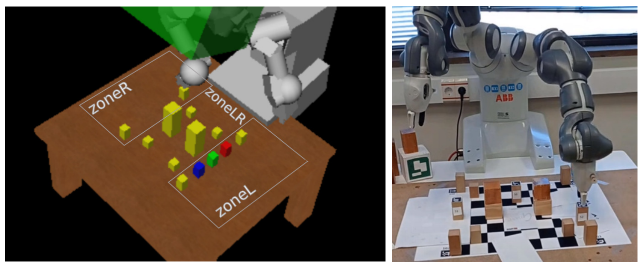

FIGURE 8: Table top manipulation task: the three colored objects must be moved between regions.

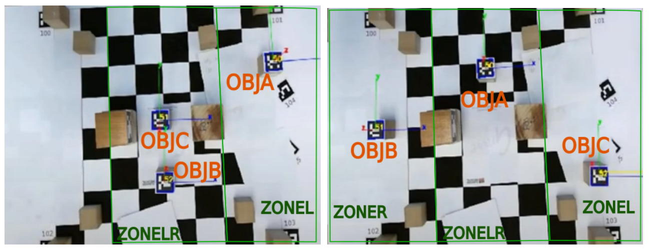

FIGURE 10: Scenario 1 example 2: Goal (left) and init (right) configurations.

The generated BT XML files are passed to the BT executor which is responsible for initializing the tree, ticking the nodes of the BT and monitoring the state of the Task-level and Action-level behaviors. If a change in state is observed, the tree is re-initialized or partially rewritten according to the change being symbolic or only geometric, hence executing the task and motion planning problem in a robust manner.

IV. VALIDATION

In this section three examples in two different scenarios have been designed to illustrate the ability of the proposed framework to automatically generate the code to robustly execute a manipulation task, recovering from different situations as required. The examples will illustrate the capability of the proposed framework to adapt to changes at geometric, symbolic or ontological level. No performance measures are reported since the single purpose is the validation of the adaptation capabilities provided.

The first scenario, used by the first two examples, is a table-top manipulation problem performed by the dual-arm Yumi robot. The table is divided into three regions, namely ZONEL, ZONER and ZONELR which are reachable by the left arm, the right arm or both of them, respectively. The task consists in moving objects (square prisms) from one zone to another (see Fig. 8).

The perception in the first scenario is done with a Kinect Camera and the use of ArUco fiducial markers attached to the objects which allow detection of their poses.

Geometric-level adaptation: The first example is used to illustrate adaptation capability at geometric level. The task consists in moving OBJB from ZONEL to ZONER (see Fig 9). The solution can be visualized here: https://youtu.be/jDIdwYBZYVg. In a second execution of the task, the object is purposely moved by an operator (see Fig. 11 left), that slightly changes its pose while being in ZONELR. Since the

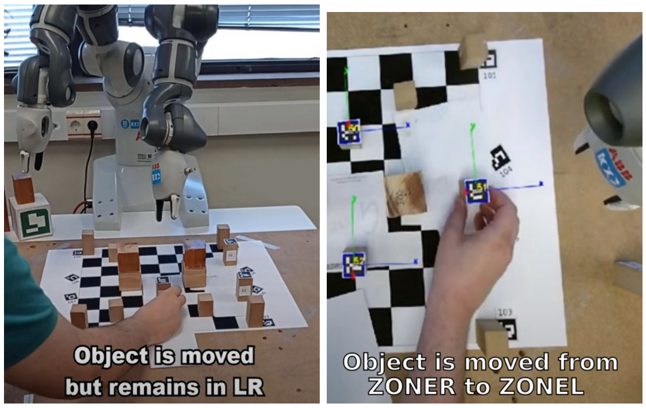

FIGURE 11: Human operator changing the object poses to force the need of replanning.

object continues to be in its expected location, no task replan is required but just the second Pick action needs to adjust its motion by calling the `ReplanMotionBT`, as shown in the second video https://youtu.be/zeRMDqWBBXc.

Task-level adaptation: The second example is used to illustrate adaptation capability at task level. In this task, objects OBJA, OBJB and OBJC are moved, respectively, from regions ZONELR, ZONER and ZONEL to regions ZONEL, ZONELR and ZONELR (see Fig 10). The original plan execution is shown in this video: https://youtu.be/zCy2hL7fELk. In a second execution of the task, the object OBJB is purposely moved by an operator (see Fig. 11 right), from ZONER where it is expected to be toward ZONEL where it will be observed. A task replan is required as shown in the second video: https://youtu.be/CI1mGexu8o8.

In the second scenario, used in the third example, there are two robots, the Yumi and the TIAGo, in a simulated environment. It is a manipulation task to be performed by the Yumi robot to stack objects on its working table (Fig. 12-top). The solution sequence of actions is: Unstack-Stack-Pick-Stack.

