2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

An Interface for Interleaved Symbolic-Geometric Planning and Backtracking

Lavindra de Silva Amit Kumar Pandey Rachid Alami

Abstract— **While symbolic planners work with an abstract representation of the real world, allowing plans to be constructed relatively quickly, geometric planning—although more computationally complex—is essential for building symbolic plans that actually work in the real world. To combine the two types of systems, we present in this paper a meaningful interface, and insights into a methodology for developing interwoven symbolic-geometric domains. We concretely present this "link" between the two approaches with algorithms and data structures that amount to an intermediate layer that coordinates symbolic-geometric planning. Since both planners are capable of "backtracking" at their own levels, we also investigate the issue of how to interleave their backtracking, which we do in the context of the algorithms that form the link. Finally, we present a prototype implementation of the combined system on a PR2 robot.**

I. Introduction

In the past decade we have witnessed much interest in bridging the gap between symbolic and geometric planning. Some have focused on defining how geometric entities and planning capabilities can be meaningfully used within symbolic planning, and how symbolic information can be used as heuristics in geometric planning. These efforts have included defining the link between geometric and symbolic actions, specifying what information should be shared with the symbolic (resp. geometric) planner, and how this information should be used in symbolic (resp. geometric) states, actions and planning algorithms. One issue that must be addressed when combining the two systems is "backtracking," i.e. trying alternative options for choices that were already made during planning. Both systems backtrack at their own "levels" when a plan being pursued turns out to not work: the symbolic planner when some precondition is not met, and the geometric planner when a planned path includes a collision. It is therefore important to investigate the relation between symbolic and geometric backtracking in order to interleave them effectively and know when to switch between them.

In [1], [2] the authors present algorithms for geometric backtracking. They extend the Justin robot with symbolic-geometric planning capabilities: for symbolic planning they use the JSHOP2 [3] totally-ordered HTN planner, and for geometric planning they use a specialised path planner. The authors follow the approach of keeping the symbolic state orthogonal to the geometric state: e.g. changing in the geometric state the pose of an object has no consequence on the symbolic state. In contrast, we require symbolic and geometric states to be "intertwined," which we find to be natural in some domains. To this end, we derive symbolic facts from the geometric state and use them in symbolic planning. Nevertheless, the geometric planner in [1], [2] is much like ours in how it backtracks when an action being considered by the symbolic planner is not applicable, forcing the system to reconsider previous geometric choices (e.g. the pose of a cup on the table) to make the new action applicable (e.g. changing the cup's pose to make room for another one). Since we derive and use geometric states as symbolic facts, however, we also need to address the ramifications of such backtracking on already pursued symbolic actions.

Also related to our objectives is work on interfacing symbolic and geometric planning. In [4], [5] the authors introduce Semantic Attachments, which associate selected predicates in the planning domain to external procedures, called at runtime to evaluate the predicates. In [6], Semantic Attachments are implemented using a trajectory planner that computes collision free trajectories, the existence of which causes the associated Semantic Attachments to evaluate to $true$, and $false$ otherwise. Likewise, "effect applicators" in effects of actions consult the geometric planner to set certain state variables (e.g. *position* and *orientation*) in the symbolic domain. Due to such variables being deterministic, effect applicators cannot "choose" between different outcomes. In contrast, it is important in our work to give the geometric planner some leeway to make such choices.

Some systems do not make a clear distinction between symbolic and geometric planning. In [7], a hierarchical planner plans all the way down to the level of geometric actions, and symbolic and geometric backtracking happens in the same "space" of solutions. Their hierarchical planner is a specialised one where primitive actions are implemented by invoking external solvers like rapidly-exploring random trees (RRTs), and states model low-level geometric details like robot joint configurations. Similarly, [8], [9] describes a hierarchical planner that is tightly integrated with a geometric motion planner for planning and then executing the most basic actions. In their work, however, geometric actions are executed while the plan is being constructed. This is different to the other work described above which formulates a complete plan first, before executing it. Finally, the Asymov [10] system is a combined task and motion planner for problems that are difficult to solve when the symbolic planner is in control of the geometric search, e.g. in the geometric Towers of Hanoi problem that the authors present. Unlike the above approaches, in Asymov

*We thank Malik Ghallab and the anonymous reviewers for their valuable feedback, and Mamoun Gharbi for Figure 2 and help with experiments. This work was conducted within the EU SAPHARI project (www.saphari.eu) funded by the E.C. division FP7-IST under contract ICT-287513.

**CNRS, LAAS, 7 avenue du Colonel Roche, F-31400 Toulouse, France.
***Univ. de Toulouse, LAAS, F-31400 Toulouse, France.
{ldesilva,akpandey,rachid}@laas.fr

the geometric planner uses the symbolic planner—as well as the symbolic model of the domain—as a heuristic when choosing roadmaps during geometric search. Similarly, in [11] a symbolic planner guides a sampling-based motion planner, which in turn sends back utility estimates to improve the guide in the next iteration.

To summarise, the contributions of this paper are the following. First, we present a precise interface between symbolic and geometric planning. In particular, we investigate the "right" level of abstraction at which the two approaches should be combined. To this end, unlike other work, we use a geometric *task* planner [12], which lets us define the interface to symbolic planning in a more meaningful way, and also give more leeway to the geometric level to make decisions. Second, we provide algorithms and data structures for an intermediate layer that coordinates symbolic and geometric backtracking and planning, as well as algorithms that highlight the characteristics of the underlying geometric task planner. Finally, we propose a methodology for developing an intertwined symbolic and geometric planning domain, including a concrete example with the PR2 robotics platform [13] in the context of Human-Robot Interaction (HRI).

II. Background

In this paper we use a STRIPS-like planning language [14]. Therefore, we briefly recall some relevant STRIPS notions. A STRIPS ***action*** is a ground instance of a STRIPS operator, or for convenience, just a ground instance of the operator's name. Given an action a and a state S such that $S \models pre(a)$ (i.e. the precondition of a holds in S), we define the ***result*** of applying a to S (with respect to a set of operators), denoted $Res(a, S)$, as $(S \setminus \mathit{eff}(a)^-) \cup \mathit{eff}(a)^+$, where $\mathit{eff}(a)$ is a set of ground literals denoting the effects of a. This definition trivially extends to a sequence of actions $a_1 \cdot \ldots \cdot a_n$: the result of applying $a_1 \cdot \ldots \cdot a_n$ to a state S, denoted $Res(a_1 \cdot \ldots \cdot a_n, S)$, is S if $|a_1 \cdot \ldots \cdot a_n| = 0$ and $Res(a_2 \cdot \ldots \cdot a_n, Res(a_1, S))$ otherwise.

