Interleaving hierarchical task planning and motion constraint testing for dual-arm manipulation*

Alejandro Suárez-Hernández, Guillem Alenyà and Carme Torras

***Abstract*— In recent years the topic of combining motion and symbolic planning to perform complex tasks in the field of robotics has received a lot of attention. The underlying idea is to have access at once to the reasoning capabilities of a task planner and to the ability of the motion planner to verify that the plan is feasible from a physical and geometrical point of view. The present work describes a framework to perform manipulation tasks that require the use of two robotic manipulators. To do so we employ a *Hierarchical Task Network* (HTN) planner interleaved with geometric constraint verification. In this framework we also consider observation actions and handle noisy perceptions from a probabilistic perspective. These ideas are put into practice by means of an experimental set-up in which two Barrett WAM robots have to cooperatively solve a geometric puzzle. Our findings provide further evidence that considering explicitly physical constraints during task planning, rather than deferring their validation to the moment of execution, is advantageous in terms of execution time and breadth of situations that can be handled.**

I. Introduction

Motion planners can solve effectively the problem of navigating the world attending to physical limitations. However, if the reasoning that the robotic agent is capable of conducting is purely geometrical, it will be closer to an automaton than to an intelligent entity. This could be enough or even desirable in restricted environments like assembly lines in industry. In other environments, one could come up with a very intricate algorithm that results in a seemingly intelligent behaviour. However, such a solution scales badly and lacks adaptability.

On the other hand, the deduction capabilities provided by symbolic planners introduce the ability to reason about the objects that conform the world. General-purpose symbolic planners operate with a description of the problem which consists of the set of actions and the state fluents. Since the nature of these descriptions is declarative, symbolic planners are quite flexible and adaptable to different problems. This also opens some interesting possibilities like automatic domain learning [1].

Our framework combines a symbolic *Hierarchical Task Network* (HTN) planner, motion planning and geometrical constraint handling to increase the effectiveness of raw symbolic and motion planners for bi-manual tasks, common in humanoid robotics. This work describes a simple but effective idea: decompose the task into sub-goals, and test the geometric constrains and dual arm collisions only at the sub-goal level. Our main contributions are:

- We propose to codify different dual arm alternative ways of implementing an action as distinct symbolic methods in the HTN. This allows our framework to choose preferentially to move both arms simultaneously, and only when it is not possible, rely on actions moving only one arm at a time. This leads to more efficient plans in terms of time, that can be easily extended to safety or other criteria.
- We propose to explicitly encode into the state the uncertainty inherent in humanoid robot tasks, and also to provide the robot with three classes of actions: actions to *solve* the task, actions to *reduce* uncertainty in perceptions, and actions to *reshape* the environment to reduce the uncertainty in the manipulation.

In this paper we will present our ideas by means of a particular application to provide better intuition about the suggested methods. Imagine we have two robots that we identify by their roles: the *picker* and the *catcher*. Their objective is to solve a geometric matching puzzle in which they have to insert several pieces with different shapes in a sphere with cavities. The pieces are initially distributed over a table in front of the robots. The *picker*'s purpose is to grasp the pieces from the table and insert them into the sphere, while the *catcher* has to facilitate the task showing the relevant cavity without hindering the *picker*'s movements. Percepts from the world come from a *Kinect* camera located at the ceiling. A piece cannot be grasped if it is too close to another one to avoid collisions of the gripper. Fig. 1 illustrates this set-up.

