Available online at www.sciencedirect.com
ScienceDirect
Robotics and Autonomous Systems 00 (2016) 1–15
Robotics and Aut. Systems

Unified Framework for Path-Planning and Task-Planning for Autonomous Robots

Pablo Muñoz[a,*], María D. R-Moreno[a], David F. Barrero[a]

[a] *Universidad de Alcalá, Departamento de Automática*
Campus Universitario, Ctra. Madrid-Barcelona, Km. 33,600
28871 Alcalá de Henares, Spain

Abstract

Most of the robotic systems are designed to move and perform tasks in a variety of environments. Some of these environments are controllable and well-defined, and the tasks to be performed are generally everyday ones. However, exploration missions also enclose hard constraints such as driving vehicles to many locations in a surface of several kilometres to collect and/or analyse interesting samples. Therefore, a critical aspect for the mission is to optimally (or sub-optimally) plan the path that a robot should follow while performing scientific tasks. In this paper, we present UP2TA, a new AI planner that interleaves path-planning and task-planning for mobile robotics applications. The planner is the result of integrating a modified PDDL planner with a path-planning algorithm, combining domain-independent heuristics and a domain-specific heuristic for path-planning. Then, UP2TA can exploit capabilities of both planners to generate shorter paths while performing scientific tasks in an efficient ordered way. The planner has been tested in two domains: an exploration mission consisting of pictures acquisition, and a more challenging one that includes samples delivering. Also, UP2TA has been integrated and tested in a real robotic platform for both domains.

Keywords: Task-planning, Path-planning, Autonomy, Robot control architectures

1. Introduction

Nowadays, most of the robotic systems that are being designed have to move and perform tasks in a variety of environments. In order to do that, they have to avoid obstacles, find a collision free trajectory, and plan tasks. Some of these systems aim to help with ordinary, but tedious tasks. Generally, they move in known surfaces and the number of decisions is usually limited such as night surveillance tasks.

However, new science missions such as the two main missions *ExoMars* and *MARS Sample Return* (part of the Aurora program of the European Union Council of Research) will require more capable systems to achieve the goals that they are designed for. Mobility and science capabilities of the rovers are increasing, which require more powerful tools to assist on-ground operators and autonomous capabilities for the rovers. In previous missions, optimality in the path followed by the rover was not a difficult task due to the short distances they could travel per day. However, a critical aspect in future missions is to optimally (or sub-optimally) plan the path that a robot should follow while performing scientific tasks. In addition, improvements for such domains can also be incorporated into commercial robotic applications, e.g. performing inventory tasks in a large warehouse or autonomous logistics domains [1].

In this direction, the paper presents some of the results obtained within the Ph.D. program on the topic of *Cooperative Systems for Autonomous Exploration Missions* supported by ESA. Particularly, we are pursuing autonomous navigation with a rover for an exploration domain with several scientific targets along a known terrain. To face this problem, we initially had a complex deliberative model expressed in Planning Domain Definition Language (PDDL) [2] in which path-planning and task-planning were solved with a PDDL planner. This solution was very limited in (i) the map size because the planner only solved small grids in a reasonable time, and (ii) the quality of path generated because the planner generated suboptimal paths that were impossible to improve by using recent research advantages on path-planning algorithms. A second approach was to partially detach path-planning and task-planning. Before

*Corresponding author
Email addresses: pmunoz@aut.uah.es (Pablo Muñoz), mdolores@aut.uah.es (María D. R-Moreno), david@aut.uah.es (David F. Barrero)

the search process starts, it is possible to include in the PDDL model a visibility graph with connections between tasks, which is generated using a path-planning algorithm. Then, the PDDL planner can obtain a feasible solution to achieve all tasks. Finally, the path-planning algorithm generates paths for the movements between each pair of locations in the plan. However, in our experiments we observed that solutions were not optimal. This was because of the domain independent heuristic employed by the task-planner. Considering a movement in the same way as other actions (e.g. take picture) leads to not properly consider the distance between tasks.

For this reason, we have developed an Artificial Intelligence (AI) planner that integrates capabilities of path-planning and task-planning. The main idea is to take advantage of path-planning heuristics and merge them with domain independent heuristics to generate better solutions in robotic domains. The proposed planner, called Unified Path-Planning and Task-Planning Architecture (UP2TA), is able to plan paths considering the shortest path while performing scientific tasks in an efficient ordered way. A PDDL planner is responsible of ordering the tasks while a path-planning algorithm searches for the route between tasks. In UP2TA, these planners are highly coupled, allowing to merge the heuristics of both planners in order to provide better solutions for mobile robotics domains. We perform an experimental evaluation of our planner on two classical exploration domains: pictures acquisition and samples delivering. In addition, UP2TA is currently deployed as the deliberative layer of the Model-Based Autonomous Controller (MoBAR) [3, 4], which has been also used in our evaluation.

The next section reviews those heuristic algorithms for path-planning that are used within our planner. Section 3 presents a brief description of heuristic planners in the state-of-the-art, with a special focus on the FF planner that we use in our integration. Section 4 defines approaches that interleave task-planning and motion/path-planning. Section 5 describes our initial attempt to solve problems without a heuristic integration schema. Section 6 introduces two application domains that are then used in an experimental evaluation, which is shown in Section 8. Section 9 presents some conclusions.

2. Path-Planning Review

In this section, we review the most common path-planning algorithms and their features. In particular, we focus on heuristic search algorithms applied to path-planning since we use those in our integration.

2.1. Representation and Notation

In classical path-planning, the environment is usually represented as a two-dimensional uniform grid with blocked and unblocked cells [5]. There are two types of representations: the node can be in the centre of the cell (centre node representation) or the node can be on the corner of the cell (corner node representation). In both cases, a valid path is the one that starts from the initial node and reaches the goal without crossing a blocked cell.

From now on, consider a node p as a random node, and s and g as the initial and goal nodes respectively. Each node p is defined by its coordinate pair (x_p, y_p). A solution has the form $(p_1, p_2, ..., p_{n-1}, p_n)$ where $p_1 = s$ and goal $p_n = g$. For each node p we require the following information:

- predecessors(p): the predecessor node of p. It is mandatory that the straight line between p and its predecessors does not cross blocked cells.
- successors(p): a list of nodes that are reachable from p. The predecessors of each node t in the list is p. Therefore, if predecessors(t) = $p \Rightarrow t \in$ successors(p).
- G(p): the length of the path from s to p.
- H(p): the heuristic value of p, i.e., an estimation of the distance from p to g. Typically, the A* algorithm [6] uses the Octile distance, while Theta* [7] uses the Euclidean distance.

2.2. Heuristic Path-Planning Algorithms

A path-planning problem can be represented as a search tree over grids. This representation has been widely discussed. Algorithms such as A* allow us to quickly find routes at the expense of an artificial restriction of direction changes of $\pi/4$. A* Post Smoothed (A*PS) [8] tries to smooth the path obtained by A* in a post-processing step. Therefore, the resulting path may be shorter than the original, but has higher running time. If A* finds a path, the smooth process checks the line of sight between every node and the successor's successor node, removing intermediate nodes when possible. Also, there have been some improvements such as its application to non-uniform costs setting in Field D* [9], or more recently Theta* [7], which aims to remove the restriction on direction changes that A* generates. Both algorithms use the same evaluation function as in eq. 1.

$$F(p) = G(p) + H(p) \quad (1)$$

The main difference between A* and Theta* [7] is that the former only allows that the predecessors of a node is its predecessor, while in the latter the predecessors of a node can be any node. Usually, path-planning algorithms compliant with this property (i.e. predecessors(t) = $p \Rightarrow t \in$ successors(p) is no longer mandatory) are called any-angle path-planning algorithms. This property allows Theta* to find shorter paths with fewer turns compared to A*. However, this improvement implies a higher computational cost due to additional operations performed during the nodes expansion process. These algorithms work on fully observable environments except Field D* that can deal with partially observable environments applying a replanning scheme.

From the Theta* algorithm, some effort has been performed in order to improve its performance such as Incremental Phi* [10] that takes into consideration the free-space assumption and angle ranges computation to provide a speed-up of approximately one order of magnitude with respect to Theta*; or Lazy Theta* [11] that delays the computation of the line of sight

checking (decreasing the number of checks), and improves the performance in order to deal with 3D cubic environments.