Ontology-level adaptation: This example is used to illustrate adaptation capability at reasoning level. Upon the start of the execution the initial pose of the blue cylinder is changed to a location out of the reach of the Yumi robot (Fig. 12-bottom). This requires that the reasoning module to conclude that the TIAGo robot is required because it is a mobile manipulator able to perform the Move action. The final solution sequence has seven actions, the first three performed by the TIAGo and the last four by Yumi: Pick-Move-Place-Unstack-Stack-Pick-Stack. The video of the simulation is available at https://youtu.be/MI7NOs1C_S0.

V. DISCUSSION AND CONCLUSIONS

Robotic manipulation requires task and motion planning capabilities in order to find a feasible sequence of actions, which is a challenging problem since the interaction between both planning levels may be relevant. In this paper, an heuristic-based task and motion planner is used that is able to find such a feasible sequence of actions, but the focus of the proposed approach is on how to actually execute it in a robust manner, i.e. the work has focused on the development of tools to provide robots with the capabilities to make them autonomous enough to automatically execute manipulation tasks, monitoring possible changes in the environment and replanning as necessary to adapt to them.

The proposed approach has first dealt with the development of reasoning capabilities to reason on the environment (objects and available robots) and on the task goal to be achieved in order to find out the required actions to solve the task, and to assist the task planner with the automatic generation of the PDDL files. Moreover, the proposal has presented a behavior-trees (BT) execution framework that: a) automatically generates the BTs required to execute the task and b) automatically updates them in order to react to possible changes at the geometric level (a new motion is needed for a given action), at symbolic level (a new sequence of actions is required to solve the task), or at the ontological level (a new PDDL domain file is required with a different set of actions in order to find the plan to solve the task). This combination of planning and reasoning within a BT execution framework increases the adaptability of the system, i.e. the reasoning mechanism provides the missing information that may be required while planning or executing the task, which increases the adaptation performance. This is not the case in similar approaches such as [22], that obtains a reactive and flexible system by combining planning and learning within a BT execution framework, but that requires having prior complete knowledge to plan the task correctly and that may fail when the necessary knowledge is missing. Moreover, in our framework, we demonstrate the proposal in a real scenario.

The adaptive task and motion planning capabilities of the proposed framework is a step towards making robots more aware, smarter and reactive. The paper has validated the viability of the approach. We are currently working in three main directions: a) extending the actions that the robots can do to quantitatively evaluate the performance of the proposal; b) incorporating the human operator as an agent so as to allow robots to play the co-worker role; c) improving the monitoring mechanisms so as to be able to cope with failures caused by perception and reasoning errors.

REFERENCES

[1] A. Akbari, F. Lagriffoul, and J. Rosell, "Combined heuristic task and motion planning for bi-manual robots," *Autonomous Robots*, no. 43, pp. 1575–1590, 2019.

[2] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, pp. 253–302, 2001.

[3] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, 2009, p. 5.

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

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

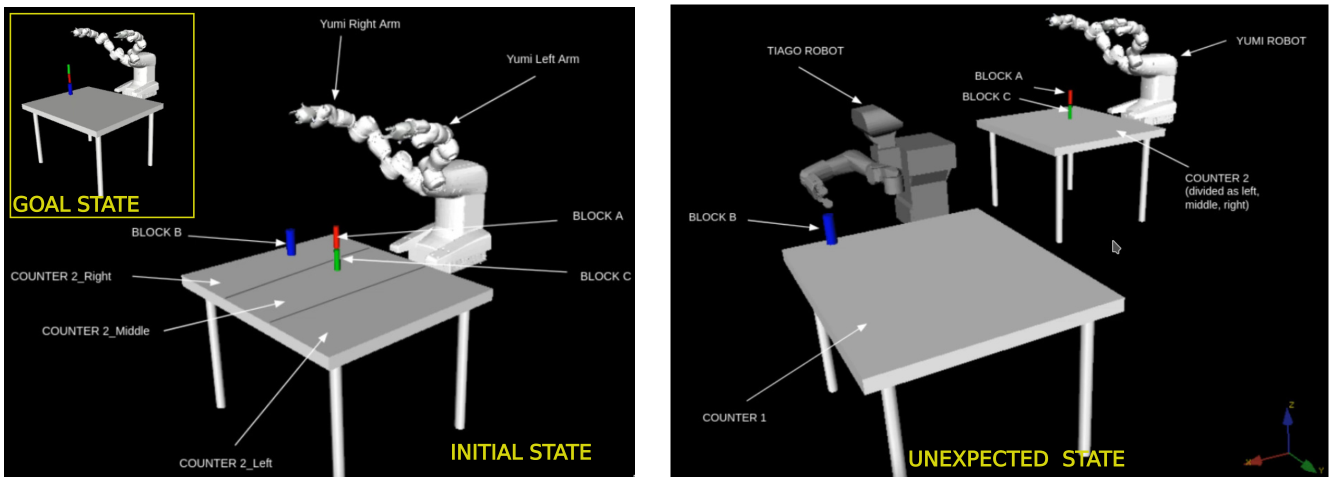

FIGURE 12: Yumi manipulation task.

