Conditional Task and Motion Planning through an Effort-Based Approach

Nicola Castaman\textsuperscript{1}, Elisa Tosello\textsuperscript{2}, and Enrico Pagello\textsuperscript{3}

Abstract—This paper proposes a preliminary work on a Conditional Task and Motion Planning algorithm able to find a plan that minimizes robot efforts while solving assigned tasks. Unlike most of the existing approaches that replan a path only when it becomes unfeasible (e.g., no collision-free paths exist), the proposed algorithm takes into consideration a replanning procedure whenever an effort-saving is possible. The effort is here considered as the execution time, but it is extensible to the robot energy consumption. The computed plan is both conditional and dynamically adaptable to the unexpected environmental changes. Based on the theoretical analysis of the algorithm, authors expect their proposal to be complete and scalable. In progress experiments aim to prove this investigation.

I. INTRODUCTION

Let a human assign a task to a robot. e.g., the pick of a can of coke from a cluttered table or the resolution of a Navigation Among Movable Obstacles (NAMO) [1], [2] problem (see Figure 1). In order to achieve these targets, the robot needs to fulfill high-level task planning in conjunction with low-level motion planning. As stated in [3], efficient algorithms exist to solve task and motion planning problems in isolation; however, their integration is still challenging in terms of generality, completeness, and scalability.

Authors provide a Task and Motion Planning (TAMP) system which combines a Fast-Forward (FF) task planner [4] and a revisited version of the Lazy Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (L-KPIECE) motion planner [5], as proposed in [6], [7]. The goal is to give the possibility to any type of robot to use this system for solving any type of task, from manipulation to navigation, in an unknown, real-world scenario. The implementation aims to reflect the human behavior: humans take plans while efficiently managing their time and energy. In detail, given the actions that the robot can perform, together with its initial and final states, one feasible task plan, namely a reachability graph, is computed by using FF (no optimality check is required). The corresponding motion plan is generated by using [7]. Based on the knowledge of the robot world at its current state, a collision checking is performed so that the final plan would take into consideration only those objects and sub-tasks that minimize robot efforts (the execution time in this case, but the energy used by the robot can also be considered). The robot starts moving. While acting, if failures or environment changes occur (e.g., objects are detected which obstruct the computed path), an online replanning routine is adopted. Starting from the robot current state, it regenerates the remaining task plan by evaluating and substituting those actions which preconditions are no longer feasible. Starting from the new task plan, the algorithm proposed in [7] is invoked to find a new feasible set of motions.

Completeness. Acting in the real world is non-deterministic: the planner has no complete and certain knowledge of the environment and actions can have unpredictable effects. This means that a robot should be able to dynamically adapt to changes and efficiently recover from failures while avoiding the explosion of the computed reachability graph because of the new alternatives introduced. In the case of analysis, starting from a relax task, i.e., a plan which ignores the delete lists of operators, FF extracts an explicit solution to the plan by using a search strategy called enforced hill-climbing (EHC). This strategy does not use backtracking, meaning that some parts of the search space are lost, but if FF fails to find a solution using EHC by getting stuck in dead-ends, it switches to standard best-first searches (e.g., greedy or A* search) which expands all search nodes by increasing order of goal distance evaluation. This recovery routine guarantees completeness. As stated in [5], KPIECE is, instead, probabilistically complete. Moreover, the proposed planner handles known and unknown events through the online replanning routine until a maximum

---

\textsuperscript{1}Nicola Castaman is with IAS-Lab, Dept. of Information Engineering, University of Padova, Padova, Italy and IT+Robotics srl, Vicenza, Italy castaman@dei.unipd.it
\textsuperscript{2}Elisa Tosello is with IAS-Lab, Dept. of Information Engineering, University of Padova, Padova, Italy tosello@dei.unipd.it
\textsuperscript{3}Enrico Pagello is with IAS-Lab, Dept. of Information Engineering, University of Padova, Padova, Italy and IT+Robotics srl, Vicenza, Italy epv@dei.unipd.it

Fig. 1: A Navigation Among Movable Obstacles problem: while navigating towards a goal, the robot takes into consideration the possibility of moving movable obstacles.
number of attempts is reached. This means that if the number of attempts tends to infinity, the probability of finding a plan, if one exists, will tend to one.