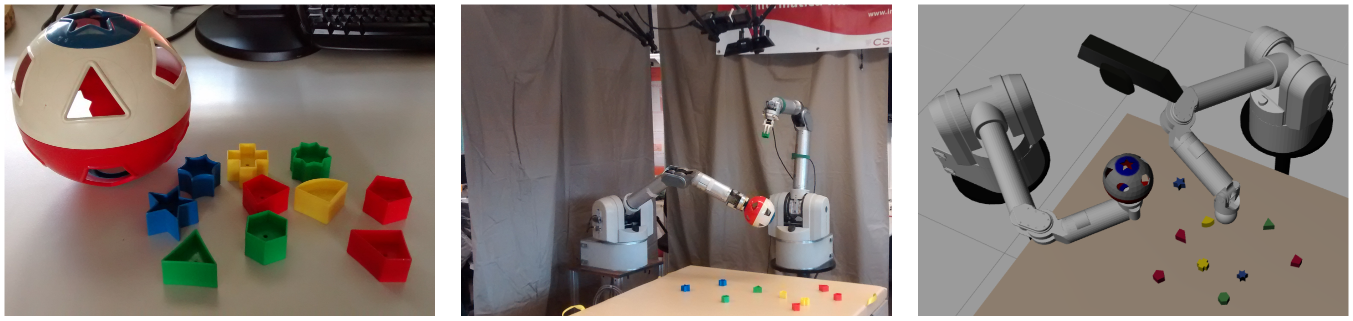
In Section II we review some related works and their influence on our own efforts. Section III is a brief reminder of the HTN theory for the unfamiliar reader. Section IV describes how we handle the uncertainty about the percepts. Section V deals with the domain representation. Section VI shows our experimental results. Finally, Section VII wraps up with some conclusions and ideas for future work.

II. Related work

The *Hierarchical Planning in the Now* (HPN) framework proposed by Kaelbling and Lozano-Pérez [2] is very relevant in the context of this paper. It consists in decomposing highly abstract goals into increasingly specific tasks via nested planning procedures until arriving to the primitive operators. The hierarchy is expanded at runtime to avoid

*This work is partially funded by the European project IMAGINE (Robots Understanding their Actions by Imagining their Effects), H2020-ICT-2016-1-731761, by the Spanish Ministry of Science and Innovation HuMoUR TIN2017-90086-R and the Spanish State Research Agency through the Maria de Maeztu Seal of Excellence to IRI (MDM-2016-0656).

*Authors are with Institut de Robòtica i Informàtica Industrial (CSIC-UPC). Llorens i Artigas 4-6, 08028 Barcelona, Spain. E-mails: {asuarez,galenya,torras}@iri.upc.edu



(a) (b) (c)

Fig. 1. Depiction of the different elements in our proposed task, both in real life and in simulation. (a) Detail of the toy sphere and the pieces. Notice that there is just one occurrence per shape and that the pieces have different colours. These facts are exploited by the perception routines. (b) Real WAM robots in their roles of picker and catcher, together with other elements of the experiment. Notice the *Kinect* camera at the ceiling. (c) Recreation of the proposed scenario in a simulation environment. We have employed the *Gazebo* robot simulator to extract some quantitative performance measures.

wasting computational resources if unexpected events arise, rendering the plan invalid (i.e. the system plans with short horizons). We believe that this is reminiscent of the HTN paradigm we have adopted. The same authors have extended HPN to operate in belief space and handle domains with uncertainty [3]. This work emphasizes the importance of planning observation actions to *reduce* the uncertainty of the current beliefs which is also within our scope. They focus on navigation and detection in pick-and-place domains, while we focus on identification and assembly-like applications.

Highly related to our own work we find Ferrer-Mestres *et al.* [4]'s approach to combine classical planning with motion planning. The authors propose a strategy to deal with symbolic/geometric domains with a functional variant of the STRIPS language and putting great emphasis on the preprocessing stage. In this preprocessing stage, they compute several motion plans that can be reused in different problems of the same domains. They focus on pick-and-place problems consisting of objects distributed over one or more tables. These objects have to be rearranged in a particular way by a PR2 robot using one arm. However, we put more emphasis on the interaction between two robotic arms, and we prefer to avoid precomputing an exhaustive library of motion plans.

It would seem that many authors adhere to the idea of task decomposition for planning in robotics, like Nau *et al.* [5], [6] and Marthi *et al.* [7], who introduced the so-called High-Level Actions (HLA) and Hierarchical Angelic Planning (HAP). In our case, we assume the whole procedure of handling a piece is itself a whole task, which we refine progressively into more concrete goals (picking the piece, verifying its shape and inserting them).

The contribution of Fraser *et al.* [8] revolves around compactly encoding large sets of execution paths with a tree structure in which nodes represent behaviours. The latter may represent atomic actions or an aggregation (AND, OR, THEN) of other behaviours. This structure generalizes HTN in the sense that plans can be modified online thanks to an activation spreading mechanism. While we have opted for HTN, we believe this framework to be compatible with ours and may constitute an idea for future improvement.