The Smooth Theta* (S-Theta*) [12] algorithm, developed from Theta*, aims to reduce the amount of turns that the robot should perform to reach the goal. The motivation is that robots could require more time when turning, although is not desirable to rotate big angles in soft terrains. The S-Theta* algorithm considers early in the search process those nodes that follow the current heading of the robot. Then, the evaluation function of A* or Theta* is modified by adding a third parameter as in eq. 2, which evaluates the direction of the successor node with respect to the current predecessors' position and the goal node.

$$F(p) = G(p) + H(p) + \alpha(p) \quad (2)$$

This means that, at the beginning of the search process the algorithm tries to follow the line connecting the initial node to the goal node. That is, the shortest route, if there are no obstacles. When the initial direction needs to be changed because of an obstacle in the path, the algorithm expands nodes taking into consideration both the path length and the heading change necessary to reach the successors. In this way, the search continues towards a path with less amplitude turns, but it may obtain a bit longer paths. This behaviour makes that S-Theta* can perform heading changes at any point of the map, not only at the corners of the obstacles as Theta* does. Fig. 1 shows the α value for nodes r and t, which can be expanded when there is an obstacle q on the current direction. We use the $\triangle ptg$ and $\triangle prg$ triangles to compute $\alpha(t)$ and $\alpha(r)$, where $p = predecessors(q)$ and $(r,t) \in successors(q)$. Since the value of $\alpha(t)$ is lower than $\alpha(r)$, S-Theta* will expand t. Conversely, Theta* would expand r.

In the same way, if we consider the $\alpha(p)$ value of a node as part of the heuristic function instead of using it in the cost function, the result is a path-planning algorithm that has a greedy behaviour. This means that the algorithm tries to follow the straight line between the initial and goal points s and g as a result of the modified heuristic, which computes $\alpha(p)$ as the angle formed by the triangle $\triangle spg$ for any node p.

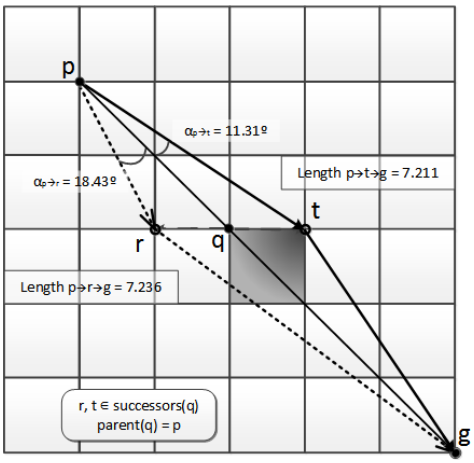


Figure 1. α values with an obstacle in the current direction.

Muñoz and R-Moreno [13] applied this technique to the A*PS and Theta* algorithms. They showed that the modified heuristic required less time and memory to obtain a path, but increased its length with respect to Theta*. This is due to the number of expanded nodes: algorithms using this heuristic expand fewer nodes than using their original ones. Fig. 2 shows a small example of A* and the path and nodes expanded using the Octile distance, and A* using $\alpha(p)$ as part of the heuristic function (presented as Greedy-A*). It is also possible to deal with the resulting path length, the search runtime, and the memory required using a factor in order to weight the value of the $\alpha(p)$ function. Since the heuristic computation is the only modification in the algorithm, it is guaranteed that if there is a path between two nodes, the algorithm will find it.

3. Heuristic Search Planners

Heuristic search planners use heuristic functions $H(s)$ to find a sequence of actions that reaches the goal from the initial state [14]. Heuristic functions are commonly generated by simplifying the original problem. This is known as relaxation. A common relaxation-based heuristic in planning is the *delete relaxation*, which consists of ignoring delete effects. Considering a delete relaxation problem, the estimated cost $H^+(s)$ from any state s to the goal state can be seen as a lower bound of the optimal cost $H^*(s)$ for the original problem. As a result, the delete relaxation heuristic $H^+(s)$ can be used as a heuristic for solving the original problem. The *additive* and *max* heuristics are examples of relaxation-based heuristics. In particular, the *additive* heuristic, $H^{add}(s)$, estimates the cost of achieving a proposition p from a state s as the minimum cost among all the actions a that produce p. Such cost is defined as the sum of the cost of the individual preconditions of a. The *max* heuristic, $H^{max}(s)$, estimates the cost of achieving a proposition p from a state s as the minimum cost among all the actions a that produce p. Such cost is defined as the maximum cost among all preconditions of a.

Another relaxed-based heuristic is the high-order heuristic, $H^m(s)$ [15], which computes estimates of costs for sets of m propositions from the initial state. The Pattern Databases (PDB) heuristic [16] abstracts some variables of the problem, mapping

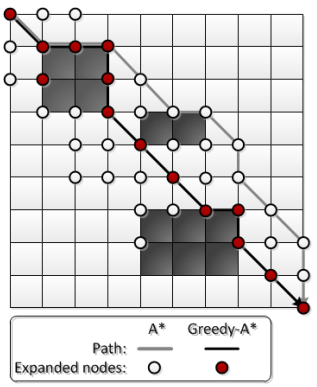


Figure 2. Greedy-A* search for path-planning versus A*.

states of the original state space into the abstracted space in a way that all states match the abstract space. The $H^{max}(s)$, $H^{m}(s)$, and PDB heuristics are admissible and, therefore, they can be used in an A* algorithm for optimal planning. On the other hand, the $H^{add}(s)$ is not admissible, but it has shown to be compelling in practice.

Recently, new search enhanced techniques such as helpful actions [17], preferred operator [18], deferred evaluation [19], and landmarks [20] have been developed to speed up the search. However, in the last few years landmark detection algorithms have been studied by the planning community to develop new techniques to find and order landmarks [21].

There is a number of relevant heuristic planners that have shown good performance in recent International Planning Competitions (IPC). Some of them were initially considered candidates for being used in the task-planning side of an autonomous controller (MoBAr in our case) such as Fast Downward [18] or LAMA [22]. These two planners use finite-domains rather than binary state variables. LAMA is currently one of the best-performing sequential planners. Another IPC winner is OPTIC [23], a temporal POP-based planner that outperforms some IPC5 planners on problems that deal with goal deadlines.

Recent planners can deal with complex domains that include temporal constraints or preferences [24, 25]. However, we have decided to use the well-known FF [17] planner in our integration. It was the winner of some competitions in the past. It is written in C, making it very compact and fast, suitable for real robots applications when the CPU is limited. Some newer planners such as Metric-FF [26] and SGPlan [27] rely on FF. Therefore, it seems to be possible that we can integrate our modifications without much effort in FF-based planners. Since the aim of the paper is to evaluate the integration between a path-planning heuristic and a domain-independent heuristic provided by a task-planner, FF seems to be good enough for our purposes without requiring some engineering effort to made the integration. Newer planners are usually more complex to be modified due to external modules, lexical analysers, among others.

The FF planner is based on a *relaxed planning graph*. In general, a planning graph is a directed graph composed of two types of nodes distributed on several levels. Starting at level 0, the even levels contain information about the propositions that are reached – *fact layers*. The odd levels contain actions whose preconditions are present at the previous level – *action layers*. The edges represent the preconditions and effect relations (positive (*add*), negative (*del*) and conditions that are maintained without executing any action (*no-op*)) between facts and actions. Since delete relaxation is assumed, a relaxed planning graph does not deal with negative edges. Fig. 3 shows the FF architecture. This architecture is implemented by three main modules that are summarized next.

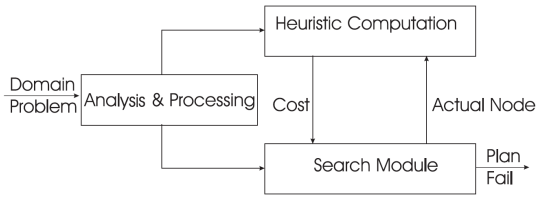


Figure 3. FF general structure.

- The *Analysis & Processing* module analyses and processes all the information that comes from the domain and the problem.
- The *Heuristic Computation* module calculates the cost of applying a determined node in the search process. The heuristic used in FF is based on the number of actions or length in an explicit solution from a relaxed-plan graph algorithm. The length of the solution (that is, the number of actions) is taken as an estimation of how far the goal is from the current state. When calculating the number of actions, the heuristic always prefers the *no-ops* against actions since they count as zero.
- The *Search* module directly depends on the heuristic computation. FF uses two search algorithms in combination with the mentioned heuristic. First, FF uses Enforced Hill Climbing; when a local minimum is encountered, breadth-first search is used until a state with a strictly better heuristic is found; if breadth-first fails, FF restarts to Best First Search (BFS). BFS is similar to the A* algorithm explained in the previous section, with the difference that nodes are sorted according to their heuristic value.