Scalability. The task planning problem is exponential in the number of sub-tasks and the motion planning problem is exponential in the number of collision objects populating the workspace. Indeed, multiple sub-tasks exist fulfilling the assigned task and different combinations of sub-tasks can bring to the same result. Moreover, when operating in the clutter, a robot has to decide which objects to move, where to place them, if moving them is necessary or convenient. Focusing on the task-level planning, FF employs an helpful actions pruning that maintains in the relaxed plan only those actions that are really useful in a state, so one can restrict the successors of any state to only those produced by members of the respective relaxed solution. Focusing on the motion-level planning, [7] is based on KPIECE because KPIECE uses the information collected during the planning process in order to decrease the amount of forward propagation it needs [5]. Moreover, by using the Lazy strategy, edges of the computed path are not immediately checked for collision: they are checked only when a better path to the goal is found. By eliminating the majority of collision checks and efficiently forward propagating the exploration during the planning, the proposed algorithm speeds up to convergence in complex problems, like that of TAMP, where collision checking is relatively expensive. It also requires restrained runtime and memory so that it can handle high dimensional systems with complex dynamics.

II. RELATED WORK

Various researchers investigated the problem of combining task and motion planning. Most of them tried to combine the symbolic reasoning of task planning with the geometric reasoning of motion planning. Dornhege et al. [8], [9], for instance, proposed a system that calls the motion planner only to check the geometric feasibility of the planned tasks. Another example is the FFRob approach proposed by Garrett [10], [11]: it integrates geometric information with the state-of-the-art Fast-Forward (FF) [4] task planner by sampling a fixed set of object poses and robot configurations and then planning with them by using FF. Benefits of FFRob are its probabilistic completeness and exponentially convergence. Its major limit is its focus: the approach solves only a particular class of pick-and-place problems. Moreover, it does not use planning to guide sampling.

Other approaches such as [12], [13], [14] still focus on manipulation tasks but they propose solutions based on Hierarchical Planning that evaluate task-level decisions by using low-level geometric-reasoning modules. In particular, Srivastava [12] combines off-the-shelf task planners with an optimization-based motion planner [15], [16] that exploits a heuristic function to remove potentially-interfering objects. This approach first plans a task and then tries to produce a motion plan that satisfies the computed set of discrete actions. If the induced motion planning problem is infeasible, the task planning is repeated taking into consideration the set of preconditions that identifies the infeasibility.

Finally, Dantam et al. [3] propose an incremental task and motion planner which combines discrete decisions about objects and actions with geometric decisions about collision free motion: they use an incremental constraint solver that adds motion constraints to the computed candidate task plan. The task plan is computed by using a Satisfiability Modulo Theories (SMT) approach, while the Open Motion Planning Library (OMPL) is used to find a feasible motion plan. At each failure, the algorithm iteratively increases the plan depth and motion planning timeouts such that it guarantees probabilistically completeness for fixed placements and grasps.

Instead of only focusing on manipulation domains, the proposed approach aims to integrate task and motion planning routines while performing generic tasks, that means robots should be able to use this system in order to handle both navigation and manipulation domains. Moreover, while most of the state-of-the-art approaches evaluate the objects relocation only when free-space motion planning is unfeasible, the proposed algorithm revalues a plan every time an action can save effort, not only when the ongoing trajectory becomes unfeasible.

III. PROBLEM STATEMENT

This Section defines the Deterministic Task (III-A) and Motion Planning (III-B) problems. These concepts will be the definition basis of the Conditional Task and Motion Planning authors will introduce in III-C.

A. Deterministic Task Planning

Suppose the assignment of a task $T$ to a robot $R$. A task planner $TP : (s_0,s_G,A) \rightarrow p^*$ aims to find an optimal plan $p^* \in P$ solving $T$. $p^*$ moves $R$ from its start state $s_0 \in S$ to a goal state $s_G \in S$ by combining the set of actions $A$ that $R$ is able to perform according to its capabilities.