While classical planners focus on bringing about states of affairs or "goals-to-be," Hierarchical Task Network (HTN) planners focus on solving *abstract tasks* or "goals-to-do." In this paper we use a popular type of HTN planning called "totally-ordered" HTN (henceforth simply referred to as HTN) planning, which, more importantly than efficiency, allows calls to external functions [15]—an important requirement in our work. An HTN ***planning problem*** is the 3-tuple $\langle d, S_0, \mathcal{D} \rangle$, where d is the sequence of (primitive or abstract) tasks to solve, S_0 is an initial state as in classical planning, and $\mathcal{D}$ is an HTN *planning domain*. Specifically, a HTN ***planning domain*** is the pair $\mathcal{D} = \langle \mathcal{A}, \mathcal{M} \rangle$ where $\mathcal{A}$ is a finite set of operators as before, and $\mathcal{M}$ is a finite set of *methods*. A ***method*** is a tuple consisting of the name of the method, the abstract task that the method is used to solve, a precondition specifying when the method is applicable (like an operator's precondition), and a *body* indicating which tasks are used to solve the task associated with the method. The method-body is a sequence of primitive and/or abstract tasks. The planning process works by selecting applicable reduction methods from $\mathcal{M}$ and applying them to abstract tasks in d in a depth-first manner. In each iteration this will typically result in d becoming a "more primitive" sequence of tasks. The process continues until d has only primitive tasks left. At any stage during planning if no applicable method can be found for an abstract task, the planner essentially "backtracks" and tries an alternative method for an abstract task refined earlier.

For the geometric counterpart, we adopt the approach of finding a solution in a discrete space of candidate grasps and placements [2], [12] for tasks involving picking and placing. Basically, our Geometric Task Planner (GTP) iterates in a four dimensional search space, consisting of a set of *agent "effort" levels*, *discrete grasps*, *object placement positions*, and *object placement orientations*. For each object, sets of possible grasps are pre-computed and stored for the anthropomorphic hands and the robot's gripper, which are later filtered based on task requirements and the environment. The amount of "effort units" required to perform certain tasks, e.g. moving the head, extending an arm, and standing, are predefined; to this end, we have made simplifying assumptions about which tasks require less effort than others (e.g. we assume that extending an arm requires less effort than standing up). At runtime, sets of placement positions and possible orientations of objects are dynamically extracted, based on the environment, the task, and restrictions on how much effort should be put into the task. The sets are then weighted based on the environment and situation, with criteria like grasp and placement stability, feasibility of simultaneous grasps by two agents, and the agent's visibility of the object. These factors are used to build the final configuration which is then used online with an RRT based planner [16] to find a feasible trajectory. Computed RRTs can be reused for successive planning requests whenever feasible. We note that while the GTP is not complete in the sense of planning in a continuous search space, if there is a solution in the discrete search space of the task, it will find one.

The advantage of the GTP framework is that a variety of day-to-day tasks like showing and giving an object can be represented in terms of different constraints, based on factors like object reachability and the ability to grasp an object. A geometric solution is found using a constraint hierarchy based approach, by carefully introducing constraints successively at the different stages of planning. This facilitates the reduction of the search space successively, before introducing relatively more computationally expensive constraints.

In this paper we use the following conventions: symbol ϵ is the empty string/sequence, variables begin with upper case, and predicates and constants with lower case. We assume that symbolic and geometric states are fully observable, and allow negative literals and universal quantification in preconditions of actions and HTN methods. Finally, this paper only deals with planning: we do not interleave planning with execution.

III. A suitable interface

The HTN developer interfaces with the GTP using evaluable predicates. Truth values to evaluable predicates are not

assigned in the conventional way—by referring to the state of the world—but by calling an associated external procedure, which returns $true$ or $false$. So every (relevant) GTP task t is associated with a non-negated evaluable predicate, denoted $t^?$, in the HTN domain (e.g. GTP task $\text{SHOW}(O, H)$ is associated with evaluable predicate $show(O, H)^?$ for some object O and human H), which basically evaluates to $true$ if t has a GTP solution and $false$ otherwise. Whenever such an evaluable predicate is mentioned in the precondition of an operator (resp. action), we call it a ***Geometric-Symbolic*** (GS) operator (resp. action).

Conceptually, a GTP task t is also associated with an add list function, denoted t^+, and a delete list function, denoted t^-, which are the (possibly empty) add and delete lists for t computed by the GTP, based basically on the world resulting from the solution that was found for t on calling evaluable predicate $t^?$.[1] For example, for GTP task $\text{SHOW}(o_1, h_1)$, $show(o_1, h_1)^+$ might return the set $\{visible(o_1, h_1), reachable(o_1, h_1)\}$ and $show(o_1, h_1)^-$ the set $\{visible(o_3, h_1), reachable(o_3, h_1)\}$: that is, object o_1 is both visible to human h_1 and reachable to h_1, but object o_3 is neither visible nor reachable to h_1. Such "side effects" can only accurately be inferred at runtime, from the most recent state of the geometric 3D world. Hence, the effects of a GS operator are the combination of its add and delete lists with its "dynamic" add and delete lists, obtained via the add and delete list functions.

It is sometimes useful to share certain symbolic predicates between the HTN planner and the GTP. A "shared predicate" (or "shared literal") is an abstract relation between objects that needs to be modelled in the HTN domain, but is based on geometrical properties and consequently modelled more accurately by the GTP. For example, predicate $on(b_1, b_2)$ in the HTN domain is an abstract representation specifying that book b_1 is on top of b_2, which in the GTP amounts to two rectangular objects in 3D space conforming to certain constraints like one object's side touching the other's. Likewise, in the HTN domain $visible(o, h)$ specifies that object o is visible to human h, which is computed by the GTP based on whether the human's field of view in a 3D world overlaps with the object [17]. There are, of course, also symbolic predicates that are specific to the HTN planner (such as $full(c)$, $breakable(c)$, and $emailed(p)$ for some cup c and person p) which are not modelled by the GTP. Furthermore, low-level predicates specific to the GTP, e.g. the exact position of an object in terms of (x, y, z) coordinates, is often too much detail for the HTN domain.