4. Integration of Task-Planning and Path-Planning

Traditionally, task-planning and motion/path-planning problems were covered by separating both problems, i.e., high-level task modelling ignores low-level constraints. This simplification results in inefficient or even infeasible solutions. Then, works that integrate task-planning and motion/path-planning to solve problems such as object manipulation or mobile robotics domains have been reported in the literature, providing better solutions. We can mention work done by Zacharias et al. [28], in which a two-arm humanoid robot must manipulate objects on a table. The particular problem addressed is how to automatically rearrange and manipulate objects without collisions. This is done by using a motion-planner to compute trajectories, and a task-planner to provide the grasping operations that achieve a free-obstacle configuration.

In this direction, Cambon et al. implemented the aSyMov planner [29], which integrates Metric-FF for task-planning and Probabilistic RoadMaps (PRM) [31] for motion-planning. For the integration, both planners share a geometric representation of the position of the robots, obstacles, and objects in the environment. The plan extraction is guided by a task-planner, which initially provides a solution for a problem in which all motion paths are valid (i.e. no obstacles are considered). Then, a motion-planner checks if it is possible to achieve the goals. Selection of actions is guided by a cost function that adds i) the number of times that an action fails due to infeasible paths, and ii) the heuristic computed by the task-planner. When there are

no valid paths to achieve a goal, the motion-planner updates the geometrical information of the task-planner, which must generate a new plan. Finally, when a valid plan is generated, an improvement process is executed with the aim of optimizing the planned paths while attempting to parallelize actions in multi-robots domains.

The I-TMP algorithm [30] uses a motion-planning algorithm (a PRM) to determine trajectory constraints for a logical description of the tasks to perform. Such tasks are encoded in a formal language similar to PDDL and solved with and integrated planner that interleaves task and motion-planning. This planner shows a good performance for planning the navigation of legged robots.

The SAHTN algorithm [32] is designed to deal with pick-and-place domains for robots equipped with robotic arms. The algorithm implements a hierarchy from Hierarchical Task Networks (HTN) for task-planning to Rapidly-exploring Random Trees (RRT) [33] for low-level actions (i.e. arm movement). In this way, the hierarchy specifies a set of high-level actions that can be refined until low-level movements. For each movement, a reachable state space is produced that can be reused for different objects. By doing this, the associated search cost is reduced and, therefore, avoids infeasible solutions for the higher level.

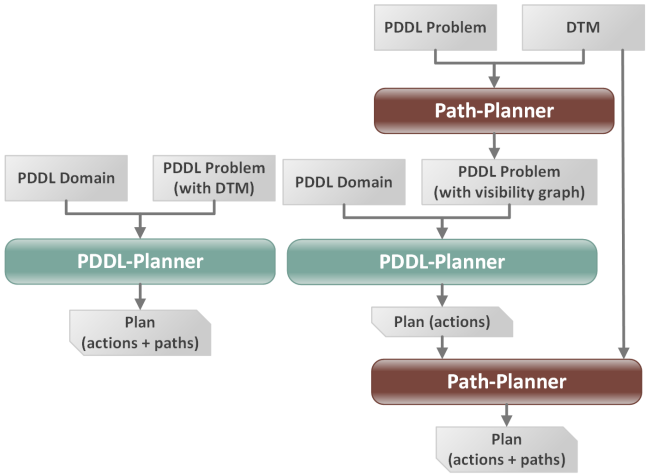
Some of the above-mentioned approaches combine task-planning and motion-planning in the search process in a very specific way, without considering a general approach for interfacing both planners. In that direction, Erdem et al. [34] present a framework that provides bilateral interaction between the task-planner and the motion-planner. This relationship allows both planners to guide the search: the task-planner aims to obtain an optimal task sequence and the motion-planner performs geometrical reasoning. Given a plan generated by the task-planner, the motion-planner must ensure that the plan is kinematically solvable. If it is not, the motion-planner will include new constraints in the task-planner. At the same time, during the plan search, the motion-planner includes specific domain information to guide the search. This framework also allows exploiting different motion-planning algorithms such as PRM or RRT.

In a similar direction, Srivastava et al. [35] propose an approach in which they integrate a PDDL planner (they use FF in their experiments) with a motion-planner without modifying them. To perform such integration, they design an interface layer that allows them to share information. The objective of the interface layer is to invoke the motion-planner to generate paths for the tasks produced by the PDDL planner. If there are no valid dynamics for a given task, the geometrics constraints discovered are abstracted and included in the PDDL model to fix the plan. The way that they perform such abstraction is one of the main contributions of the work.

In general, the above-mentioned approaches focus on integrating task-planning and motion-planning, which are more related to the manipulation of objects employing robotics arms. However, our work focuses on mobile robotics, having similarities with the previous works such as integrating domain specific information in the task-planner, interleaving task-planning with path-planning, or the possibility of using different planners.

Our first approach to solve a mobile robotics domain was to integrate a Digital Terrain Model (DTM) within the PDDL problem and provide a description of the movement actions in the domain. Then, we solved the problem using a PDDL planner (see fig. 4 (a)). However, this solution did not provide good results. The problem of using a PDDL-based planner to integrate both path-planning and task-planning presents two drawbacks. First, the complexity of the domain and the problem. In our domain we used eight possible movement actions *north, south, east west, north-east, north-west, south-east*, and *south-west*. However, the critical part is in the problem, where we need to provide the DTM in a grid form. This means that we have to define all nodes and their connections. In other words, what nodes are adjacent with each other. Second, the size of the grids (at least 500x500 nodes). The planner must instantiate a huge number of actions. Particularly, the movement action is instantiated for every adjacent pair of nodes. Considering the possibility of movement in eight directions, we can instantiate eight possible move actions for each node. This results in an exponential explosion of the search space. The more locations we define in the problem, the more instantiated operators will be generated. So, even for medium size maps, the search is intractable and no state-of-the-art planner could solve it (partially or fully grounded planners).

Solving small maps with this approach leads to routes with headings restricted to multiples of $\pi/4$ since we have defined movements in diagonal, horizontal and vertical, which may cause sub-optimal paths. We could have defined more operators in order to smooth this restriction, but the search was already intractable. In this direction, we have first used FF, increasing the grid size. Results show that the planner can manage 50x50 grids with 6 and 9 tasks in 1 and 2 seconds respectively, but requires a high amount of memory (near 1GB). Increasing the map size implies that FF is unable to manage the problem size, and thus, it cannot provide a solution. Since we wanted to test if the behaviour only occurred in FF, we did the same test with other fully grounded planners such as SGPlan, getting the same



a) Using a PDDL based planner b) Interleaving both planners

Figure 4. Possible solutions to merge path-planning and task-planning.

behaviour. We have also performed the same test for the partially grounded planner SatPlan2006 [36]. Results were even less encouraging since more time was needed to solve the basic problems that FF was able to solve.

A better solution is to decouple path-planning and task-planning using a three phases planning process (see fig. 4 (b)). The idea is to include a path-planner that generates a visibility graph considering all the waypoints given in a PDDL problem (that is, the initial and goal positions). This reduces the complexity of the PDDL domain because all the movement actions are reduced to a single one that traverses between waypoints. On the other hand, the PDDL problem includes only the relevant positions. By doing this, it is no longer required to include the DTM in the PDDL model, as the path-planner considers it during the path-search. This reduction in the number of actions and objects also decreases the workload of the task-planner. Then, the PDDL planner provides a solution in which there are movements between waypoints, but without providing the real path between them. The last phase replaces the movements between waypoints with a valid path by executing again the path-planner. Finally, we obtain a plan in which tasks and movements are properly interleaved.

This approach has two advantages i) it allows to easily interchange the PDDL planner and path-planner to exploit different algorithms, and ii) it eliminates zig-zag patterns and provides better routes when it uses an any-angle path-planning algorithm. However, this schema presents a major drawback: as there is no shared information between the path-planner and the task-planner during the search phase, solutions generated can be highly inefficient. In particular, the election of the task-planner has a big impact in the solution optimality as the domain independent heuristic does not take into account the distance among task during the search.