III. HTN FORMALISM

For the sake of clarity in future sections, we summarize here the fundamentals of HTNs. HTNs are a family of planners that, like classical planners such as *Fast Forward* [9] or *LAMA* [10], accept a declarative description of the domain in the form of predicates that describe the planning state, and operators that can be applied to modify the state. However, HTNs are *task driven* (as opposed to action driven, like other planners). This means that HTNs aim at achieving one or more tasks, instead of reaching a goal state.

In order to fully specify the domain for a HTN planner, one has to include the description of the tasks and methods as well. Of course, these must be tailored to each particular problem. In this sense, HTN planners sacrifice part of the generality and ready-to-go capabilities of conventional planners that only need a very basic description of the domain mechanics. In turn, the inclusion of the tasks and methods can lead to a highly efficient planning procedure, since the task hierarchy encodes exclusively the paths that are relevant for solving a problem. This is the reason why we have chosen the HTN paradigm: it can encode quite complex control structures that may involve one or two robotic arms, conditioned by different geometrical constraints.

Tasks may be *primitive* or *compound*. When we use the term *task* we are referring to a task that belongs to either of these types. A primitive task is essentially a wrapper around an operator, and can be accomplished immediately provided that the preconditions of the operator are satisfied. On the other hand, compound tasks have to be decomposed into increasingly simple sub-tasks, using one of the possibly many methods that allow this decomposition. We will use the term *sub-task* for tasks that come from the decomposition of a task that precedes it in the task hierarchy.

Methods are the mechanism that allow a compound task to be decomposed into one or more sub-tasks, which are added to the planning stack (we stick to the case in which the tasks are achieved *in-order*). One can define several methods for the same task. In this case, the planner would try them

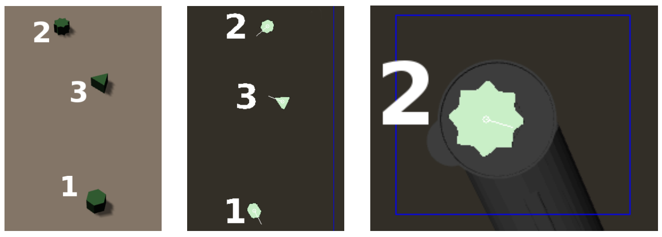


Fig. 2. (a) Green pieces as seen in the simulator. Pieces 1, 2 and 3 have a hexagonal, octagram-like and triangular base, respectively. (b) Segmentation of the pieces while they are resting on the table. (c) Segmentation of the octagram while being held by the robot arm near to the camera.

one by one until finding a valid plan, or until there are no possibilities left. Methods have a set of preconditions that dictates under which circumstances they can be used to perform the decomposition.

An HTN planner is ready to operate when provided with a domain description and a set of tasks to be achieved. For the sake of brevity, we will identify this set as a *goal* (not to be confused with the specification of the goal state for a conventional planner). Sometimes, we will talk about a *subgoal* when referring to one of the tasks in this set (e.g. the insertion of an individual piece). For a more comprehensive understanding of HTNs, we recommend reading Nau *et al.*'s work about SHOP2 [11].

IV. Handling noisy percepts

The detection and identification of the pieces consists of colour segmentation and XOR-based shape comparison. It is not our intention to cover here the details of the perception pipeline since they are not relevant from a planning perspective. Instead, we would rather discuss their implications in the development of the proposed framework. One of our fundamental premises is that perception is noisy and the system may have to consider additional observation actions to obtain a better state estimation.

First of all let us identify the source of the uncertainty and an adequate solution using our example. Fig. 2 illustrates the main perception issue tackled, as described in this section. The distance and perspective (Fig. 2a) makes it difficult to identify the correct shape of the segmented *blobs* (see Fig. 2b). The solution we propose to this issue is to add an uncertainty *reduction* action to the repertoire of the robot for bringing the piece closer to the camera and then perform a careful examination. We show in Fig. 2c that this leads to a much clearer contour and, therefore, a more accurate guess.

Carefully observing all the pieces has associated a significant overhead. Therefore it is important to quantify the uncertainty so we can actually plan when an observation action is necessary. See for instance that in Fig. 2b the triangle can be easily identified without close examination. The same happens when there is a single piece of a certain colour in the scene. Next, we outline our strategy for dealing with this, which is generally applicable to reduce uncertainty when discriminating objects within a given set.

