Conditional Task and Motion Planning through an Effort-based Approach

Nicola Castaman, Elisa Tosello, and Enrico Pagello
Intelligent Autonomous Systems Laboratory, Department of Information Engineering, University of Padova, Italy
{castaman, toselloe, epv}@dei.unipd.it

***Abstract*—This paper proposes a Conditional Task and Motion Planning algorithm able to find a plan that minimizes robot efforts while solving assigned tasks. Unlike most of existing approaches that replan a path only when it becomes unfeasible, the proposed algorithm takes into consideration a replanning every time an effort saving is possible. The effort is here considered as the execution time but it is extensible to the energy consumption. The computed plan is both conditional and dynamically adaptable to the unexpected environment changes.Authors prove the completeness and scalability of their proposal.**

I. Introduction

Let a human assign tasks to a robot. These tasks can require, for example, the pick of a can of coke from a cluttered table or the navigation towards a goal state. 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 [2], efficient algorithms exist to solve task and motion planning problems in isolation; however, their integration is still a challenge in terms of scalability, completeness, and generality.

An algorithm is scalable if it maintains the same efficiency when the workload grows. In the above examples, while solving the assigned tasks, the robot has to evaluate its surrounding and decide which objects to move and where to place them in order to pick up the coke. It has to solve a Navigation Among Movable Obstacles (NAMO) [4, 9] problem while finding a path to the assigned goal state. It has to decide if manipulating an obstacle is necessary, it can considerably reduce the planning costs, or it is contrariwise avoidable and moving on the free space is more convenient. Moreover, multiple subtasks exist fulfilling the assigned ones and different combinations of subtasks can bring to the same result: a mobile manipulator robot can move both its arm and base in order to approach the table and it can manipulate objects on the table in multiple ways and orders. These considerations let deduce that a task planning problem is exponential in the number of subtasks and a motion planning problem is exponential in the number of objects populating the working scene.

Starting from [1], authors combine a Fast-Forward (FF) task planner [8] with a Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (KPIECE) motion planner [13] generating a planner that takes into consideration only those objects and subtasks that minimize robot efforts; in the case in analysis the effort is the execution time, but also the robot's energy can be considered. This implementation choice aims to reflect the human behavior: humans take plans while efficiently managing their time and energy. Moreover, limiting each time choices revaluation to the ones that minimize the robots effort means trying to maintain the scalability of the system in terms of both Task and Motion Planning. In addition, if compared with other Motion Planning algorithms, KPIECE makes a better use of the information collected during the planning process. This information is used by KPIECE to decrease the amount of forward propagation it needs. As consequence, both runtime and memory requirements decrease making the algorithm suitable to handle high dimensional systems with complex dynamics [13].

To be complete, an algorithm must find a solution if one exists; otherwise, it should correctly report that no solution is possible. This goal becomes challenging if applied on a robotics system acting in a real world because the real world is non-deterministic: the planner has no complete and certain knowledge of the environment and actions can have unpredictable effects. If these effects are known and few in number, Conditional Planning can handle all possible outcomes. If the full set of possible effects is either unknown or too large to be enumerated in an operator, actions execution should be monitored in order to check if the current state of the world is as the plan says it should be. Replanning should take place when somethings goes wrong. Therefore, robot plans should dynamically adapt to changes while avoiding the explosion of paths due to the multiple alternatives introduced by Conditional Planning. In these terms, guaranteeing completeness is not trivial.

Authors have been inspired by [10], where a Hybrid Conditional Planning integrates low-level feasibility checks into the executability conditions of actuation actions. Based on robots capabilities, the proposed Task Planning pipeline is composed of a set of conditions that generate an actions tree. If some conditions cannot be a priori defined, an online replanning is adopted, that means: the robot monitors its actions during their execution by handling the sensors feedback, and it replans the path in case of failure. It is possible to prove the completeness of the proposed algorithm. In fact, FF always finds a task plan if one exists. KPIECE is probabilistically complete, as stated in [13], and the implemented planner handles known and unknown events through a Conditional Planning and an online replanning, until a maximum 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.

II. Related Work

Various researchers tried to combine the symbolic reasoning of task planning with the geometric reasoning of motion planning. Dornhege et al. [5] calls the motion planner to check the geometric feasibility of the planned tasks. Garrett et al. [6] integrates geometric information with the state-of-the-art FF task planner. [12, 3, 7] propose solutions based on Hierarchical Planning that evaluate task-level decisions with low-level geometric-reasoning modules. In particular, Srivastava et al. [12] combines off-the-shelf task planners with an optimization-based motion planner that exploits a heuristic function to remove potentially-interfering objects. All these approaches evaluate the objects relocation only when free-space motion planning is unfeasible. The proposed approach, instead, revalues a plan every time an action can save effort, not only when a 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 $\mathrm{TP} : (s_0, s_\mathrm{G}, A) \rightarrow p^*$ 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 PDDL domain [11] with a set $\mathrm{precon}(a)$ of preconditions and a set $\mathrm{effect}(a)$ 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

$$p = \langle s_0, a_0, ..., s_{\mathrm{N}-1}, a_{\mathrm{N}-1}, s_\mathrm{N} \rangle, \quad s_\mathrm{N} = s_\mathrm{G}$$

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

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

$$p^* = \mathrm{argmin}_{p \in P} \sum_{\langle s,a \rangle \in p} \mathrm{Cost}(\langle s, a \rangle)$$

$\mathrm{Cost}(\langle s, a \rangle)$ is the cost of action a being executed in state s.