We have performed an experiment using SGPlan, OPTIC and UP2TA in a classical exploration domain where each task performs a single action in a determined location. In particular, the test consists of 10 maps of 100x100 nodes (considering a distance between nodes of 1 meter) with 20% of blocked cells. For each map, two scenarios with 6 and 12 tasks randomly distributed are defined. For all planners, the path-planning algorithm employed is S-Theta*, so the differences in the path-length to accomplish all tasks are due to the task ordering rather than the path-planning algorithm. The average results of this test is shown in table 1. Regardless of the reduced number of scenarios tested, the difference between the solutions provided is significant. SGPlan provides worse results than OPTIC, but UP2TA gives the best results. This is evidence that when we merge the heuristic of the path-planner and the heuristic of the task-planner we get better results than independently. A detailed discussion on this issue will be presented in the experimental sections.

Table 1. Results for the execution of different problems with UP2TA and two different PDDL-based planners. In bold best values.

# tasks	Planner	Runtime (*ms*)	Path length (*m*)
6	SGplan	2007	413.780
	OPTIC	2653	389.600
	UP2TA	**1661**	**164.363**
12	SGPplan	**8364**	1785.121
	OPTIC	69800	1386.400
	UP2TA	8917	**718.340**

5. Unified Path-Planning and Task-planning Architecture

The idea behind the UP2TA planner comes from our experience using a PDDL planner as the deliberative layer for the control of a mobile robot in exploration domains. We wanted to develop a planner that gets a closer to the optimal ordering of multiple tasks placed in a grid. In order to do that, we combine a modified FF planner and a path-planning algorithm to work together in a coordinated way. The resulting system, called UP2TA, takes the benefits of i) a PDDL planner for task-planning using PDDL problems and domain independent heuristics, and ii) a path-planning algorithm for generating better routes using a DTM and domain-specific heuristics.

The following subsections describe the files that are required by the new planner and some concepts that we use later in this paper. Then, we present the UP2TA planner, with a special focus on how the path-planning heuristic is combined with the heuristic search planner.

5.1. Input files

The task-planner cannot have any path-planning related operation. For this reason, the PDDL domain and problem do not contain the DTM (this is only used by the path-planning algorithm). The files required by UP2TA are:

- *Domain and problem:* UP2TA accepts any PDDL version that is supported by the PDDL planner used in the integration. However, UP2TA requires in the domain description an action called `Move_To` that allows the robot to move between waypoints (`wp`). Besides, the problem file does not need to provide a DTM, just the waypoints in which we want to perform a task or traverse through. These waypoints must have the form `Ca_b` to represent a position with coordinates `x=a` and `y=b`. Fig. 5 (a) shows a valid domain for UP2TA. Fig. 5 (b) shows an example of a problem file that has three waypoints: `C1_1`, `C9_5`, and `C6_8`; the first one represents the initial position (which is also the desired final one), and the second and third ones represent where the goals are placed. More details about this domain are explained in sec. 6.

- *DTM:* this file contains the information of the terrain. The codification and content depend on the path-planning algorithm used. If the algorithm works as the ones presented in sec. 2, a map with blocked and unblocked cells is required. However, more complex path-planning algorithms require also a traversal cost map, which contains the costs related to move through each region of the map for algorithms like Field D*, or a more complex DTM for working in 3D environments. There is a relationship

```
(define
 (:domain UP2TA-exploration)
 (:types wp subsyst ptu_aim - object)
 (:predicates
   (on ?s - subsyst) (off ?s - subsyst)
   (robot_at ?n - wp) (ptu_pos ?p - ptu_aim)
   (picture ?n - wp ?p - ptu_aim) ...
 )
 (:action Move_To
  :parameters (?n1 ?n2 - wp)
  :preconditions (and (robot_at ?n1) (on gnc)
                      (ptu_pos P0_0))
  :effect (and (not (robot_at ?n1))
               (robot_at ?n2))
 )
 (:action Take_Picture
  :parameters (?n - wp ?p - ptu_pos)
  :precondition (and (robot_at ?n) (off gnc)
                     (on cam) (ptu_pos ?p))
  :effect (picture ?n ?p)
 )
 (:action Drill
  :parameters (?n - wp)
  :precondition (and (robot_at ?n) (off gnc)
                     (on drill) (ptu_pos P0_40))
  :effect (sample ?n)
 )
)
```

a) Domain

```
(define
(problem explor01)
(:domain UP2TA-exploration)
(:objects
     C1_1 C9_5 C6_8 - wp
     P0_0 P30_20 P0_40 - ptu_aim
     gnc ptu cam drill - subsyst
 )
 (:init
     (robot_at C1_1)
     (ptu_pos P0_0)
     (off gnc)
     (off ptu)
     (off cam)
     (off drill)
 )
 (:goal (and
  (picture C9_5 P30_20)
  (sample C6_8)
  (robot_at C1_1))
 )
)
```

b) Problem

Figure 5. Example PDDL files for the UP2TA planner.

between the DTM and the PDDL files: each location defined in the PDDL problem (`Ca_b`) corresponds to a position in the DTM with coordinates x=a and y=b.

Using these files, the UP2TA planner provides a sequence of actions with the optimal (suboptimal) paths between the tasks, which achieves all the goals defined in the PDDL problem.

5.2. Concepts definition for UP2TA

The following terms are defined in order to understand the integration explained in the next subsection.

- T_i is a task to be performed, defined as a goal in the PDDL problem. Each task takes place on a waypoint `Ca_b`.
- $h_{FF}(T_i)$ is the FF heuristic function, which is computed using a relaxed plan graph. It returns the estimated number of actions required to reach a plan for a given goal.
- $dist(T_i, T_j)$ is a specific path-planning heuristic. It is generally computed as the Euclidean distance (straight line distance between two waypoints).
- $greedy(T_i, T_j)$ is the cost of a fast computed path between two waypoints. This path is generated using a modified greedy path-planning algorithm [13], which returns a suboptimal path with lower runtime and memory usage.
- $path(T_i, T_j)$ is the path length between two waypoints. This path is computed using a classical path-planning algorithm such as. A* or Theta*.
- $H(T_i)$ is the heuristic function computed for a task. When a task requires a movement between waypoints, the heuristic is computed by adding the heuristic of the task-planner (h_{FF}) and the heuristic from the path-planner.
- $C(T_i)$ is the cost associated to perform T_i. This is value is computed by the task-planner. Since we use FF, each action has unit cost (for non-movement actions).

5.3. Integrated planner description

Fig. 6 shows the UP2TA planner scheme. It works as a single system, but it has two different parts: the task-planner and the path-planner. The control of the system resides in the task-planner, i.e. a PDDL planner. The process starts reading the PDDL data and instantiating the defined objects such as the robot's initial position, the subsystems' state, and the tasks to be performed. However, we can introduce other parameters and/or constraints into the PDDL domain such as the subsystems restrictions or hierarchical tasks. The DTM is given to the path-planner.

Consider the domain and problem definitions shown in fig. 5. In order to achieve the goal `picture C9_5 P30_20` (denoted as T_1), the robot must move (action `Move_To`) from the

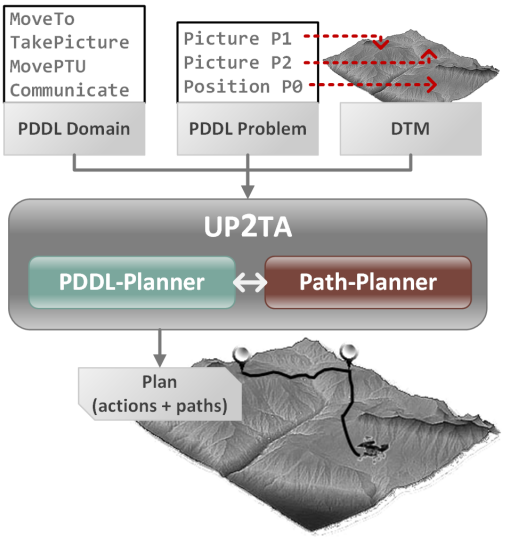


Figure 6. UP2TA general structure.

initial position `robot_at C1_1` to the waypoint `C9_5`. Next, the pan-tilt unit has to point at the angle `P30_20` (action `Move_PTU` not shown in fig. 5 (a)), that is *pitch* = 30° and *yaw* = 20°. Then, the robot can take the picture (action `Take_Picture`). This action sequence achieves goal T_1. In general, the robot should perform a `Move_To` and a `Perform_Task` actions to reach a goal T_i. In this example, each task is performed at a particular position. Therefore, T_i denotes the task and the location. For instance, T_1 is located at `C9_5`, which corresponds to coordinates x=9 and y=5 in the DTM. Then, we need to plan for the second goal T_2 = `drill C6_8`, and finally the third goal T_3 = `robot_at C1_1`. Note that the robot ends at the same position where it starts.