It is worth mentioning that checking if a symbolic predicate *holds* in a geometric world state requires anchoring [18] the predicate (e.g. $visible(o, h)$ for some object o and human h) to the right geometrical attributes (e.g. position, orientation and field of view). We refer the reader to [19], [17] for a detailed account of how we address this problem. In this paper we ignore the issue of anchoring a ground literal to its corresponding geometrical attributes, and assume that we can simply check if a ground literal holds in a geometric world state.

With sharing symbolic predicates between the geometric and symbolic planners comes the need to manage consistency in the two world representations. Specifically, we may reach a point during symbolic-geometric planning where there is an inconsistency between shared (ground) literals implied by the GTP world state and those actually in the HTN state. This could happen, for instance, if the HTN developer incorrectly predicts, while writing the domain, the side effects of the $\text{SHOW}(o_1, h_1)$ action mentioned before. We address this issue by disallowing the HTN developer from mentioning shared predicates in add and delete lists. Specifically, such predicates can only be added to and removed from the HTN state via add list and delete list functions. Likewise, when constructing the HTN initial state, the GTP must be consulted to get the shared (ground) predicates that hold in the initial GTP state.

We will now be precise about how we interface the geometric planner to the symbolic. A GS operator maps to exactly one GTP task: only one evaluable predicate that corresponds to a GTP task can appear in an operator's precondition. Intuitively, this means that the lowest "level of abstraction" in an HTN domain—i.e. primitive tasks—is the highest "level of abstraction" in the GTP domain—i.e. compound GTP tasks. This can always be accounted for, as a GTP task can be as "compound" as necessary, where a compound GTP task encompasses multiple, more specific ones that are too "low level" to be included in the HTN domain. For example, consider the rightmost (fourth) column of Figure 2 which shows the grasp and placement actions that the four "compound" GTP tasks in the third column refine into. We could imagine a domain where these actions are encapsulated into one compound $\text{PICKMAKEACCESSIBLE}(O, H)$ task (for object O and human H), instead of the two tasks $\text{PICK}(O)$ and $\text{MAKEACC}(O, H)$. This will let the GTP backtrack from $\text{PLACE}(O)$ to $\text{GRASP}(O)$, facilitating early failure detection in HTN planning—i.e. $\text{PICKMAKEACCESSIBLE}(O, H)$ will have no solution when the "future" GTP task $\text{PLACE}(O)$, due to be tried later by the HTN planner, is predicted by the GTP to be impossible even with different solutions for $\text{PICK}(O)$. Such encapsulation can also minimise GTP backtracking by, from the outset, planning with respect to definite future GTP tasks (e.g. planning the grasp with respect to the placement).

IV. A SYMBOLIC-GEOMETRIC PLANNING EXAMPLE

In this section we detail a concrete domain that illustrates how symbolic and geometric planning is interfaced and combined. Particularly, we show how HTN and GTP backtracking is interleaved, and shed light on a methodology for building symbolic-geometric planning domains. Our example is a PR2 robot working as a library receptionist. We focus here on the important aspects of this domain and refer the reader to [20] for more details.

Members of the library can use their membership ID to do things like reserve books online and top up their library

[1] Add and delete list functions are only conceptual entities. Later, in the algorithms, we present a slightly different representation.

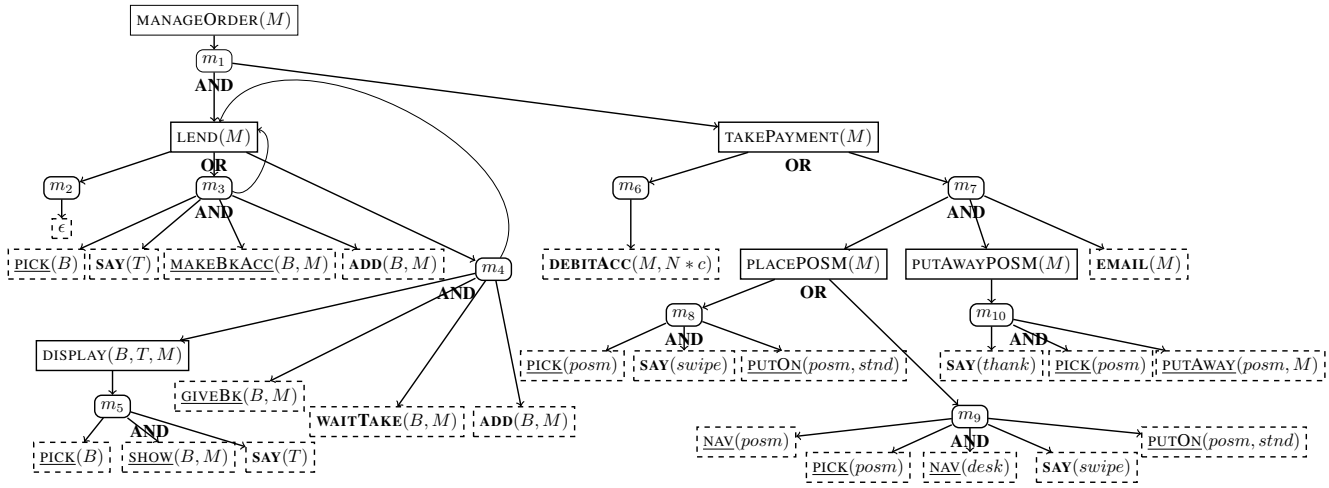


Fig. 1: The part of the HTN domain that handles online book reservations. Solid rectangles are HTN abstract tasks; rounded rectangles methods (where the incoming vertex is the task that the method solves, and outgoing vertices are the tasks in the method's body); and dashed rectangles actions. Standard HTN actions are in bold and GS actions are underlined.