The problem is deterministic if the actions domain is fully observable and every action $a \in A$ is fully defined as a sentence in the Planning Domain Definition Language (PDDL) [16] with a set $\text{precon}(a) = \{\text{precon}_0(a),...\text{precon}_N(a)\}$ of preconditions and a set of effects $\text{effect}(a) = \{\text{effect}_0(a),...\text{effect}_N(a)\}$ of effects, described as conjunctive lists of literals in first-order logic.

$TP$ computes a set of plans $P$, where $p \in P$ is defined as

$$p = \langle s_0,a_0,...,s_N-1,a_N-1,s_N \rangle, \quad s_N = s_G$$

and $(s_i,a_i) \rightarrow s_{i+1}$ iff $\text{precon}(a_i)$ is satisfied by $s_i$ and $\text{effect}(a_i)$ brings to $s_{i+1}$.

A plan $p^* \in P$ is optimal if it has the lowest cost among all the computed plans:

$$p^* = \arg\min_{p \in P} \sum_{\langle s,a \rangle \in p} \text{Cost}(\langle s,a \rangle)$$

$\text{Cost}(\langle s,a \rangle)$ is the cost of action $a$ being executed in state $s$. 
B. Deterministic Motion Planning

A motion planner $MP: (s_0, s_G, A) \rightarrow t^*$ tries to find an optimal path $t^* \in \tau$ that lets $R$ move from $s_0$ to $s_G$ in $S$ while avoiding collisions. The problem is deterministic if the working space is fully observable. In this case, $MP$ can find a set of paths $\tau$, where $t \in \tau$ is a path in the free space:

$$\tau: [0, 1] \rightarrow C_{\text{free}}, \quad \tau(0) = s_0, \quad \tau(1) = s_G$$

$t^*$ is optimal if its trajectory is of minimum length:

$$t^* = \arg\min_{t \in \tau} \{\text{Length}(t)\}$$

C. Conditional Task and Motion Planning

Suppose the existence of a Task and a Motion Planner (see III-A and III-B). Suppose that $T$ is assigned to $R$. TMP: $(s_0, s_G, A) \rightarrow t^*$ finds the optimal plan $p^* \in P$ performing $T$ and returns the optimal trajectory $t^* \in \tau$ executing $p^*$. The solution is optimal if $t^*$ is of minimum cost:

$$t^* = \arg\min_{p \in P} \left( \sum_{0 \leq i \leq |p|} \text{Cost}(t_i|a_i) \right)$$

where $\text{Cost}(t_i|a_i)$ is the cost of the trajectory necessary to perform the i-th planned action. Without loss of generality, in this paper the solution is optimal if it minimizes the robot’s effort: $\text{Effort}(t_i|a_i)$ and $\text{Effort}(t|p) = \sum_{0 \leq i \leq |p|} \text{Effort}(t_i|a_i)$. The effort is defined as the execution time.

The problem is deterministic if the actions space is a priori fully defined and each action is executed infallibly. However, in the real world actions can generate unexpected events and the robot can perceives changes at its surroundings. This means that the outcomes of environment and actuation actions should be processed in order to address uncertainties due to partial observability at the time of offline planning [17]. The definition of $t^*$ is unchanged but the way used to find it is new: the plan $p^*$ should handle every known condition through the definition of a reachability graph and an online recovery procedure should handle unexpected events by combining sensing and actuation actions and minimizing the global cost of the computed path.

IV. ALGORITHM

A task $T$ is assigned to a robot $R$. $R$ asks $R$ to reach a goal state $s_G$. The complete set of actions $A$ that $R$ can perform and its initial state $s_0$ are known. Actions are expressed in PDDL. Starting from this information, Algorithm I is applied.