As mentioned previously, the order in which we achieve the goals has a significant impact in the quality solution due to the total distance travelled. To optimally order the tasks, we have modified the task-planner search algorithm to merge its heuristic with the path-planner heuristic. For UP2TA, we use FF as the task-planner and S-Theta* as the path-planner algorithm. For the first one, we have transformed the BFS algorithm into an A* algorithm. The only difference between BFS and A* is that the latter includes the heuristic computation in the node evaluation as is shown in eq. 1. In addition, we have disabled the Hill Climbing search to avoid local minimums. For the path-planner, we do not need to perform any modification. As we discuss later, the path-planner algorithm is easy to replace with a different one.

The search begins when UP2TA takes each task T_i that can be performed from the initial position (*start*) defined in the problem (denoted as `robot_at` in the example). It calculates the task-planner heuristic, $h_{FF}(T_i)$, and the one provided by the path-planning algorithm, $dist(start, T_i)$. The $h_{FF}(T_i)$ heuristic, as mentioned in sec. 3, is a domain-independent heuristic that computes an estimation of the actions required to accomplish a goal. The $dist(start, T_i)$ heuristic is computed by the path-planning algorithm, but it is only required when the task-planner needs to instantiate an action `Move_To`. In general, this heuristic is computed as the Euclidean distance, but others can be used. The way in which we have merged both heuristics in UP2TA is by adding them as in eq. 3, where T_j = *start* at the beginning of the search. By doing this UP2TA can decide which goals to reach first based on the distance and not only in the number of actions required to accomplish it.

$$H(T_i) = h_{FF}(T_i) + dist(T_j, T_i) \quad (3)$$

The reason to combine the heuristics of the task-planner and the path-planner is because of the way FF performs the search. If h_{FF} is equal to 0, it means that the action under evaluation has reached the goal, and FF returns the plan. However, if the last action is a movement action, we need to consider the distance before returning the plan. That is the reason why we include the path-planning heuristic in $H(T_i)$.

Once the heuristic value is calculated, we need to compute the $G(T_i)$ value. That is, the cost associated to reach task T_i from the previous task. If a movement action is required, we need to compute the distance between the two tasks. This step plays a fundamental role in the search process because obtaining a close to optimal ordering of the task depends on the environment. It is unusual to transverse routes between two points in straight-line because of the presence of obstacles or terrain features that make the path more costly. This is specially problematic when there are tasks that are close to each other, or with high number of obstacles in the map. In these cases, the real path length may be too far from the heuristic value and the selection of the actions to perform first depends on the presence of obstacles. Therefore, it seems to be a good idea to compute the real path when dealing with a pair of tasks. However, when we deal with a higher number of tasks, computing the distance between them requires a higher computational effort. The solution adopted here is to use a greedy path-planning algorithm (like the ones described in sec. 2) to calculate estimations of the path length between a pair of tasks. The idea behind these algorithms is to take the straight-line distance between the start and the goal positions and try to follow this line. The probability of expanding the nodes is directly proportional to the proximity to this line, and inversely proportional to the number of blocked cells. This modification implies that fewer nodes will be expanded and, therefore, less memory and time are spent during the search of paths for partial plans.

Although this approximation does not generate the real path, the associated path length allows us to take into consideration the presence of obstacles without spending a lot of time in obtaining paths between tasks that probably will not be used in the final solution. We define $greedy(T_i, T_j)$ as the path cost between tasks T_i and T_j using a greedy path-planning algorithm. In addition, we need to compute the cost of the actions involved in performing a task T_i. This value is represented as $C(T_i)$ and it is given by the task-planner. $G(T_i)$ is a cumulative cost that depends on the cost of reaching the previous task, the distance (estimated) between the two tasks, and the cost of performing a task i as in eq. 4 shows.

$$G(T_i) = greedy(T_i, T_j) + C(T_i) + G(T_j) \quad (4)$$

The task-planner keeps a list of partial plans (where a partial plan is a plan that achieves an arbitrary task T_i) ordered by the $F(T_i)$ values. So, each time that the task-planner extracts a partial plan, it adds it to the list. Then, the search process continues with the extraction of the best promising partial plan. UP2TA generates all possible partial plans starting from the new location where the robot is located. The task-planner continues extracting partial plans until there are no more goals (all objectives are accomplished), or there is no solution (unreachable task/s). When all task are performed, UP2TA does not give the solution directly. The paths computed between each pair of consecutive tasks are processed using suboptimal algorithm (the greedy ones). So, the system takes each pair of tasks and requests the path-planning algorithm for the real path (using the S-Theta* algorithm). Fig. 7 shows the scheme integration between the PDDL planner and the path-planner. The task-planner is in charge of ordering the tasks and obtaining a plan, while the path-planning module provides an estimation of the cost between partial plans and the routes for the final solution.

Fig. 8 shows an example for the search process of the UP2TA planner for a problem with three goals. There are three possible tasks T_1, T_2, and T_3 at the *start* position. The first step is to calculate the heuristic and cost values $H(T_i)$ and $G(T_i)$ for each task. The heuristic $H(T_i)$ is calculated using eq. 3 and it corresponds to the Euclidean distance between the task and the current position plus the heuristic given by the task-planner, i.e., the estimated number of actions. The cost $G(T_i)$ of achieving the position of each task is calculated using eq. 4. Once $H(T_i)$ and $G(T_i)$ are calculated, there are three partial plans to be evaluated, one for each task T_i.

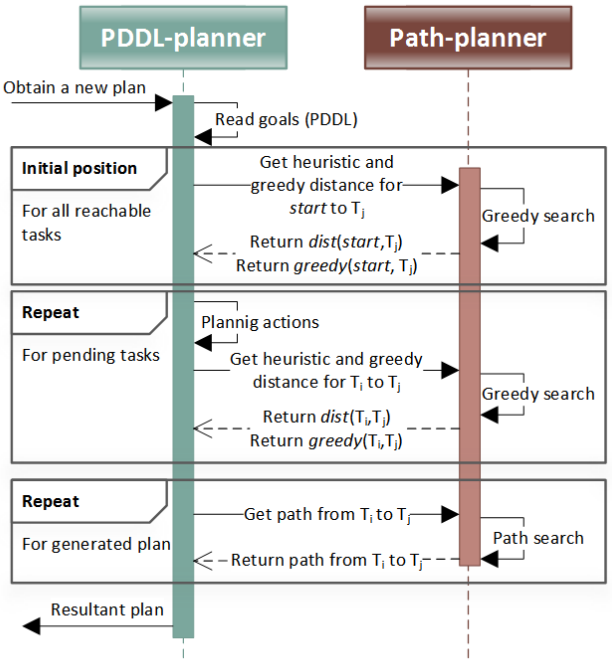


Figure 7. UP2TA algorithm integration.

UP2TA takes the one with lower $F(T_i)$ value. In our example, it is the partial plan that performs T_1. Next, it evaluates the other two possible tasks T_2 and T_3. In the same way, the system calculates the heuristic and cost to reach T_2 and T_3 from T_1, which in our case T_2 has the lower value. At this point, there is only one remaining task T_3, so the search finishes by adding that task. Once all tasks have been selected, the ordered tasks sequence for the solution is $T_1 \rightarrow T_2 \rightarrow T_3$. Fig. 8 shows the most promising partial plans connected through a bold line (others have dotted lines). Then, UP2TA will calculate the real path between each pair of consecutive tasks that have been previously ordered. Finally, the system provides an ordered list of tasks and the paths between them.