B. Deterministic Motion Planning

A motion planner $\mathrm{MP} : (s_0, s_\mathrm{G}, A) \rightarrow t^*$ tries to find an *optimal* path $t^* \in \tau$ that lets R move from $s_0 \in S$ 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 τ, where $t \in \tau$ is a path in the free space:

$$\tau : [0, 1] \rightarrow C_\mathrm{free}, \quad \tau(0) = s_0, \quad \tau(1) = s_\mathrm{G}$$

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

$$t^* = \mathrm{argmin}_{t \in \tau} \Big(\mathrm{Length}(t) \Big)$$

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. $\mathrm{TMP} : (s_0, s_\mathrm{G}, A) \rightarrow t^*$ 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:

$$t^* = \mathrm{argmin}_{p \in P} \Big(\sum_{0 \leq i \leq |p|} \mathrm{Cost}(t_i | a_i) \Big)$$

where $\mathrm{Cost}(t_i|a_i)$ 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: $\mathrm{Cost}(t_i|a_i) = \mathrm{Effort}(t_i|a_i)$ and $\mathrm{Effort}(t|p) = \sum_{0 \leq i \leq |p|} \mathrm{Effort}(t_i|a_i)$. The effort is defined as the execution time.

The problem is deterministic if the actions space is a propri fully defined and each action is executed infallibly. However, in the real world actions can generate unexpected effects 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 [10]. 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 decision tree 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

Authors use FF as Task Planner and KPIECE as Motion Planner. FF is used to compute one possible sequence of actions letting the robot accomplish the assigned task. Given the sequence of actions, KPIECE is used to compute the path and, possibly, the sequence of control inputs bringing a robot from its start configuration to the goal one. The sequence of control inputs lets deduce the effort needed by the robot to perform motions, in this case in terms of execution time.

Given all possible actions A that the robot can perform, expressed in PDDL, Algorithm 1 begins by computing one feasible task plan $p = \langle s_0, a_0, ..., s_\mathrm{i}, a_\mathrm{i}, s_{\mathrm{i}+1}, ..., s_\mathrm{G} \rangle$ able to solve the assigned task (by using FF) and a decision k-ary tree data structure T is built having as edges the actions of p and as nodes the states of p (see Algorithm 2). This plan is supposed to be optimal and Algorithm 1 computes the trajectory t^* and cost c^* associated to its execution. t^* is estimated by performing a *Lazy* motion search connecting every couple of states of $p \in T$, starting from the root (s_0). A *Lazy* motion planner, in this case a *Lazy* KPIECE, finds a path between two states while not checking for collisions. The aim is driving the robot towards the shortest path, in terms of Euclidean distance to the goal. If the environment is unknown, t^* is computed until the last visible state.

Given $t^* = t_\mathrm{Lazy}$, the collision checking is performed on every node of T, starting from s_0, and every edge, starting

Algorithm 1: TMP algorithm

Input: s_0: Start state; s_G: Goal state; A: Set of actions that R can do
Output: (t^*, c^*): Path of minimum cost letting R execute the best plan

P = {}; // set of plans
$Obstacles$ = {}; // list of encountered obstacles
Initialize an empty k-ray tree data structure T;
$p \leftarrow \mathrm{TP}(s_0, s_\mathrm{G}, A)$;
P.pushBack(p);
$t^* \leftarrow \mathrm{LazyKPIECE}(s_0, s_\mathrm{G})|p$;
$c^* \leftarrow \mathrm{Effort}(t^*|p)$;
$T^* \leftarrow \mathrm{KTree}(P)$;
$node^* \leftarrow T.root()$;
$cost$ = 0;
t = {};
Traverse($node$);
return (t^*, c^*);

Algorithm 2: KTree(P)

Input: P: the set of plans
Output: T: the k-ary tree of plans

foreach *state* $s_i \in P$ **do**
  if $(s_i, a_i) \rightarrow s_{i+1}$ **then**
    s_i.children[].pushBack(s_{i+1});
    s_{i+1}.parent[].pushBack(s_i);