Let: - $P$ be the set of plans $\{p_0, ..., p_N\} \in P$ found by recursively applying the TP (FF) when an online replanning is necessary; - $Obstacles$ be the set of objects encountered during the robot motion; - $G$ be the reachability graph having as edges the actions of $P$ and as nodes the states of $P$. More in detail, every edge models the motion path used to execute that action. Initially, all these sets are empty.

Algorithm I starts by applying the TP (FF) and computing one sequence of possible actions letting $R$ accomplish $T$: $p = \langle s_0, a_0, ..., s_i, a_i, s_{i+1}, ..., s_G \rangle \in P$. The plan may not be optimal. Starting from $P$, Algorithm 2 initiates $G$ with the set of actions to be performed and the list of states that are consequently reached.

Given the sequence of actions to be performed, [7] (a variation of the Lazy KPIECE) is used to compute the motion path $t^*$ connecting, when possible, every couple of states of $G$ through a motion trajectory not checked for collisions. The feasibility of the connection is checked in terms of actions’ preconditions and effects. The aim is driving the robot towards the shortest path, in terms of Euclidean distance to the goal (see [7]). If the environment is unknown, $t^*$ is computed until the last visible state. If $p^* \in G$ expects the execution of a navigation action, $t^*$ is computed in terms of the mobile base reference system. If it expects a manipulation action, $t^*$ combines the motion of the base with that of the manipulator robot. Moreover, [18] is used to detect all possible 2-fingers grasps and a geometric variation of it should be used to detect 3-fingers grasps.

Starting from $t^*$, the sequence of control inputs letting $R$ perform the trajectory is computed. This sequence lets deduce the effort $c^*$, in terms of costs, needed by the robot to perform actions. In the case in analysis, the algorithm takes into consideration the execution time (see [7]).

Being the robot at its current state $s_t$ (i.e., $s_0$ at the very beginning), the graph expansion starts (see Algorithm 3).
Fig. 2: Algorithm Pipeline. The algorithm computes a candidate Task Plan (TP), explores the best branch, and computes the corresponding candidate Motion Plan (MP). It checks for collisions in the candidate Motion Plan. If a collision exists, it generates a new subtask to handle the collision object, it replans, and updates the reachability graph. If there is no collision, it updates the reachability graph with the MP. It repeats the pipeline until the goal state is reached.

Algorithm 3: Traverse($v$): Expand $v$ to find the best $t^*$

Input: $v$: the node to be expanded

while ($v$.hasChild()) && ! (MaxAttempts reached) do

1. $t_{lazy} \leftarrow$ LazyKPIECE($v$, child);
2. while $t_{lazy}$ has new collision $\in C_{movable}$ do
3. $obj \leftarrow$ findCollisionObject(collision);
4. if ($ob$,label,$ob$,pose) $\notin$ Obstacles then
5. $Obstacles.pushBack(obj)$;
6. $A_{obj} \leftarrow$ findPossibleActions(obj);
7. foreach $a \in A$ do
8. $S_{effect} \leftarrow$ findEffectStates($a$);
9. foreach $s \in S_{effect}$ do
10. if $\exists_{before} \leftarrow$ TP($v$, $s$, $A$) then
11. foreach child of $v$ do
12. if $\exists_{after} \leftarrow$ TP($s$, $child$, $A$) then
13. $G \leftarrow$ updateRGraph($G$, $p_{before}$);
14. $G \leftarrow$ updateRGraph($G$, $p_{after}$);
15. $t \leftarrow t + KPIECE(v$, child$)$;
16. cost $\leftarrow$ cost + Effort($v$, child$)$;
17. if cost $< c^*$ then
18. if $v == s_G$ then
19. $c^* \leftarrow$ cost;
20. $t^* \leftarrow t$;
21. return;
22. else
23. Traverse(child)
24. 

Given $t^* = t_{lazy}$, every node $s_i \in G$ and every edge $(s_i,s_{i+1}) \in G$ are checked for collisions. Authors remember that $G$ already contains the best plan performing the assigned task: the one which motion path has minimum Euclidean distance to the goal. Let the robot be at state $s_i$, for every new collision detected in the space of movable obstacles $C_{movable}$, a state $s_j \in S$ is sampled. Based again on the set of feasible actions $A$, their preconditions, and effects, $G$ is extended by adding the task plans from/to $s_j$ (it is a new iteration of FF).