6. Mobile robot exploration domains

In this section, we describe a scenario that we use in our experimental evaluation. It consists of a mobile robot that must achieve a set of exploration tasks in different locations. As explained in sec. 5.1, we must provide three files to UP2TA: the PDDL domain and problem, and the terrain information (DTM). The PDDL domain was partially presented in fig. 5. It describes robot operations to manage the different subsystems (power on/off), move between points (`Move_To`), and perform exploration tasks (`Take_Picture`, `Drill`, etc.) to accomplish mission goals. The robot is modelled to have different subsystems. In particular, it has a scientific/navigation camera mounted on top of a pan-tilt unit, a navigation subsystem, and drilling equipment. Each subsystem can be *on* or *off* (must be *on* before operation). In addition, there are some constraints that must be considered when operating the different subsystems. Some of these constraints (shown in fig. 5 (a)) are presented in the `Move_To`, `Take_Picture`, and `Drill` acquisition operators. Concretely:

- In order to move between waypoints, the rover must power on the navigation subsystem (`gnc`) and point the pan-tilt unit at the front position (represented as `P0_0`) since the navigation camera is mounted on it. The other subsystems must be *off* to avoid mechanical problems and energy overconsumption.
- In order to take a picture, the rover must be stopped in the desired location and with a specific pan-tilt position, which are defined in the goal. The camera must be active and the locomotion subsystem must be *off*.
- In order to get a sample from a desired location, the rover must use the drilling equipment to perforate the subsurface. To do this, the rover must be stopped in the desired position and the pan-tilt unit must point at the surface – this allows the rover to take pictures during the drilling process.

Initially, all rover's subsystems are off with the pan-tilt unit pointing to the front. The DTM, or terrain information, is in a different file. Particularly, the one that we used within our

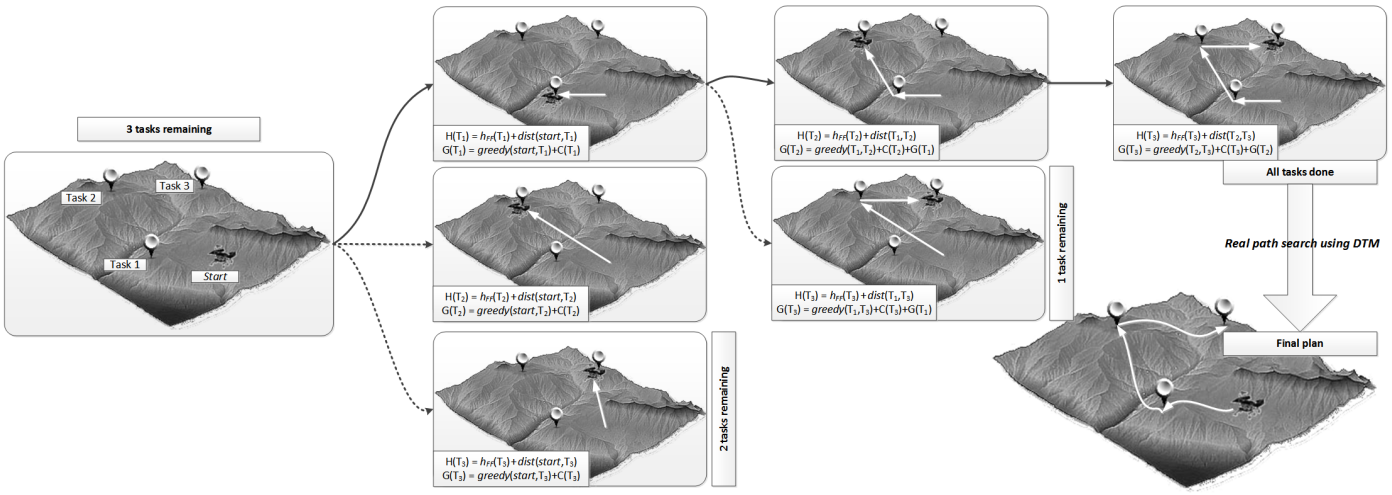


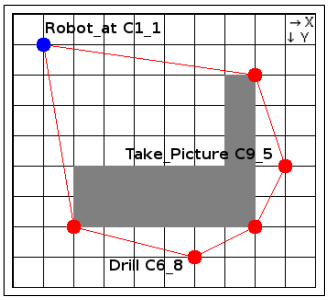
Figure 8. Representation of the UP2TA expansion process for a problem with three tasks.

path-planning algorithms provides a grid with blocked and unblocked cells (see fig. 9 (b)). Given these data, UP2TA generates the solution shown in fig. 9 (a). It is remarkable that the path-planner expands the move actions between tasks to provide free collision paths, while the task-planner orders the goals to minimize the distance travelled for the final plan.

For this domain, each task can be performed independently. That is, there is no relationship between tasks except for the robot position. For this reason, we have also defined a more challenging domain where tasks interfere with each other. This new domain is an extension of the previous one that includes extra requirements for the drilling process: it can only take one sample at a time, and it is necessary to deliver each sample before getting a new one. The delivering process implies going to a particular location and execute a deliver action. Once the robot delivers the sample, it can get another one. For this domain, the planner has to take into consideration the limited capacity of the drill to order the goals. The drill samples must be gotten one by one, but the pictures can be taken independently. This means that the planner must try to optimize the pictures acquisition interleaving them with the drilling/delivering operations, when possible.

```
Move_To C2_7
Move_To C6_8
Move_PTU P0_40
Drill C6_8
Move_PTU P0_0
Move_To C8_7
Move_To C9_5
Move_PTU P30_20
Take_Picture C9_5 P30_20
Move_PTU P0_0
Move_To C8_2
Move_To C1_1
```

a) Plan given by UP2TA



b) Plan graphical representation

Figure 9. Solution for the problem of fig. 5 (b).

7. Experimental Results

In the following, we present an experimental evaluation on the mobile robot exploration domains described in the previous section. The test consists of running the UP2TA planner with different path-planning algorithms (the algorithms use the same methods and structures to manage the DTM) over 100 uniforms flat surfaces of 500x500 nodes with 40% of blocked cells (the map generation algorithm can be found in Muñoz et al. [37]) and randomly placed tasks. In addition, we try different heuristic configurations for both, task-planning and path-planning. For each test, we measure the three parameters: (i) the runtime for the planning process, (ii) the total distance travelled to achieve all goals, and (iii) the total turns in degrees for the final path. The experiments were performed with a time limit of 2 minutes for each execution. The execution was conducted on a 2.0 GHz Intel Core i7 with 4 GB of RAM under Ubuntu 12.04 (64 bits).

We have performed two sets of experiments. In the first one, we use the domain shown in fig. 5 (b) and different path-planning heuristics combination. In the second one, we use the extended version of the exploration domain (with samples delivering and unitary capacity for the drill) in which we have substituted the drill operation with a collect and deliver task to analyse the importance of the task-planner heuristic. In general, the objective is to evaluate the best configuration of UP2TA in terms of runtime and distance travelled (modifying the path-planning algorithm), and to compare it with the results obtained with approaches that do not share information between the path-planner and the task-planner during the search.

For the first test (fig. 10), we want to analyse how the path-planner heuristic affects the tasks ordering by the task-planner. We are also interested in analysing the paths generated to select the best performance configuration as function of the distance travelled. The way that we do this is by testing three different cost functions, $F(T_i)=[H(T_i)] +[G(T_i)]$, for the integration of task-planning and path-planning:

1. We only use the FF heuristic and the cost is computed using the greedy path-planning algorithm as shown in eq. 5. This technique does not share information between the task-planner and the path-planner. That is, it provides solutions in a similar way that the solution proposed in sec. 4 in which planners are interleaved without sharing information (see fig. 4 (b)).

$$\begin{aligned} \mathrm{F}(T_i) &= [h_{FF}(\mathrm{T}_i)] + \\ &\quad [\mathrm{G}(\mathrm{T}_j) + greedy(\mathrm{T}_i, \mathrm{T}_j) + \mathrm{C}(\mathrm{T}_i)] \end{aligned} \tag{5}$$

2. We use the FF heuristic and the euclidean distance as the task-planner heuristic. The cost is computed using a normal path-planning algorithm as show in eq. 6.

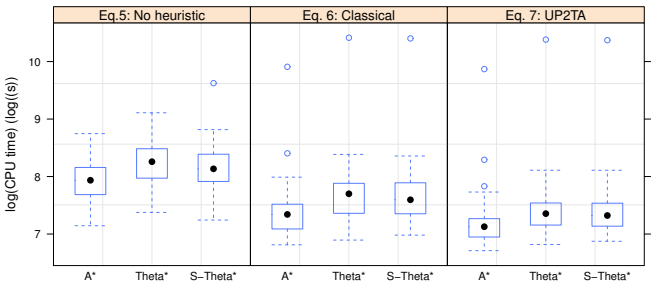
$$\begin{aligned} \mathrm{F}(T_i) &= [h_{FF}(\mathrm{T}_i) + dist(\mathrm{T}_i, \mathrm{T}_j)] + \\ &\quad [\mathrm{G}(\mathrm{T}_j) + path(\mathrm{T}_i, \mathrm{T}_j) + \mathrm{C}(\mathrm{T}_i)] \end{aligned} \tag{6}$$

3. We use the FF heuristic and the Euclidean distance as the task-planner heuristic. The cost is computed using a greedy path-planning algorithm as show in eq. 7.