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

Input: v: the node to be expanded

while *(v.hasChilds()) && !(MaxAttempts reached)* **do**
  $t_\mathrm{Lazy} \leftarrow \mathrm{LazyKPIECE}(v, child)$;
  while t_Lazy *has new collision* $\in C_\mathrm{movable}$ **do**
    $obj \leftarrow \mathrm{findCollisionObject}(collision)$;
    if $(obj_\mathrm{label}, obj_\mathrm{pose}) \notin Obstacles$ **then**
      $Obstacles$.pushBack(obj);
    $A_{obj} \leftarrow \mathrm{findPossibleActions}(obj)$;
    foreach $a \in A$ **do**
      $S_\mathrm{effect} \leftarrow \mathrm{findEffectStates}(a)$;
      foreach $s \in S_\mathrm{effect}$ **do**
        if $\exists p_\mathrm{before} \leftarrow \mathrm{TP}(v, s, A)$ **then**
          foreach *child of* v **do**
            if $\exists p_\mathrm{after} \leftarrow \mathrm{TP}(s, child, A)$ **then**
              $T \leftarrow \mathrm{updateKTree}(T, p_\mathrm{before})$;
              $T \leftarrow \mathrm{updateKTree}(T, p_\mathrm{after})$;
  $t \leftarrow t + \mathrm{KPIECE}(v, child)$;
  $cost \leftarrow cost + \mathrm{Effort}(v, child)$;
  if $cost < c^*$ **then**
    if $v == s_\mathrm{G}$ **then**
      $c^* \leftarrow cost$;
      $t^* \leftarrow t$;
      return;
    else
      Traverse($child$)

from (s_0, s_1). An edge is the motion path used to execute an action. For every new collision detected in the space of movable obstacles C_movable, a state $s_\mathrm{j} \in S$ is sampled. Based again on the set of feasible actions A, their preconditions, and effects, T is extended by adding the plans from/to s_j. The algorithm evaluates all the plans $\langle s_\mathrm{i}, a_\mathrm{i}, ..., s_\mathrm{j}, a_\mathrm{j}, ..., s_{\mathrm{i}+1} \rangle$ and the related paths. Given a plan connecting s_i to $s_{\mathrm{i}+1}$, if the cost

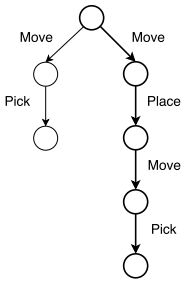


Fig. 1: Conditional Planning: the robot has to pick an object. The figure depicts the $Pick$ decision tree generated in accordance with the satisfied preconditions. On the left, there is the sequence of actions to be performed when the gripper is empty. On the right, there is the sequence of actions to be performed when the gripper is not empty and the object it is holding has to be placed down before executing a new pick.

of the associated path is less than c^*, the algorithm continues the exploration of this branch until reaching s_G. Otherwise, it explores the other branches until a better solution is found, all children have been visited, or the maximum number of attempts has been reached.

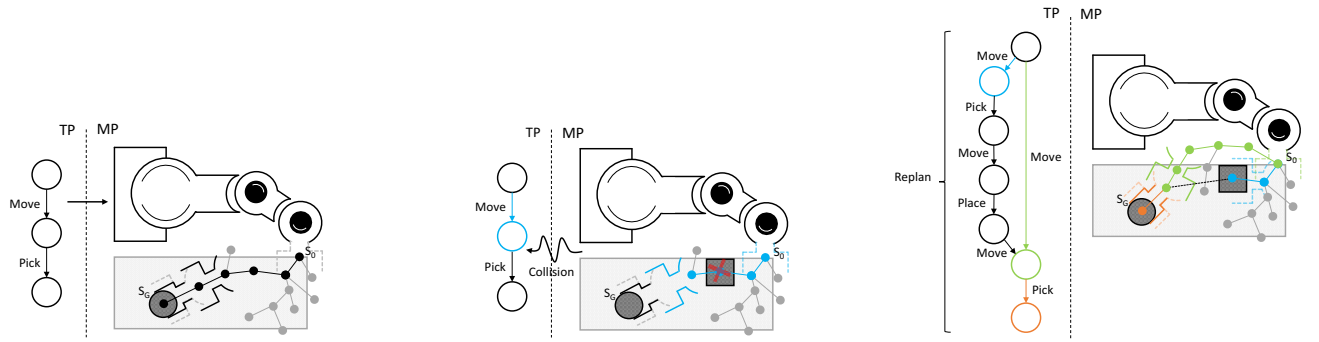
This means that p is revalued not only when unfeasible, but every time better solutions exist with respect to the selected one. E.g., once an obstacle is found and a path avoiding it is computed, the algorithm evaluates both the action of avoiding the object and the one of manipulating it. It then selects the alternative of minimum effort, that in the case in analysis means the one requiring the minimum execution time.