ACTION/HTN TASK	M	PRECONDITION	METHOD-BODY/ACTION-EFFECTS
MANAGEORDER(M)	m_1	$held(B, M)$	LEND(M) · TAKEPAYMENT(M)
LEND(M)	m_2	$\forall B, \neg held(B, M)$	ϵ
	m_3	$held(B, M) \wedge title(B, T)$	PICK(B) · **SAY**(T) · MAKEBKACC(B, M) · **ADD**(B, M) · LEND(M)
	m_4	$held(B, M) \wedge title(B, T) \wedge \neg hvy(B)$	DISPLAY(B, T, M) · GIVEBK(B, M) · **WAITTAKE**(B, M) · **ADD**(B, M) · LEND(M)
DISPLAY(B, T, M)	m_5	$true$	PICK(B) · SHOW(B, M) · **SAY**(T)
TAKEPAYMENT(M)	m_6	$nLnt(M, N)$	**DEBITACC**$(M, N * c)$
	m_7	$nLnt(M, N) \wedge cr(M, C) \wedge (C < N * c)$	PLACEPOSM() · PUTAWAYPOSM(M) · **EMAIL**(M)
PLACEPOSM()	m_8	$reachable(posm, pr2)$	PICK$(posm)$ · **SAY**$(swipe)$ · PUTON$(posm, stnd)$
	m_9	$true$	NAV$(posm)$ · PICK$(posm)$ · NAV$(desk)$ · **SAY**$(swipe)$ · PUTON$(posm, stnd)$
PUTAWAYPOSM(M)	m_{10}	$true$	**SAY**$(thank)$ · PICK$(posm)$ · PUTAWAY$(posm, M)$
SAY(T)		$true$	$\{spoke(T)\}$
MAKEBKACC(B, M)		$held(B, M) \wedge makeAcc(B, M)^?$	$\{\neg held(B, M),\ lent(B, M)\},\ makeAcc(B, M)^-,\ makeAcc(B, M)^+$
WAITTAKE(B, M)		$held(B, M) \wedge gave(B, M)$	$\{\neg held(B, M), \neg gave(B, M), lent(B, M)\}$
ADD(B, M)		$lent(B, M) \wedge nLnt(M, N)$	$\{\neg nLnt(M, N),\ nLnt(M, N + 1)\}$
GIVEBK(B, M)		$held(B, M) \wedge give(B, M)^? \wedge \neg hvy(B)$	$\{gave(B, M)\},\ give(B, M)^-,\ give(B, M)^+$
PICK(O)		$pick(O)^?$	$pick(O)^-,\ pick(O)^+$
DEBITACC$(M, Cost)$		$cr(M, C) \wedge (C \geq Cost)$	$\{\neg cr(M, C),\ cr(M, C - Cost)\}$
EMAIL(M)		$lent(B, M)$	$\{emailed(M)\}$

TABLE I: The table for Figure 1. The top half (above the thick line) are methods (M) and the bottom half are operators. For legibility, empty sets have been omitted from the table. Omitted operators are defined similarly to PICK(O).

credit. The ID corresponds to information including the member's name, books reserved, borrowed, and remaining credit. Reserved books need to be collected in person from the library. After an online reservation, (human) librarians find the books and make them accessible to the PR2 on a shelf adjacent to it; basically, an object is accessible to an agent if it is both *visible*, and *reachable* without needing to navigate from the current position.

The HTN hierarchy is shown in Figure 1 and detailed in Table I. The top-level HTN task MANAGEORDER(M) for member M has one method (m_1) with two subtasks: LEND(M) and TAKEPAYMENT(M). The first task has methods m_2, m_3, and m_4 and these are "ordered," i.e. tried in that order. Method m_2 trivially succeeds if there are no (more) books held by the member. If it is not applicable, m_3 is tried, which has the following actions: pick from the adjacent shelf a book reserved (held) by the member, speak out the title, make it accessible to him/her on the desk, do some bookkeeping—e.g. send the current total to the Point-of-Sale (POS) machine—and then recursively call LEND(M).

Method m_4 starts with an HTN abstract task to display a book, which refines into the three steps related to showing a book to the person. The abstract task is followed by: giving the book to the person;[2] waiting for it to be taken, which is a non-GS action that uses the gripper angle and force sensors to check if an object is in the gripper; the bookkeeping action; and finally a recursive call to LEND(M). Note that a book is given only if it is deemed light enough to be directly taken from the gripper, and that by forcing an ordering between m_3 and m_4 we are encoding a preference for placing a book on the table over handing it (directly) to the member, allowing the member to pick up the book and put it in a bag at his/her own pace. Note also that if a book is heavy ($hvy(B)$) it could be made accessible rather than given directly.

The TAKEPAYMENT(M) task of m_1 is associated with methods m_6 and m_7. Method m_6 involves debiting the member according to the number of books lent $nLnt$ (all

[2]We could also imagine more generic GIVEBK(B, M) and MAKEBKACC(B, M) actions that can handle any object type or give and make books accessible for reasons other than lending.

books have the same cost c), provided there is enough credit C in the member's account. If not, the member must pay by debit/credit card (m_7). The PLACEPOSM() task is associated with two ordered methods for giving the POS machine ($posm$): if it is reachable, it is picked up and put onto the stand ($stnd$) after asking the user to swipe the card and enter the PIN, but if not—presumably because it is with another receptionist—the PR2 must navigate to pick it up and bring it back. The PUTAWAYPOSM(M) task includes actions to thank the person and to move the machine away from the person's reach, and EMAIL(M) emails the member with details including the books that were lent.

The example serves to highlight some key characteristics. First, there is some interesting interleaving of HTN planning and geometric task planning that is possible with the domain described. Suppose that the library reception desk is small and somewhat cluttered with boxes, and that a member has reserved two big books b_1 and b_2. While there is enough space on the table to make one of them accessible (to the member), there is not enough space to make them both accessible, nor to make one accessible but give the other (directly), as the books are so big that they block the path to the member. Figure 2 illustrates how the combined planning process might work with such a setup. In the figure b_1 is successfully picked and made accessible (i.e. the GTP tasks corresponding to the PICK(b_1) and MAKEBKACC(b_1, m) GS actions are successfully planned), and LEND(M) is recursively called. However, because of our setup, the attempt to make book b_2 accessible will fail. In our current approach, at this point the GTP must backtrack to reconsider its choices—first for PICK(b_2), and if that fails, for MAKEBKACC(b_1, m), and if that also fails, for PICK(b_1). Since the GTP will still not find a way to change the pose of the first book so as to make the second accessible, the system will then resort to HTN backtracking, which will choose the alternative method m_4 to give b_2 directly to the person.[3] According to our initial setup, this will also fail even after the GTP tries to change the pose of b_1. The HTN planner will then backtrack once again (not shown), this time right to the beginning, and perhaps choose method m_4,[4] to give b_1 directly to the person; if this happens the planner will then successfully make b_2 accessible and continue planning.