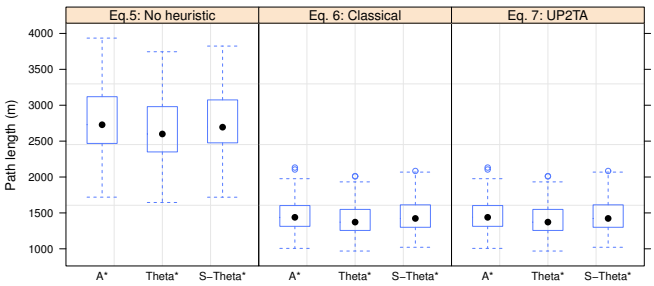
$$\begin{aligned} F(T_i) &= [h_{FF}(\mathrm{T}_i) + dist(\mathrm{T}_i, \mathrm{T}_j)] + \\ &\quad [G(\mathrm{T}_j) + greedy(\mathrm{T}_i, \mathrm{T}_j) + C(\mathrm{T}_i)] \end{aligned} \tag{7}$$

Fig. 10 shows the results for the above mentioned approaches (fig. 11 shows solution examples). For each case, nine tasks will be executed. As we can see, when we only exploit the FF heuristic we obtain the worst results for all parameters. That is especially relevant in the case of the distance travelled. The reason why we obtain these results is that we do not provide any information about the distance in the heuristic, and then, the task-planner will only take into consideration the number of pending tasks. This may lead to connect tasks that are located far from each other. When we include the path-planning heuristic (eq. 6 and 7), we get the same results in path length and heading changes. The difference resides in the runtime: when we compute the path length using a non-modified path-planning algorithm, the planner requires near 20-30% more time to generate a solution. In eq. 7 we use the greedy (or the modified) path-planning algorithm that is faster since it provides an estimation of the path cost. Then, it requires less execution time. Therefore, the best configuration is the one that uses the evaluation function presented in eq. 7, being the one adopted for the UP2TA deployment. In the case of the path-planning algorithm, A* provides the longer paths and the highest heading changes, while Theta* provides the best paths. However, S-Theta* obtains the best results in heading changes, and, also, the best paths when combining path length and heading changes. Thus, in the following we adopt S-Theta* as the path-planner for UP2TA.

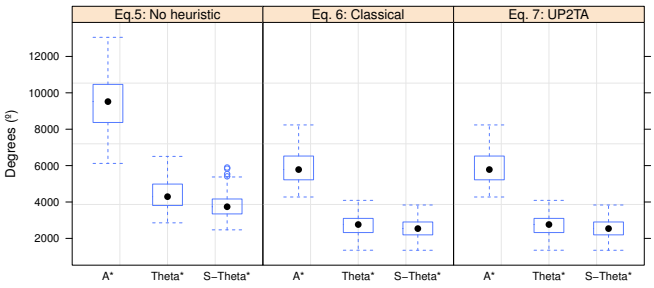
For the second test (fig. 12), we have used the extended PDDL domain in which we have substituted 1/3 of the problem's goals by a collect and deliver task that requires 4 actions:



a) Runtime in seconds



b) Path length in meters



c) Heading changes in degrees

Figure 10. How the path-planner affects the task ordering. Solutions of 100 pictures acquisition problems with maps of 500x500 nodes with 40% of obstacles and 9 randomly placed pictures. Results are clustered by the path-planning algorithm (A*, Theta* and S-Theta*) and the F function used (from left to right): no path-planning heuristic (eq. 5); cost function using classical path-planning algorithms (eq. 6); cost function using greedy path-planning algorithm (eq. 7).

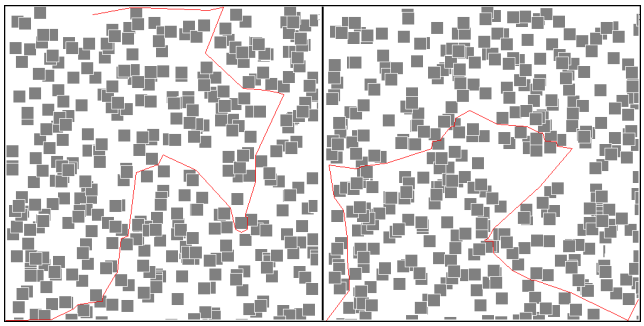


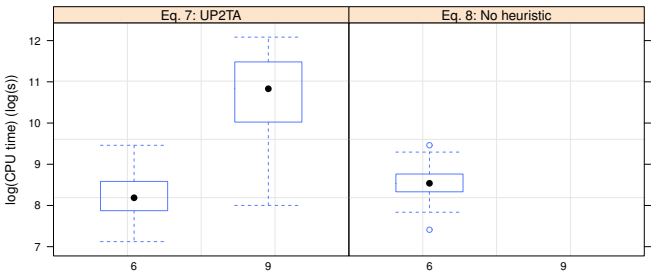
Figure 11. Generated paths for 9 pictures in two 500x500 maps with 40% blocked cells using UP2TA with S-Theta* for path-planning.

(i) move to a location, (ii) collect a sample, (iii) move to the delivering location, and (iv) deliver the sample. In our test, the system has unary capacity, which means that the current sample should be delivered before getting a new one. This implies that the planner should take such constraint into consideration

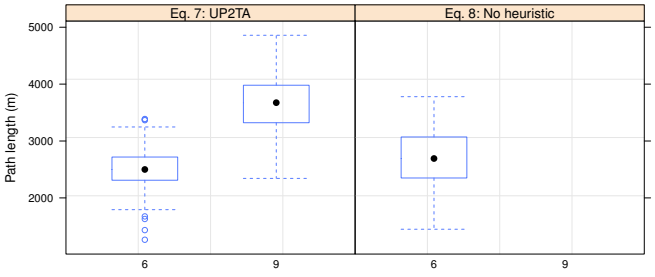
to order the goals in the best way possible. This is done by the task-planner. There two problems for this domain, which consist of delivering two and three samples and taking 4 and 6 pictures, being 6 and 9 the total number of tasks respectively. The rest of parameters are the same as the previous experiment – 100 maps of 500x500 nodes with 40% of obstacles. To perform this test we use the UP2TA planner with eq. 7 to compute $F(T_i)$. In order to analyse the impact of the task-planner heuristic, we have removed the FF heuristic and used the $F(T_i)$ shown in eq. 8. In any case, we use S-Theta* as the path-planner.

$$F(T_i) = [dist(T_i, T_j)] + [G(T_j) + greedy(T_i, T_j) + C(T_i)] \quad (8)$$

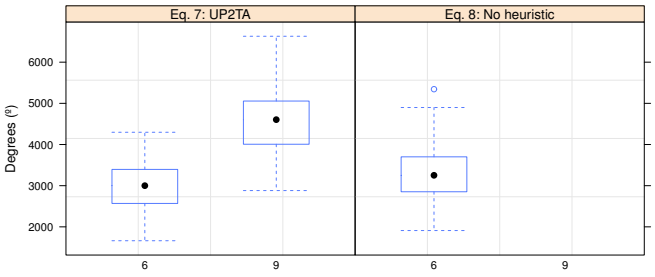
Fig. 12 shows the results for the second test. We can see that when we remove the h_{FF} heuristic, the system is not able to find any solution for the 9 tasks case within the given time limit. Moreover, all parameters are worse than using UP2TA with $F(T_i)$ computed as in eq. 7. Conversely, UP2TA solves the 6 tasks and 9 tasks problems within 6 and 60 seconds respectively. The problem of removing the h_{FF} heuristic is similar to removing the path-planning heuristic in the previous experiment. In this extended domain, the planner without the h_{FF} heuristic only tries to follow the shortest path between tasks. However, the problem is when a task has dependencies with other tasks. In that case, the system behaves like a breadth search algorithm, where the search space grows uncontrolled as a function of the number of tasks. The planner is not able to properly manage the tasks ordering to accomplish first the ones required to complete others actions. This generates undesirable behaviours such as movements to a location in which we want to take a sample when the previous sample is not delivered, delaying the solution of the problem.



a) Runtime in seconds



b) Path length in meters



c) Heading changes in degrees

Figure 12. How the task-planner heuristic affects the solutions. Results for 100 sample and deliver problems with maps of 500x500 nodes with 40% of obstacles. Parameters clustered by the number of tasks (6 and 9) and the F function used (from left to right): cost function using greedy path-planning algorithm (eq. 7) and; no task-planning heuristic (eq. 8). In all cases employing S-Theta* as path-planning algorithm.