Let us notice that there is just one piece per shape and that the sets of shapes per colour are disjoint (e.g. a triangle shall always be green, and green pieces consist exclusively of the triangle, the octagram and the hexagon). We define a matrix $S = (s_{ij})$, where the element s_{ij} is a number between 0 and 1 (calculated by the perception algorithm) that represents the similarity between the ith detected blob and the jth shape. Now, we are interested in obtaining from S a matrix of probabilities $P = (p_{ij})$. The p_{ij} element represents the probability of the $i \to j$ blob/shape assignment.

There are many ways of deriving such a matrix P from the similarity matrix. We have chosen to use *matrix scaling* to generate a *doubly stochastic matrix* [12] that meets our requirements. However, there are other possibilities like using undirected probabilistic graphical models (e.g. pairwise Markov networks).

V. Domain specification

Next we describe the information contained in the planning state, the available operators and the HTN methods.

A. State specification

The state is represented as a Python object. The problem information is encoded with built-in types (e.g. list, dictionaries, strings) as attributes. More specifically the state gathers the following attributes:

- `joints`: virtual representation of the robot state, essential for feeding the inverse kinematics solver.
- `tool_status`: a boolean fluent indicating whether the gripper is open or closed.
- `holding`: id (numeric index) of the currently held object, or *None* if the gripper is empty.
- `objects`: a vector of piece locations.
- `at`: a dictionary from robot names (*picker* and *catcher*) to potentially parameterized pose identifiers (e.g. *(show-cavity, hexagon)*).
- `P`: the matrix of assignment probabilities.
- `available_shapes`: the remaining shapes' names (those that have not been inserted yet according to the robot's information) as a vector. Moreover, The similarity and probability matrices' column ordering is the same as in this list.
- `cost`: metric that we seek to minimize. In our case we are measuring the execution time, although it is possible to focus on any other measure like safety.

In the following we use the square brackets ($[\cdot]$) notation to denote the *access-by-key* operation (both for arrays and dictionaries). Although matrices and lists are indexed by non-negative integers (0-based indexing) we abuse notation and employ the shape names to index the `S` and `P` columns for simplicity. We will use colons inside the brackets to denote the *slice* operation.

B. Operators

Now let us explain the different actions that we have considered in this domain. These actions are typically known as *primitive tasks* or *operators* in the HTN paradigm. For each operator we provide a natural language description along the pre-conditions and post-conditions:

`move_j_one(robot, pose-id)`
Description: Move one of the arms in joint space avoiding obstacles and self-collisions.
Pre: `at[robot]` $\neq$ `pose-id` $\wedge\ \exists$ motion plan
Post: `at[robot]` $\leftarrow$ `pose-id`

`move_j_both(pose-id1, pose-id2)`
Description: Moves both arms simultaneously.
Pre: `at`[*picker*] $\neq$ `pose-id1` $\wedge$ `at`[*catcher*] $\neq$ `pose-id2` $\wedge\ \exists$ motion plan
Post: `at`[*picker*] $\leftarrow$ `pose-id1` $\wedge$ `at`[*catcher*] $\leftarrow$ `pose-id2`

`push(blob)`
Description: *picker* action for pushing a piece. Used for isolating pieces. The movement is organized in 4 waypoints (transitions in Cartesian space).
Pre: `at`[*picker*] $\leftarrow$ (*push-stage*, `blob`, 0) $\wedge\exists$ motion plan
Post: `objects[blob]` modified according to push direction $\wedge$ `at`[*picker*] $\leftarrow$ (*push-stage*, `blob`, 3)

`pick(blob)`
Description: *picker* action for grasping a piece that is resting on the table. The movement is organized in 3 waypoints, and the transitions are done in Cartesian space.
Pre: `holding` = $\varnothing$ $\wedge$ `at`[*picker*] = (*pre-grasp*, `blob`) $\wedge\exists$ motion plan
Post: `holding` $\leftarrow$ `blob` $\wedge$ `at`[*picker*] $\leftarrow$ (*post-grasp*, `blob`)