The second characteristic that the domain depicted in Figure 1 highlights is "GTP influenced" HTN preconditions. In Table I, subsets of preconditions of the GIVEBK(B, M) and MAKEBKACC(B, M) operators are duplicated in the HTN methods where they occur. This is solely for efficiency: to ensure that the preceding GS actions (e.g. PICK(B)) in those methods, which are costly to plan, are not planned unnecessarily, i.e. if the preconditions of GIVEBK(B, M) and MAKEBKACC(B, M) are guaranteed to eventually not hold. So intuitively, for any GS action occurring in the method body, it makes sense to duplicate in the method-

[3]we assume here that the book is light enough to be handed to the person

[4]Method m_3 could also be chosen here to make book b_2 accessible.

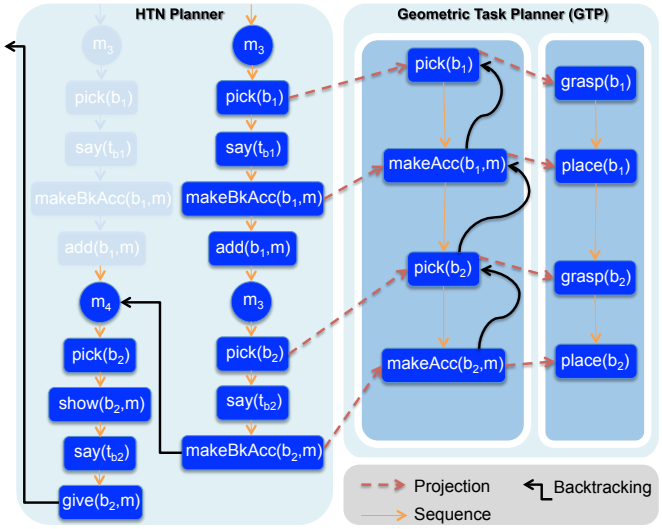


Fig. 2: Interleaved GTP and HTN backtracking and planning.

precondition the subset of the action's precondition[5] that does not "depend" on—that has no causal links [21] with—any preceding action in the body (e.g. the $held(B, M)$ literal in the precondition of MAKEBKACC(B, M) has no causal link with any of its preceding actions in the body). We could also order methods based on their estimated "computational cost." For example, by forcing method m_8 to be tried before m_9 we avoid unnecessarily planning to navigate—when the POS machine is likely reachable—as generally planning to navigate and pick up an object is more computationally expensive than planning to simply pick it up.[6] Interestingly, the notion of a "computationally costly" action here is different to the standard notion of "action cost," which is the estimated cost of executing the action.

Finally, note that the $reachable(O, A)$ predicate in the precondition of m_8 is a weaker notion than "real" reachability: when agent a can really reach object o, i.e. a collision-free trajectory exists from the arm of a to o, then $reachable(o, a)$ holds, but the converse is not necessarily true; the predicate is derived from the 3D world with a heuristic based on the area covered on extending the robot's arm with respect to all its degrees of freedom. Hence, the weak notion will clearly not hold if there is an obstacle blocking the path from the robot's arm to where the object is located. With such a "lazy predicate" we can quickly determine if an object is definitely unreachable and also if an object is *likely* to be reachable, e.g. before testing $putOn(posm, stnd)$? to see if it is really the case. In contrast, the $visible(O, A)$ shared predicate and others like $on(O, O_2)$ and $in(O, O_2)$ (for an agent A and objects O, O_2) only depend on the current geometric state—not on the existence of a trajectory—and thereby accurately

[5]excluding the evaluable predicate linked to a GTP task

[6]If there are abstract tasks in the method body, then the GS actions they might refine into will also need to be taken into account, and this process repeated until leaf-level methods (those that do not mention any abstract tasks in their bodies) are reached, as done for instance in [22] to "summarise" HTN hierarchies.

correspond to their "real" notions.

V. Algorithms for symbolic-geometric planning

We now present the Algorithms and data structures for an "intermediate layer" that coordinates HTN and GTP backtracking and planning, and a skeleton of the underlying GTP algorithm with certain features highlighted. We begin by illustrating some of the main ideas with an example. Consider a similar HTN domain to Figure 1 where the PR2 first makes reserved books accessible to the member (to show what he/she has borrowed), and then the POS machine if it is not already accessible, and finally gives the books (directly) to the member after asking him/her to insert the card and enter the PIN. To avoid any confusion when the user is asked to enter the PIN, the PR2, just before asking, hides from the member's view any other POS machines lying around on the table that are not involved in the transaction. Asking the member to enter the PIN is similar to the (non-GS) SAY$(swipe)$ action from Figure 1 except for an additional requirement to have only one POS machine visible to the member—all others must be hidden.

Now suppose that initially two POS machines $posm_1, posm_2$ were accessible to a member who had reserved two books b_1, b_2. Suppose also that: *(i)* MAKEBKACC(b_1, m) was successful, but with the side effect where $posm_2$, the one not used for the transaction, became hidden from the member; *(ii)* MAKEBKACC(b_2, m) was successful without leading to anything being hidden from the member; and finally, *(iii)* SAY$(swipe)$ was successful because literals $visible(posm_1, m)$ and $\neg visible(posm_2, m)$ were both true. Now, if for some reason GTP backtracking is triggered at some later point and the GTP picks different solutions for MAKEACC(b_2, m) and/or MAKEACC(b_1, m), the GTP must ensure that those two literals will still hold: it must "protect" them when backtracking. Not doing this may render the current HTN plan invalid. So if we use sequence $\vec{L}$ to keep track of the shared literals to protect per action in the HTN partial plan, at this point in our example we have $\vec{L} = \emptyset \cdot \emptyset \cdot \{visible(posm_1, m), \neg visible(posm_2, m)\}$. Consequently, the final state resulting from consecutively applying any new solutions for the first two actions' GTP tasks must be consistent with the literals in the last set in $\vec{L}$.

In the algorithms we use $Res(\sigma, s)$ and $Res(\vec{\sigma}, s)$ to denote, like in classical planning, the GTP world state resulting from applying respectively the GTP solution σ and the sequence of GTP solutions $\vec{\sigma}$ to the GTP world state s.[7] We also keep the notion of a GTP (world) state and GTP solution abstract. Basically, a GTP state is a 3D world model with positions and configurations of all objects, and a solution σ is sometimes simply a trajectory, but in general includes additional domain-specific information like the effort required to perform the task and symbolic information like the relative location of a manipulated object. We refer the reader to [12] for more details.