A. Conditional Planning and on-line Replanning

Suppose the robot R is executing a trajectory t^*, coming from an action $a \in p^*$, and meanwhile it is perceiving its surrounding. Two kind of events may occur: 1) contemplated events (e.g., the robot has to manipulate an object but its gripper is not empty or the object is occluded); 2) unpredictable faults (e.g,. the manipulated object falls down from the gripper, an unexpected obstacle blocks the way, a new movable obstacle $(obj_{\mathrm{new}_\mathrm{label}}, obj_{\mathrm{new}_\mathrm{pose}}) \notin Obstacles$ is visible, manipulation fails due to an underestimated effort needed to move the object). In the first case, events are already handled inside the plan thanks to a conditional tree. No replanning procedure is involved, just the right sequence of actions is chosen (see Figure 1). The second case, instead, requires a replanning: the procedure samples the new encountered state s_new, it tries to connect the state to T using the set of actions that the robot is able to perform, and invokes $\mathrm{Traverse}(s_\mathrm{new})$ trying to find a path that minimizes the execution time (see Figure 2). A number of attempts are chosen a priori: if no plan is found and totally executed within those attempts, the system outputs a failure.

V. Experiments

Figure 3 depicts the use cases authors are studying at the time of the submission. Figure 3a shows a mobile manipulator



(a) TP calculates a sequence of actions that achieves the assigned task. MP computes the corresponding sequence of motions.

(b) During the execution of the motion plan, the robot perceives a new obstacle.

(c) TP samples new states and tries to connect them to the existing plan. If a connection exists, a new sequence of actions is generated and the less onerous branch is expanded.

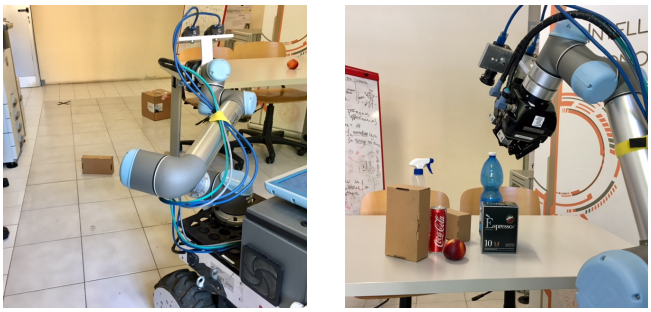
Fig. 2: Online Replanning: a robot perceives a new obstacle while trying to pick up a known object. On the left of each figure, the plan of the Task Planner (TP) is depicted. On the right, the motion planner (MP)'s search space is visible.

robot trying to solve a NAMO problem. Figure 3b shows the same robot trying to pick up an occluded can of coke from a cluttered table. The robot can perform four different actions:

- $Move_{\text{base}}(pose_{\text{start}}, pose_{\text{goal}}, traj)$;
- $Move_{\text{arm}}(pose_{\text{start}}, pose_{\text{goal}}, traj)$;
- $Pick(obj, gripper, pose_{\text{gripper}}, pose_{\text{obj}}, conf_{\text{joints}}, traj)$;
- $Place(obj, gripper, pose_{\text{gripper}}, pose_{\text{obj}}, conf_{\text{joints}}, traj, pose_{\text{goal}})$.

If $pose_{\text{goal}}$ is not given as input, $Move$ randomly samples it on the free space and $Place$ does the same on a flat surface in the neighbourhood of the manipulated object.

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.



(a) A mobile manipulator robot trying to solve a NAMO problem.

(b) The same robot trying to pick up an occluded can of coke from a cluttered table.

Fig. 3: Use cases.

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] N. Castaman, E. Tosello, and E. Pagello. A Sampling-Based Tree Planner for Navigation Among Movable Obstacles. In *ISR 2016: 47st International Symposium on Robotics; Proceedings of*, pages 292–299. VDE, 2016.

[2] 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.

[3] L. de Silva, A. K. Pandey, M. Gharbi, and R. Alami. Towards combining htn planning and geometric task planning. *arXiv preprint arXiv:1307.1482*, 2013.

[4] M. Dogar and S. Srinivasa. A framework for push-grasping in clutter. *Robotics: Science and systems VII*, 1, 2011.

[5] 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*, pages 99–115, 2012.

[6] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *arXiv preprint arXiv:1608.01335*, 2016.

[7] 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*, pages 6360–6365. IEEE, 2015.

[8] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[9] M. Levihn, J. Scholz, and M. Stilman. Hierarchical decision theoretic planning for navigation among movable obstacles. In *Algorithmic Foundations of Robotics X*, pages 19–35. Springer, 2013. ISBN 978-3-642-36279-8.

[10] 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*, pages 692–702. Springer, 2016.

[11] 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*, 33(9):1251–1270, 2014.

[12] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 639–646. IEEE, 2014.

[13] I.A. Şucan and L.E. Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*, pages 449–464. Springer, 2009.