`insert(blob, cavity)`
Description: *picker* action for inserting the piece into a particular cavity of the sphere. It removes the piece from the problem along with the cavity's shape. It modifies the `objects` and `available_shapes` arrays, and the `S` and `P` matrices.
Pre: `holding` = `object-id` $\wedge$ `at`[*catcher*] = *(show-cavity, cavity)* $\wedge$ `at`[*picker*] = *(pre-ungrasp, insert-pose)* $\wedge\exists$ motion plan
Post: *delete*(`objects`, `blob`) $\wedge$ *delete*(`available_shapes`, `cavity`) $\wedge$ *delete_row*(`S`, `blob`) $\wedge$ *delete_column*(`S`, `cavity`) $\wedge$ `P` $\leftarrow$ *scale_matrix*(`S`)

`examine(blob, shape, min_sim, max_ent)`
Description: close examination of a piece. The action will fail if the label of the grasped piece cannot be identified with enough certainty.
Pre: `at`[*picker*] = (*close-look*) $\wedge$ `holding` = `blob`
Post: `S[blob, shape]` > `min_sim` $\wedge$ *normalized_entropy*(`P[blob,:]`) < `max_ent` $\wedge$ $argmax_s$(`P[blob,:]`) = `shape`

The `examine` and `push` are, respectively, an *uncertainty reduction* operator and a *reshaping* operator, according to the action classes presented in Section I.

C. Compound tasks and methods

Fig. 3 shows the hierarchy of tasks and method for handling a single piece, to be completed *in-order*. That is, methods are tried from left to right and tasks are completed from left to right as well. In order to account for the possibility of a task being already done, we include methods that succeed trivially with an empty task decomposition should the task be already completed. For instance this is the case of `grasped` and `robot-at`. In general we have grouped the cheapest methods (the ones that require less execution time) at the left hand side of the tree so these are tried first.

VI. Experimental evaluation

Our framework has been implemented in a *ROS Kinetic* environment and tested both in simulation and in the real world. The interested reader can find demonstration videos in the following web page: `http://www.iri.upc.edu/groups/perception/dual_arm_planning/`. The employed motion planning library is *MoveIt!* [13]. We have used *TRAC-IK* [14] as the IK solver as in our case provides better results than the *KDL* plugin.

We have simulated the set-up via *Gazebo*. We have evaluated six different algorithms in three scenarios that are meant to pose increasing levels of difficulty (see Fig. 4).

The first three algorithms (named **A**, **B** and **C**) evaluate strategies about the perception noise and do not consider bimanual planning, while the three next algorithms (named **D**, **E** and **F**) are variations of the framework presented in this paper:

- **A**: *always believe perception*. Automaton with no symbolic planning whatsoever. Only one arm moves at a given moment. The arms adopt a resting (home) position frequently to avoid collisions between them. No close examinations are performed. The next piece to insert is selected randomly. This is the simplest strategy that solves the proposed problem. We expect it to fail occasionally due to the non-ideal perception.
- **B**: *never believe perception*. Similar to **A**, but always performing careful examinations. We include this algorithm to assess the effectiveness of the careful examination actions. We foresee that this algorithm will