[7]In practice, we do not explicitly compute $Res(\sigma, s)$ as the GTP stores the end state of a solution found.

Algorithm 1 Evaluate Symbolic Predicate (GTP Task)

```
1: function TESTEVALPRED(t, len)
2:     Let t⃗ = t_1 · ... · t_n and L⃗ = L_1 · ... · L_{n+1}
3:     t_cur ← t
4:     if len < |t⃗| then        // backtracked symbolically
5:         t⃗ ← (len > 0 ? t_1 · ... · t_len : ϵ)
6:         L⃗ ← (len > 0 ? L_1 · ... · L_len · ∅ : ∅)
7:     ans ← PLANTASK(t⃗ · t, L⃗)
8:     return (ans = fail ? false : true)
```

Before planning starts, the HTN initial state must be made consistent with that of the GTP initial state. This is done using function GETSYMFACTS(), which will compute ground literals from the current geometric state, e.g. which objects are visible and reachable to which agents [19], [17], and which objects are on top of which other objects [23]. Moreover, before planning starts the following global variables are initialised: the sequence $\vec{t}$ (initially ϵ) of GTP tasks successfully planned so far; the sequence $\vec{L}$ (initially $\emptyset$) of sets of literals to protect; the current GTP task t_{cur} (initially ϵ) for which a GTP solution is being sought; its corresponding GTP solution $\vec{\sigma}_{cur}$ (initially ϵ); solution sequence $\vec{\sigma}$ (initially ϵ); and initial GTP world state s_0, where the last three variables are only visible to the GTP.

Given a sequence of GTP tasks $\vec{t}$ and a GTP solution sequence $\vec{\sigma}$ for it from world state s,[8] we say that $\vec{\sigma}$ ***protects*** a set of ground literals L from s, denoted $Pr(\vec{\sigma}, L, s)$, if all $l \in L$ hold in $Res(\vec{\sigma}, s)$. As illustrated in the previous example, we maintain the sequence of sets of ground literals $\vec{L} = L_1 \cdot \ldots \cdot L_n$ that need to be protected by new GTP solutions for the currently pursued sequence $\vec{t}$ of GTP tasks, and then try to ensure, given a prefix $t_1 \cdot \ldots \cdot t_i$ of $\vec{t}$, that some solution sequence $\sigma_1 \cdot \ldots \cdot \sigma_i$ for $t_1 \cdot \ldots \cdot t_i$ protects $L_{i+1} \in \vec{L}$ from the initial GTP state.

Algorithm 1 is the function called to evaluate predicates associated with GTP tasks. Indeed, for efficiency, this function should be called last while evaluating preconditions to ensure it is not called unnecessarily, i.e. when the rest of the precondition does not hold. In this algorithm, to detect if backtracking has occurred at the HTN level that warrants reconsidering certain GTP tasks in the postfix of $\vec{t}$, the length of the currently pursued HTN plan—actually, its subsequence containing GS actions—is checked: if less than the length of the current sequence $\vec{t}$ of GTP tasks then the algorithm explicitly "backtracks" by trimming the sequence accordingly. To understand how this works, consider Figure 2. Suppose again that we have a member m who has reserved two books b_1, b_2, and that the HTN planner chose method m_3 (for b_1) followed by m_3 again, to solve task LEND(m). Now, if the precondition of MAKEBKACC(b_2, m) is not applicable, the HTN planner must "discard" the second method m_3, and then continue perhaps by checking the precondition of PICK(b_2) of task DISPLAY(B, T, M). So the *length* we are interested in is two, as the two GS actions from the initially

[8]that is, $\sigma_1 \in \vec{\sigma}$ is the solution for $t_1 \in \vec{t}$ from s; $\sigma_2 \in \vec{\sigma}$ for $t_2 \in \vec{t}$ from $s_1 = Res(\sigma_1, s)$; $\sigma_3 \in \vec{\sigma}$ for $t_3 \in \vec{t}$ from $Res(\sigma_2, s_1)$; and so on

Algorithm 2 Protect Ground Literals and Apply GTP Task

1: **function** PROTECTANDAPPLY(L)
2: Let $L' \subseteq L$ be the set of shared literals in L
3: $\vec{\boldsymbol{L}} \leftarrow L_1 \cdot \ldots \cdot L_{|\vec{t}|} \cdot (L_{|\vec{t}|+1} \cup L')$, where $\vec{\boldsymbol{L}} = L_1 \cdot \ldots \cdot L_n$
4: **if** $\boldsymbol{t}^{?}_{\boldsymbol{cur}} \in L$ **then** // applying effects of a GS action
5: $\vec{\boldsymbol{t}} \leftarrow \vec{\boldsymbol{t}} \cdot \boldsymbol{t_{cur}}$
6: $\vec{\boldsymbol{L}} \leftarrow \vec{\boldsymbol{L}} \cdot \emptyset$
7: APPLYLASTTASK()
8: **return** GETSYMFACTS()
9: **return** $\emptyset$

Algorithm 3 Geometric Task Planning (Top-Level)

1: **function** PLANTASK($\vec{t}, \vec{L}$)
2: Let $\vec{\boldsymbol{\sigma}} = \sigma_1 \cdot \ldots \cdot \sigma_n$
3: $\vec{\boldsymbol{\sigma}} \leftarrow (|\vec{t}| > 1 \; \textbf{?} \; \sigma_1 \cdot \ldots \cdot \sigma_{|\vec{t}|-1} \; \textbf{:} \; \epsilon)$
4: $\vec{\boldsymbol{\sigma}}_{\boldsymbol{cur}} \leftarrow$ PTR($\boldsymbol{s_0}, \vec{t}, \vec{\boldsymbol{\sigma}}, \vec{L}$)
5: **return** $(\vec{\boldsymbol{\sigma}}_{\boldsymbol{cur}} = \mathit{fail} \; \textbf{?} \; \mathit{false} \; \textbf{:} \; \mathit{true})$

selected method m_3 are still in the partial plan.[9]

After line 7 we assume a cache is kept of previous solutions for PLANTASK($\vec{\boldsymbol{t}} \cdot t, \vec{\boldsymbol{L}}$), to avoid recomputing a plan for the same input on repeated calls to Algorithm 1. This is possible, for example, if the HTN planner needs to check multiple ground instances of a precondition before finding one that holds in the current state.