The algorithm evaluates all the task plans $\langle s_1,a_1,\ldots,s_j,a_j,\ldots,s_{i+N},s_G \rangle$ that connect the robot current state to the current plan passing through $s_j$. The related motion paths are evaluated too. In detail, starting from the original path $\langle s_1,\ldots,s_G \rangle$, if the new path $\langle s_1,\ldots,s_j \rangle$ has a cost $c^*(s_1,s_j)$ less than $c^*(s_i,s_G)$, the algorithm continues the exploration of this branch until reaching $s_G$ or until exceeding $c^*(s_i,s_G)$. In this case, continuing this road is no longer convenient and the exploration must focus on other branches. The exploration proceed until a better solution is found, all children have been visited, or the maximum number of attempts has been reached. This means that $p$ is revalued not only when unfeasible, but every time better solutions exist with respect to the selected one. For example, once an obstacle is found and a path avoiding it is computed, the algorithm evaluates both the action of avoiding the object and the one of manipulating it. It then selects the alternative of minimum effort, that in the case in analysis means the one requiring the minimum execution time.

Figure 3 depicts the proposed algorithm pipeline, starting from the desired task assignment to the TAM solution. The online replanning routine is also depicted; its details follow.

A. On-Line Replanning

Suppose the robot $R$ is executing a planned trajectory $t^*$, coming from an action $a \in p^*$, and in the meanwhile it continuously perceives its surrounding. While acting in the real world, unpredictable faults could happen. E.g., the
Contemplated events may occur. E.g., the robot has to take an action B. Conditional Planning

The online replanning procedure exploits the same method described above to add a new sub-task to the reachability graph. Indeed, if the ongoing action fails, a new sub-task bypassing the faulty state is generated. The sub-task is computed by the task planner using the set of actions that the robot is able to perform and it is added to the reachability graph. This sub-task is naturally connected to the next non-faulty action node, so that the approach does not need to perform a total replanning from the actual state s_i to the goal state s_G. In the end, the Traverse(s_i) function is invoked in order to try to find a path that minimizes the execution time. A number of attempts is chosen a priori: if no plan is found and totally executed within those attempts, the system outputs a failure.

Figure 3 presents a possible scenario where a robotic hand has to pick up an object. The Task and Motion Planner elaborates a plan to achieve the assigned task in which the robot moves near the object and then picks it up, as in Figure 3a. During the execution of the Move action a new object that prevents the robot to accomplish the movement is perceived by the sensors (see Figure 3b). The online replanning generates a new sub-task corresponding to the obstacle replacement and adds it in the reachability graph, as in Figure 3c. The algorithm explores the graph and expands the less onerous branch that, in this example, corresponds to the elusion of the obstacle. As expected, the subsequent Pick action is exactly the same for both suitable branches.

B. Conditional Planning

Suppose R is executing a trajectory t^*, coming from an action a ∈ ρ^*, and meanwhile it is perceiving its surrounding. Contemplated events may occur. E.g., the robot has to manipulate an object but its gripper is not empty or the object is occluded. From the literature, these events are treated by Condition Planning routines. In the case in analysis, they are already handled by the planner through the construction of the reachability graph. No replanning procedure is involved, just the right sequence of actions is chosen (see Figure 4). This approach’s capability justifies the choice of authors to call the proposal as a Conditional approach.

V. Experiments

Figure 5 depicts the use cases authors are studying at the time of the submission. Figure 5a shows a mobile manipulator robot trying to solve a NAMO problem. Figure 5b shows the same robot trying to pick up an occluded can of coke from a cluttered table. The robot can perform four different actions:

- Move_{base}(pose_{start},pose_{goal}, traj);
effectiveness of the proposed approach. They aim to prove the adaptability and scalable.

Experiments aim to prove:
1) the adaptability of the algorithm when dealing with a perceived workspace;
2) the effectiveness of weighing paths based on the effort done.