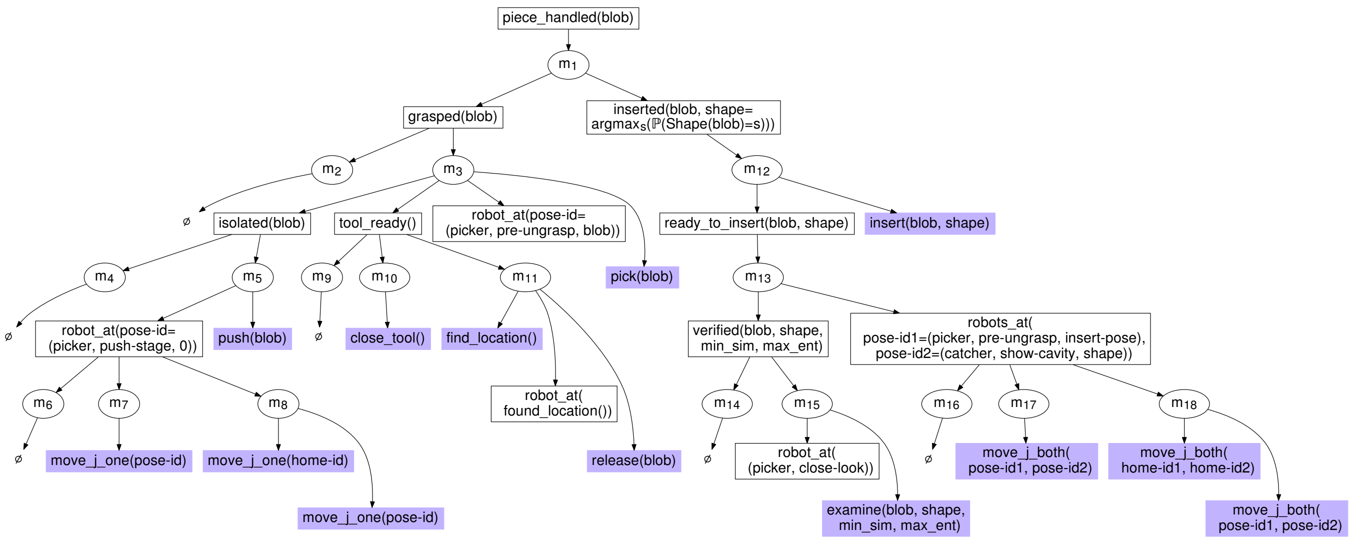

piece_handled(blob)
m1
grasped(blob)
inserted(blob, shape= argmax_s(ℙ(Shape(blob)=s)))
m2
m3
m12
∅
isolated(blob)
tool_ready()
robot_at(pose-id= (picker, pre-ungrasp, blob))
ready_to_insert(blob, shape)
insert(blob, shape)
m4
m5
m9
m10
m11
pick(blob)
m13
∅
robot_at(pose-id= (picker, push-stage, 0))
push(blob)
∅
close_tool()
find_location()
verified(blob, shape, min_sim, max_ent)
robots_at(pose-id1=(picker, pre-ungrasp, insert-pose), pose-id2=(catcher, show-cavity, shape))
m6
m7
m8
robot_at(found_location())
m14
m15
m16
m17
m18
∅
move_j_one(pose-id)
move_j_one(home-id)
release(blob)
∅
robot_at((picker, close-look))
∅
move_j_both(pose-id1, pose-id2)
move_j_both(home-id1, home-id2)
move_j_one(pose-id)
examine(blob, shape, min_sim, max_ent)
move_j_both(pose-id1, pose-id2)

Fig. 3. Hierarchy of tasks and methods. Tasks are represented with boxes. Primitive tasks are shaded and are explained in more detail in Section V-B. Methods are represented with ellipses.

not make any mistake. However, we believe that the execution of unnecessary actions will lead to a high execution time.

- **C**: *examine when necessary*. Symbolic planning without collision checking nor any other geometric consideration. The home position is adopted frequently to avoid collisions. Observations are planned according to the strategy described in section Section IV. The piece selected at each step is the one with least entropy. We expect it to be closer to **A** in terms of execution time, but without errors.
- **D**: *lowest cost plan*. This algorithm conforms to the framework presented in this paper. The strategy for choosing the next piece is to select the task with least cost, i.e., the plan for each piece is computed and then the one with the minimum execution time is selected. We believe this will achieve a very low execution time.
- **E**: *lowest cost plan with a longer horizon*. Similar to **D** but planning ahead with a horizon of 2, i.e., deciding at each step the combination of two pieces (considering exhaustively all the pairs) with least cost. It is our intuition that this will result in lower execution time, but at the cost of increasing greatly the complexity of planning.
- **F**: *simple position-based heuristic*. Another variation of our framework. In this case the pieces with the lowest Y coordinate (nearest to the *catcher*) are handled first. The rationale of this heuristic is that these are the pieces that are most likely to make the *catcher* and the *picker* interfere with each other. We expect the heuristic to be beneficial specially at the beginning. Moreover it does not rely on the planning cost to select the next piece, so we believe that the time spent planning will be notably lower.