The HTN planner calls Algorithm 2 when an action's effects need to be applied to the state of the world. The input is the set of ground literals that was checked in the precondition, and it returns the set of shared (ground) literals (both positive and negative) resulting from executing GTP task $\boldsymbol{t_{cur}}$.

Algorithms 3 and 4 (PTR is short for PLANTASKRECURSE) depict the basic geometric task planning algorithm. They illustrate how the GTP backtracks on choices and how literals are protected during geometric planning. Indeed, such a basic depth-first search algorithm will need to be supported by efficient heuristics to be practical: some promising steps in this direction are proposed in [2].

Algorithm 3 simply truncates solution sequence $\vec{\boldsymbol{\sigma}}$ if its corresponding sequence of GTP tasks $\vec{t}$ was already truncated in Algorithm 1, and then calls Algorithm 4. Algorithm 4 is basically backward state space search from the new GTP task that needs to be planned to the last GTP task that was already planned, and eventually to the first GTP task that was already planned. Lines 5 and 6 try to keep the largest possible prefix of the previously planned GTP solution sequence $\vec{\sigma}$. Line 8 ensures that the relevant literals are protected when looking for a new solution for t_1.

VI. IMPLEMENTATION AND EFFICIENCY ISSUES

For the integration we use the HATP [24] HTN planner and our GTP [12], which have both been used extensively (albeit separately) in the LAAS architecture [25] for HRI experiments. We will start with a brief overview of the LAAS architecture. The PR2 uses the Move3D [26] integrated planning and visualisation platform for representing the real

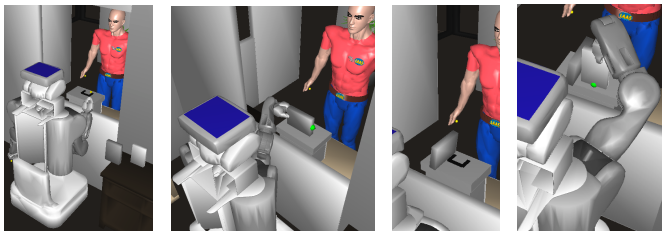
[9]This length could be maintained straightforwardly in the HTN domain without any modifications to the planner, and included as an extra parameter in the evaluable predicate.

Algorithm 4 Geometric Task Planning

1: **function** PTR(s, $\vec{t}, \vec{\sigma}, \vec{L}$)
2: Let $\vec{t} = t_1 \cdot \ldots \cdot t_n$ // t_n is the new GTP task
3: Let $\vec{L} = L_1 \cdot \ldots \cdot L_n$ and $\vec{\sigma} = \sigma_1 \cdot \ldots \cdot \sigma_{n-1}$
4: **if** $|\vec{t}| = 0$ **then return** ϵ
5: **if** $|\vec{\sigma}| > 0$ **then** // try stored solution
6: $\vec{\sigma}_{sol} \leftarrow$ PTR($Res(\sigma_1, s), t_2 \cdot \ldots \cdot t_n, \sigma_2 \cdot \ldots \cdot \sigma_{n-1}, L_2 \cdot \ldots \cdot L_n$)
7: **if** $\vec{\sigma}_{sol} \neq \mathit{fail}$ **then return** $\sigma_1 \cdot \vec{\sigma}_{sol}$
8: **if** $\nexists \sigma$ for t_1 from s such that $Pr(\sigma, L_2, s)$ **then return** *fail*
9: Nondeterministically choose such a σ
10: $\vec{\sigma}_{sol} \leftarrow$ PTR($Res(\sigma, s), t_2 \cdot \ldots \cdot t_n, \epsilon, L_2 \cdot \ldots \cdot L_n$)
11: **if** $\vec{\sigma}_{sol} \neq \mathit{fail}$ **then return** $\sigma \cdot \vec{\sigma}_{sol}$
12: **return** *fail*

world in 3D and for doing geometric task planning with it. Through various sensors the PR2 can also update the 3D world state in real-time. To this end, a tag-based stereovision system is used for object identification and localisation, and a Kinect (Microsoft) sensor for localising and tracking the human. With our geometric tools we not only anchor a symbolic fact, but also compute the cost (in terms of effort) associated with the fact [19], [17].

Although we keep the symbolic-geometric planning separate from execution, we have implemented the mechanisms to execute on the real PR2 platform the plans produced. We have also implemented the HTN domain and all GTP tasks in Figure 1, except for NAV(Obj) which requires further work, and sufficiently implemented the algorithms presented in this paper to gain some preliminary insights.



(a) (b) (c) (d)

Grey is Vis with E=1 & Reach with E=4
White is Vis with E=4 & Reach with E=4
(e)

Grey is Vis with E=1 & Reach with E=1
White is Vis with E=4 & Reach with E=4
(f)

Grey is Vis with E=1 & Reach with E=1
White is Vis with E=1 & Reach with E=2
(g)

Fig. 3: Screenshots showing backtracking and facts produced.

In our GTP domain the GETSYMFACTS() function is able to compute predicates such as $visible(O, A)$, $reachable(O, A)$, $on(O, O_2)$, and $inside(O, O_2)$ for an agent A and objects O, O_2. The first two are computed using techniques in [17], and the others using domain-specific heuristics. We found that in practice, computing visibility and reachability in a standard domain with 3 agents and 6 objects takes approximately 0.2 seconds on average, which is negligible compared to the time it takes to do geometric backtracking [2]. We refer the reader to [20] for some preliminary results on the runtime performance of our symbolic-geometric planning system.

In Figure 3, (a) to (d) show in Move3D a part of the backtracking depicted in Figure 2, and (e) to (g) summarises the $visible(O, A)$ and $reachable(O, A)$ shared literals involving the human, for each of the Move3D states, with (f) corresponding to both (b) and (c). Sub-figure (a) shows a grey and white book on the shelf adjacent to the PR2, and a person ("HUMAN") at the reception counter; (b) shows the PR2 making the grey book ("Grey") accessible on a box on the desk; (c) shows how it is later moved, after backtracking, to make room for the white book ("White"); and (d) shows the PR2 placing the white book on the box. Sub-figure (e) shows that the grey book is (easily) visible ("Vis") to the human, i.e. with an effort ("E") of either 0 or 1, but that it cannot be (easily) reached ("Reach") by him—that requires an effort greater than 1. Columns (f) and (g) are self explanatory. Although not shown in the figure, both books were always visible and reachable to the PR2 with E=1.