[6] D. Furelos-Blanco, G. Francès, and A. Jonsson. Universal PDDL parser multiagent. [Online]. Available: https://github.com/aig-upf/universal-pddl-parser-multiagent

[7] T. Migimatsu and J. Bohg, "Object-centric task and motion planning in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 844–851, 2020.

[8] Y. Zhu, J. Tremblay, S. Birchfield, and Y. Zhu, "Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 6541–6548.

[9] 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.

[10] M. Beetz, D. Beßler, A.Haidu, M. Pomarlan, Bozcuoglu, and G. Bartels, "KnowRob 2.0 — a 2nd generation knowledge processing framework for cognition-enabled robotic agents," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 512–519.

[11] 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.

[12] S. Saoji and J. Rosell, "Flexibly configuring task and motion planning problems for mobile manipulators*," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1285–1288.

[13] J. Rosell, A. Pérez, A. Aliakbar, Muhayyuddin, L. Palomo, and N. García, "The Kautham Project: A teaching and research tool for robot motion planning," in *IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 2014.

[14] I. Sucan, M. Moll, L. E. Kavraki *et al.*, "The open motion planning library," *Robotics & Automation Magazine, IEEE*, vol. 19, no. 4, pp. 72–82, 2012.

[15] P. Verma, M. Diab, and J. Rosell, "Automatic generation of behavior trees for the execution of robotic manipulation tasks," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2021.

[16] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of Behavior Trees in robotics and AI," *Robotics and Autonomous Systems*, vol. 154, p. 104096, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889022000513

[17] G. O'Regan, *Automata Theory*, 2016, p. 117–126.

[18] C. Friedrich, R. Gulde, A. Lechler, and A. Verl, "Maintenance automation: Methods for robotics manipulation planning and execution," *IEEE Transactions on Automation Science and Engineering*, pp. 1–11, 2022.

[19] A. Rastegarpanah, H. C. Gonzalez, and R. Stolkin, "Semi-autonomous behaviour tree-based framework for sorting electric vehicle batteries components," *Robotics*, vol. 10, no. 2, 2021. [Online]. Available: https://www.mdpi.com/2218-6581/10/2/82

[20] J. Haviland, N. Sünderhauf, and P. Corke, "A holistic approach to reactive mobile manipulation," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3122–3129, 2022.

[21] M. Colledanchise, D. Almeida, and P. Ögren, "Towards blended reactive planning and acting using behavior trees," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8839–8845.

[22] J. Styrud, M. Iovino, M. Norrlöf, M. Björkman, and C. Smith, "Combining planning and learning of behavior trees for robotic assembly," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2022, pp. 11 511–11 517.

[23] D. Faconti. BehaviorTree.ROS. [Online]. Available: https://github.com/BehaviorTree/BehaviorTree.ROS

[24] O. Ruiz, M. Diab, and J. Rosell, "Reasoning and state monitoring for the robust execution of robotic manipulation tasks," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022.

ORIOL RUIZ-CELADA recently graduated in the double Master in Industrial Engineering and Automatic Control and Robotics at the Barcelona School of Industrial Engineering - Universitat Politècnica de Catalunya (UPC). His final thesis explored the inclusion of Behavior Trees in an autonomous robot manipulator system, with the focus on reacting to changes in the environment and performing Task and Motion replanning. He is pursuing the Ph.D. in Automatic Control, Robotics, and Computer Vision at UPC, working on the development of an adaptive manipulation execution framework for multiple robots.

PARIKSHIT VERMA is currently pursuing his Industrial PhD in mobile robotics. He completed his graduation in 2016 from Thapar University, India in mechatronics engineering. He then completed his masters in Automatic Control and Robotics at the Barcelona School of Industrial Engineering - Universitat Politècnica de Catalunya (UPC) in 2021. He is passionate about applied robotics and integration of robots in real industrial scenarios. He is pursuing the Ph.D. in Automatic Control, Robotics, and Computer Vision at UPC, working on control strategies for the traffic management of AGV-based transportation systems.

MOHAMMED DIAB is research associate in the Personal Robotics Lab. (PRL), Imperial College London (ICL). He has been awarded his Ph.D. degree in Automatic Control, Robotics, and Computer Vision at the Universitat Politècnica de Catalunya (UPC). During his Ph.D., he developed knowledge-driven frameworks for autonomous robots that have manipulation skills. He is a voting member of the "Autonomous Robots" sub-group of the IEEE WG ORA that works on the standardization of ontologies for robotics and automation. He is currently working for the UK Research Innovation (UKRI) Trust Node project funded by Innovate UK, in which he develops human behaviour inference methods for Trustworthy Human-Robot Interaction.

JAN ROSELL received the BS degree in Telecomunication Engineering and the PhD degree in Advanced Automation and Robotics from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 1989 and 1998, respectively. He joined the Institute of Industrial and Control Engineering (IOC) in 1992 where he has developed research activities in robotics. He has been involved in teaching activities in Automatic Control and Robotics as Assistant Professor since 1996 and as Associate Professor since 2001. His current technical areas include task and motion planning, mobile manipulation, and robot coworkers.