The comparison is done in relation to the following measures:

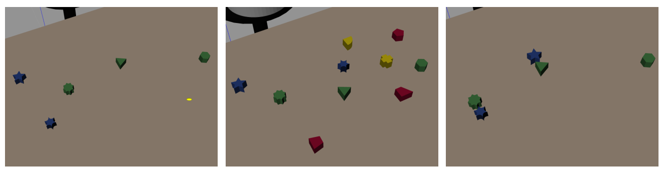

(a) (b) (c)

Fig. 4. Considered scenarios in the experimental evaluation. (a) Simplified scenario with 5 pieces out of 10. (b) More complex scenario with the 10 pieces. (c) Scenario in which the *push* action is required to grasp some pieces.

TABLE I

COMPARISON OF THE DIFFERENT ALGORITHMS IN THE PROPOSED SCENARIOS. RESULTS FOR FIRST (FIG. 4A), SECOND (FIG. 4B) AND THIRD (FIG. 4C) SCENARIOS SHOWN IN (A), (B) AND (C), RESPECTIVELY.

Alg.	Plan. (s)	Exe. (s)	Time (s)	#Re-plan	#Mist.
A	-	140.3 ± 7.6	140.3 ± 7.6	-	2
B	-	165.6 ± 8.0	165.6 ± 8.0	-	**0**
C	**0.3** ± 0.1	140.4 ± 8.3	140.7 ± 8.4	2	**0**
D	74.8 ± 11.6	93.4 ± 9.7	168.2 ± 11.5	[2, 4]	**0**
E	264.4 ± 47.1	94.7 ± 6.6	359.1 ± 48.5	[2, 4]	**0**
F	24.7 ± 3.4	**91.6** ± 6.3	**116.3** ± 7.2	2	**0**

(a)

Alg.	Plan. (s)	Exe. (s)	Time (s)	#Re-plan	#Mist.
A	-	280.8 ± 9.3	280.8 ± 9.3	-	2
B	-	325.7 ± 9.0	325.7 ± 9.0	-	**0**
C	**0.7** ± 0.2	278.9 ± 9.7	279.6 ± 9.5	2	**0**
D	247.0 ± 8.9	178.0 ± 12.4	425.0 ± 18.5	[4, 6]	**0**
F	41.9 ± 1.9	**176.2** ± 6.9	**218.1** ± 6.7	4	**0**

(b)

Alg.	Plan. (s)	Exe. (s)	Time (s)	#Re-plan	#Mist.
D	80.0 ± 8.8	**103.7** ± 6.0	183.7 ± 10.2	2	**0**
F	**26.0** ± 3.0	106.4 ± 9.6	**132.3** ± 8.7	2	**0**

(c)

- **Time**: Time until the completion of the experiment. This time is broken down into **plan time** and **execution time**.
- **#Re-plan**: number of times that the planner throws away a plan and recomputes a new one (typically because the grasped piece is not the expected one after close examination). Each re-plan adds an average of 4.98s to the total time for algorithms **D**, **E** and **F**. The contribution to the total time in the case of **C** is considered negligible.
- **#Mistakes**: number of times that the robots try to insert a piece in the wrong cavity. This is a non-recoverable and severe failure. Re-planning is preferred.

The results are shown in Table I. Experiments were executed five times to obtain the average and the standard deviation of the time measures. For the **#Re-plan** and **#Mistakes** columns, the minimum and the maximum are reported whenever they vary among the experiments.

The results show that examinations do contribute to avoid mistakes. In all the experiments **A** consistently made two mistakes, while in the other algorithms these mistakes are replaced either by a significant increment in the execution time (**B**) or by re-planning (**C**, **D**, **E**, **F**). The speed-up of **C** with respect to **B** (while maintaining at 0 the number of mistakes) evidences the effectiveness of the strategy proposed in Section IV to handle uncertainty.

Much as we expected, the execution time when considering simultaneous movements of both arms is lower. This can be already appreciated in **D**. However, **D** has associated a very high planning time, so it falls behind both **B** and **C** in terms of overall time.

On the other hand, considering a horizon of 2 as in **E** scales badly, yielding impractical high planning times. Contrarily to our expectations, the execution time seems to be unaffected. For this reason, we stopped considering **E** in scenarios 2 and 3.

In fact, in our experiments the usage of a simple heuristic (as that of **F**) for deciding the next task results in much less planning time with no noticeable impact on the execution. This suggests that the greatest speed-up comes from the management of the two arms rather than from finding the best ordering of the pieces. To confirm this, we have computed the execution time associated to all the permutations of pieces in the first scenario. The lowest execution time is around 84.9s which is not much better than the lowest execution time of Table I (91.6s). However, the ordering of the sub-goals may be much more relevant in other applications.