VII. Discussion and conclusion

In this paper, we have investigated issues related to linking geometric reasoning with symbolic planning. To this end, we focussed on defining a clear interface between the two planners, in particular, defining how geometric entities, like compound tasks and 3D world states, can be meaningfully included in HTN planning. This led to a first attempt at a methodology for developing symbolic-geometric planning domains. We showed with a non-trivial example how carefully written HTN domains coupled with "lazy" conditions could improve performance. We discussed an approach to interleaved backtracking and provided algorithms for it, together with algorithms that link the two planners. These included a way to "protect" shared literals, which may otherwise be negated while backtracking in the GTP, possibly making the pursued HTN plan invalid. Finally, we discussed a prototype that implements some of the useful bits of the combined system, and presented a basic backtracking scenario along with the shared literals computed at each step.

We note that while our aim is to preserve completeness as much as possible, and evaluable predicates corresponding to GTP tasks trigger GTP backtracking to this end, a shared literal (e.g. $visible(b_1, h_1)$) mentioned in an HTN precondition will not trigger GTP backtracking, resulting in the possible loss of solutions: the literal might just not hold because of a particular GTP solution choice made earlier (e.g. the pose of book b_1 with respect to human h_1). To address this, we are working on an alternative combined planning strategy for domains where completeness is important. Intuitively, in our new approach the HTN planner backtracks to ask the GTP to find alternative solutions for relevant GTP tasks. To this end, certain GS operator instances essentially map to different GTP solutions for the operators' corresponding GTP tasks.

References

[1] L. Karlsson, J. Bidot, F. Lagriffoul, A. Saffiotti, U. Hillenbrand, and F. Schmidt, "Combining task and path planning for a humanoid two-arm robotic system," in *ICAPS Workshop on Combining Task and Motion Planning for Real-World Applications*, 2012, pp. 13–20.

[2] F. Lagriffoul, D. Dimitrov, A. Saffiotti, and L. Karlsson, "Constraint propagation on interval bounds for dealing with geometric backtracking," in *IROS*, 2012, pp. 957–964.

[3] D. Nau, H. Muñoz Avila, Y. Cao, A. Lotem, and S. Mitchell, "Total-order planning with partially ordered subtasks," in *IJCAI*, 2001, pp. 425–430.

[4] C. Dornhege, M. Gissler, M. Teschner, and B. Nebel, "Integrating symbolic and geometric planning for mobile manipulation," in *IEEE International Workshop on Safety, Security and Rescue Robotics*, 2009.

[5] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, "Semantic attachments for domain-independent planning systems," in *ICAPS*, 2009, pp. 114–121.

[6] C. Dornhege, P. Eyerich, T. Keller, M. Brenner, and B. Nebel, "Integrating task and motion planning using semantic attachments," in *Bridging the Gap Between Task and Motion Planning*, 2010.

[7] J. Wolfe, B. Marthi, and S. J. Russell, "Combined task and motion planning for mobile manipulation," in *ICAPS*, 2010, pp. 254–258.

[8] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *ICRA*, 2011, pp. 1470–1477.

[9] ——, "Unifying perception, estimation and action for mobile manipulation via belief space planning," in *ICRA*, 2012, pp. 2952–2959.

[10] S. Cambon, F. Gravot, and R. Alami, "A robot task planner that merges symbolic and geometric reasoning," in *ECAI*, 2004, pp. 895–899.

[11] E. Plaku and G. Hager, "Sampling-based motion and symbolic action planning with geometric and differential constraints," in *ICRA*, 2010, pp. 5002–5008.

[12] A. K. Pandey, J.-P. Saut, D. Sidobre, and R. Alami, "Towards planning human-robot interactive manipulation tasks: Task dependent and human oriented autonomous selection of grasp and placement," in *IEEE RAS/EMBS BioRob*, 2012, pp. 1371–1376.

[13] K. Wyrobek, E. Berger, H. Van der Loos, and J. Salisbury, "Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot," in *ICRA*, 2008, pp. 2165–2170.

[14] R. Fikes and N. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[15] D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila, "SHOP: Simple hierarchical ordered planner," in *IJCAI*, 1999, pp. 968–973.

[16] M. Gharbi, J. Cortes, and T. Simeon, "A sampling-based path planner for dual-arm manipulation," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2008, pp. 383–388.

[17] A. K. Pandey and R. Alami, "Mightability maps: A perceptual level decisional framework for co-operative and competitive human-robot interaction," in *IROS*, 2010, pp. 5842–5848.

[18] S. Coradeschi and A. Saffiotti, "An introduction to the anchoring problem," *Robotics and Autonomous Systems*, vol. 43, pp. 85 – 96, 2003.

[19] A. K. Pandey and R. Alami, "Visuo-spatial ability, effort and affordance analyses: Towards practical realization of building blocks for robots complex socio-cognitive behaviors," in *AAAI Workshop on Cognitive Robotics*, 2012.

[20] L. de Silva, A. K. Pandey, M. Gharbi, and R. Alami, "Towards combining HTN planning and geometric task planning," in *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*, 2013.

[21] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Elsevier, 2004.

[22] B. J. Clement and E. H. Durfee, "Theory for coordinating concurrent hierarchical planning agents using summary information," in *AAAI*, 1999, pp. 495–502.

[23] S. Lemaignan, R. Alami, A. K. Pandey, M. Warnier, and J. Guitton, *Bridges between the Methodological and Practical Work of the Robotics and Cognitive Systems Communities - From Sensors to Concepts*. Springer Publishing, 2012, ch. Towards Grounding Human-Robot Interaction.

[24] S. Alili, R. Alami, and V. Montreuil, "A task planner for an autonomous social robot," in *Distributed Autonomous Robotic Systems*, 2009, pp. 335–344.

[25] S. Fleury, M. Herrb, and R. Chatila, "Genom: A tool for the specification and the implementation of operating modules in a distributed robot architecture," in *IROS*, 1997, pp. 842–848.

[26] T. Simeon, J.-P. Laumond, and F. Lamiraux, "Move3d: a generic platform for path planning," in *4th Int. Symp. on Assembly and Task Planning*, 2001, pp. 25–30.