Authors consider the effort as the time spent and they aim to prove that the obtained solution is the fastest one.

VI. CONCLUSION

Authors presented a new algorithm able to solve a Task and Motion Planning problem through an effort-based approach. The effort is the time spent to accomplish the task and the algorithm finds the plan that can be executed in the shortest possible time. The non-determinism of the real world is faced by providing a Conditional Planning and a recovery routine that handles unexpected events and new scene detections. The proposed algorithm is complete and scalable.

Authors expose some use cases whose implementation is still in progress. They aim to prove the adaptability and effectiveness of the proposed approach.

REFERENCES

[1] M. R. Dogar and S. S. Srinivasa, “A framework for push-grasping in clutter,” Robotics: Science and systems VII, vol. 1, 2011.
[2] M. Leibn, J. Scholz, and M. Stilman, “Hierarchical decision theoretic planning for navigation among movable obstacles,” in Algorithmic Foundations of Robotics X. Springer, 2013, pp. 19–35.
[3] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, “Incremental task and motion planning: a constraint-based approach,” in Robotics: Science and Systems. Proceedings of, 2016.
[4] J. Hoffmann and B. Nebel, “The FF planning system: Fast planning generation through heuristic search,” Journal of Artificial Intelligence Research, vol. 14, pp. 253–302, 2001.
[5] I. A. Șucan and L. E. Kavraki, “Kinodynamic motion planning by interior-exterior cell exploration,” in Algorithmic Foundation of Robotics VIII. Springer, 2009, pp. 449–464.
[6] N. Castaman, “A sampling-based tree planner for robot navigation among movable obstacles,” Master’s thesis, University of Padova, 2016.
[7] N. Castaman, E. Tosello, and E. Pagello, “A Sampling-Based Tree Planner for Navigation Among Movable Obstacles,” in ISR 2016: 47th International Symposium on Robotics; Proceedings of. VDE, 2016, pp. 292–299.
[8] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, “Semantic attachments for domain-independent planning systems,” Towards Service Robots for Everyday Environments, pp. 99–115, 2012.
[9] C. Dornhege, A. Hertle, and B. Nebel, “Lazy evaluation and subsumption caching for search-based integrated task and motion planning,” in IROS workshop on AI-based robotics, 2013.
[10] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, “Frob: An efficient heuristic for task and motion planning,” in Algorithmic Foundations of Robotics XI. Springer, 2015, pp. 179–195.
[11] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, “Frob: Leveraging symbolic planning for efficient task and motion planning,” arXiv preprint arXiv:1608.01335, 2016.
[12] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, “Combining task and motion planning through an extensible planner-independent interface layer,” in Robotics and Automation (ICRA), 2014 IEEE International Conference on. IEEE, 2014, pp. 639–646.
[13] L. de Silva, A. K. Pandey, M. Gharbi, and R. Alami, “Towards combining htn planning and geometric task planning,” arXiv preprint arXiv:1307.4382, 2013.
[14] M. Gharbi, R. Lallement, and R. Alami, “Combining symbolic and geometric planning to synthesize human-aware plans: toward more efficient combined search.” in Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. IEEE, 2015, pp. 6360–6365.
[15] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, “Finding locally optimal, collision-free trajectories with sequential convex optimization.” in Robotics: Science and Systems, vol. 9. Citeseer, 2013, pp. 1–10.
[16] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, “Motion planning with sequential convex optimization and convex collision checking.” The International Journal of Robotics Research, vol. 33, no. 9, pp. 1251–1270, 2014.
[17] A. Nouman, I. F. Yalciner, E. Erdem, and V. Patoglu, “Experimental evaluation of hybrid conditional planning for service robotics,” in International Symposium on Experimental Robotics. Springer, 2016, pp. 692–702.
[18] A. ten Pas, M. Gualtieri, K. Saenko, and R. Platt, “Grasp pose detection in point clouds,” The International Journal of Robotics Research, vol. 36, no. 13-14, pp. 1455–1473, 2017.