As shown at the beginning of the section, we also performed a qualitative assessment of our implementation with real WAM arms. The grasping and insertion tasks require high precision, that could be achieved up to certain extent.

VII. Conclusions and future work

This paper outlines a strategy to combine symbolic and motion planning to perform manipulation tasks with dual manipulators. We use HTNs to encode different sequences of movements for both arms and to handle possible interferences between them. We tackle situations in which the sub-goals can be easily identified and it is possible to plan individually for each of them. At the same time we deal with uncertainty about the current state and consider two types of actions to reduce it, namely observation actions and environment reshaping actions. We have successfully tested our strategy on a puzzle-like problem with WAM arms. The experiments conducted in simulation suggest that our framework can deal naturally with problems that require bi-manual capabilities, reducing the execution time with respect to more conservative algorithms. The proposed framework has been tested with real-world robots, as well.

In future work we would like to put emphasis in dealing with dynamic, factoring in exogenous effects [15]. Moreover, we believe that more sophisticated decomposition strategies (like the one proposed by Fraser *et al.* [8]) could be incorporated to our framework to solve tasks in more complex scenarios. Another line of work that we would like to explore is the optimization of other metrics instead of (or in addition to) execution time.

References

[1] D. Martínez, G. Alenya, and C. Torras, “Relational reinforcement learning with guided demonstrations,” *Artificial Intelligence*, vol. 247, pp. 295–312, 2017.

[2] L. P. Kaelbling and T. Lozano-Pérez, “Hierarchical task and motion planning in the now,” in *IEEE International Conference on Robotics and Automation*, 2011, pp. 1470–1477.

[3] L. P. Kaelbling and T. Lozano-Pérez, “Integrated task and motion planning in belief space,” *International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[4] J. Ferrer-Mestres, G. Francès, and H. Geffner, “Combined task and motion planning as classical ai planning,” *arXiv preprint arXiv:1706.06927*, 2017.

[5] M. Ghallab, D. Nau, and P. Traverso, “The actor’s view of automated planning and acting: A position paper,” *Artificial Intelligence*, vol. 208, pp. 1–17, 2014.

[6] D. S. Nau, M. Ghallab, and P. Traverso, “Blended Planning and Acting: Preliminary Approach, Research Challenges.” in *AAAI*, 2015, pp. 4047–4051.

[7] B. Marthi, S. J. Russell, and J. A. Wolfe, “Angelic hierarchical planning: Optimal and online algorithms.” in *ICAPS*, 2008, pp. 222–231.

[8] L. Fraser, B. Rekabdar, M. Nicolescu, M. Nicolescu, D. Feil-Seifer, and G. Bebis, “A compact task representation for hierarchical robot control,” in *IEEE International Conference on Humanoid Robots*, 2016, pp. 697–704.

[9] J. Hoffman and B. Nebel, “The FF Planning System: Fast Plan Generation Through Heuristic Search,” *Journal of Artificial Intelligence Research*, vol. 14, no. 27, pp. 253–302, 2001.

[10] S. Richter and M. Westphal, “The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks,” *Journal of Artificial Intelligence Research*, vol. 39, pp. 127–177, 2010.

[11] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, “SHOP2: An HTN planning system,” *Journal of Artificial Intelligence Research (JAIR)*, vol. 20, pp. 379–404, 2003.

[12] R. Sinkhorn and P. Knopp, “Concerning nonnegative matrices and doubly stochastic matrices,” *Pacific Journal of Mathematics*, vol. 21, no. 2, pp. 343–348, 1967.

[13] I. A. Sucan and S. Chitta. ”MoveIt!”. [Online]. Available: http://moveit.ros.org

[14] P. Beeson and B. Ames, “TRAC-IK: An open-source library for improved solving of generic inverse kinematics,” in *IEEE International Conference on Humanoid Robots*, 2015, pp. 928–935.

[15] D. Martínez, G. Alenya, T. Ribeiro, K. Inoue, and C. Torras, “Relational reinforcement learning for planning with exogenous effects,” *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 2689–2732, 2017.