8. Robotic platform deployment tests

In this section, we present a demonstration of UP2TA as the deliberative layer of an autonomous controller called MoBAr [3, 4]. This controller is a 3-layer architecture formed by a functional layer supported on ROS [38], an executive based on PLan EXecution Interchange Language (PLEXIL) [39], and a PDDL based deliberative, which is UP2TA in this case. UP2TA is in charge of generating a plan to achieve a set of goals (e.g., pictures or samples acquisition for the domains used in our experimentation). The plan is executed and monitored by the PLEXIL executive, which decomposes each action into low level functional commands. Finally, the functional layer is connected to the robot hardware and provides an interface for its operation.

The robot used for testing UP2TA with MoBAr is a modified TurtleBot 2 (see fig. 13). The default TurtleBot robot provides a mobility subsystem, which is based on two motorized wheels with integrated encoders and its electronic support to provide odometry. It has a maximum lineal velocity of 70 cm/s and rotational of 110 deg/s. Also, it has three infra-red sensors and a frontal collision detector. It is typically used with an attached Microsoft Kinect camera, which provides optic and depth field images. In our robot, we have mounted the Kinect on top of the robot with a pan-tilt (formed by two MG996R servo motors) to enable camera pointing. Our robot is controlled using an embedded PC endowed with a dual core Intel Atom processor, 2GB of RAM with Ubuntu 14.04 and ROS Indigo. Also, we have added an Arduino board to control the pan-tilt unit and other sensors that will be added for specific applications. The battery enclosed in the robot base has a capacity of 4.4Ah. The total mass including the control hardware is near 7 Kg with a dimension of 35 x 35 x 55 cm.

We have performed an experimental evaluation using TurtleBot in two different scenarios. The first one consists of acquiring six pictures using the mobile robot exploration domain. The second one consists of acquiring three pictures, and getting and delivering a sample. We have performed each scenario three times to gather average values of the execution. In addition, we have employed UP2TA and two more deliberatives following the schema of fig. 4 (b). In the first one, we have combined SGPlan

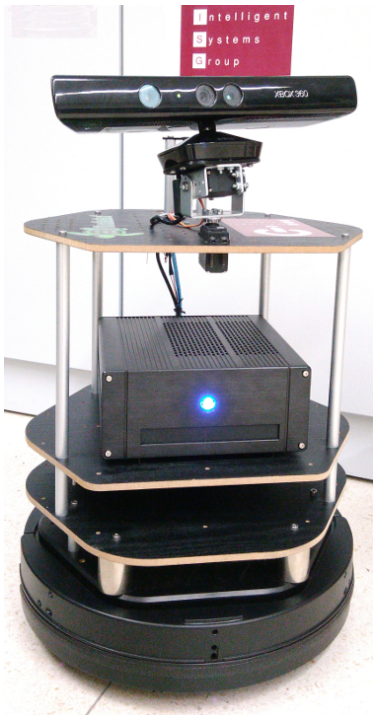


Figure 13. The TurtleBot II used in the experiments.

and S-Theta* (we have called this system SGPlan$_{S\theta}$). And in the second one, we have used OPTIC with S-Theta* (we have called it OPTIC$_{S\theta}$). The floor has a dimension of 7.6 x 6.8 m, modelled as 19 x 17 cells (each cell is 0.4 x 0.4 meters). We report the *search time* to generate a plan (in milliseconds), the *execution time* employed by the robot to complete the plan (in seconds), the planned *path* length (in meters), the distance travelled measured by the *odometry* sensors of the robot (in meters) and the total *turns* of the path (degrees). In both domains, all planners provide a good estimation of the distance that the robot should travel with less than half a meter of error between the planned distance and the real travelled distance measured using odometry (columns 3 and 4 of tables 2 and 3).

Table 2 shows the results for the pictures acquisition scenario. The data shows that UP2TA obtains better results that the interleaving approach in terms of path length and execution time, but at the expense of a larger deliberation time. We can see that SGPlan$_{S\theta}$ and OPTIC$_{S\theta}$ generate two times longer paths than UP2TA; and SGPlan$_{S\theta}$ and OPTIC$_{S\theta}$ perform similarly. However, it is remarkable that the path provided by SGPlan$_{S\theta}$ requires more heading changes, slightly increasing the execution time. Fig. 14 shows the location of the six taking pictures goals and the planned paths for each planner. The dots represent points in which the robot performs a heading change. The initial position of the robot is represented in the map by the S label. The #P labels identify the positions of the taking picture goals, preceded by a number that represents the order in the plan. We can see that UP2TA provides paths easier to follow compared to the ones generated by SGPlan$_{S\theta}$ and OPTIC$_{S\theta}$. These two systems provide paths that traverse from south to north and vice-versa several times. This increases the distance travelled

Table 2. Parameters measured for 6 pictures acquisition scenario.

Planner	Search time (ms)	Exec. time (s)	Path (m)	Odometry (m)	Turn (deg)
SGPlan$_{S\theta}$	47	395	23.44	23.16	527
OPTIC$_{S\theta}$	68	370	23.57	23.32	428
UP2TA	308	251	11.10	11.41	534

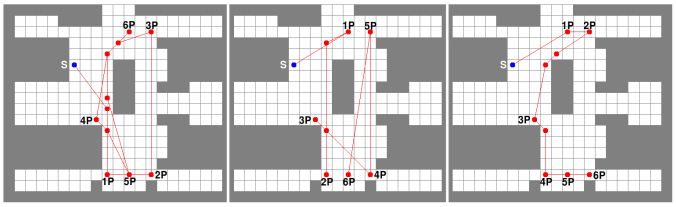


Figure 14. Planned path for 6 pictures acquisition scenario for planners (from left to right): SGPlan$_{S\theta}$, OPTIC$_{S\theta}$ and UP2TA. **#P** identifies the order in which pictures are taken. **S** denotes the start position.

and, therefore, the execution time. If we analyse the routes, the path generated by SGPlan$_{S\theta}$ first acquires the picture on the bottom left (labelled as 1P) and then traverses to the bottom-right where it acquires the second picture (2P). However, it would be better to complete task numbered as 5P than 2P, which is in the path. In the same way, in the traverse from the start position to the fist picture there is a goal in the middle of the map (4P) that can be acquired, but it is achieved later on. OPTIC$_{S\theta}$ has a similar behaviour, traversing from south to north several times. Instead, UP2TA acquires first the picture on the top-middle (1P) of the map followed by the one at the top-right (2P). Then, it goes to the bottom of the map, but acquiring the picture in the middle (3P) before reaching the south side. Then, it takes the last pictures of the bottom from left to right (4P, 5P and 6P).

For the sample and delivering scenario, the results are presented in table 3. In this scenario UP2TA requires more time to achieve a plan than SGPlan$_{S\theta}$ or OPTIC$_{S\theta}$. However, the length of the path is nearly 33% shorter than the one generated by OPTIC$_{S\theta}$. Also, OPTIC$_{S\theta}$ outperforms SGPlan$_{S\theta}$ in distance and execution time. The paths generated can be seen in fig. 15. In the same way as the previous scenario, paths are represented by lines and heading changes as dots. In this case, there are three labels: P for Pictures, T for Taking a sample and D for Delivering a sample (the deliver point is on top of the start position, S). If we analyse the paths followed by SGPlan$_{S\theta}$ or OPTIC$_{S\theta}$, we can observe that, usually, they do not interleave pictures acquisition with samples collecting/delivering. For example, SGPlan$_{S\theta}$ first takes the sample at the bottom-left (1T) and then delivers it (2D). Then it acquires the sample at the bottom-right (3T) and delivers it (4D). However, the picture at the bottom-middle (8P) or the one in the middle of the map (7P) are delayed to the end of the plan. This issue results in longer paths and suboptimal solutions. In a similar way, OPTIC$_{S\theta}$ does not properly interleave samples acquisition/delivering with pictures acquisition. It first acquires the picture at the top-middle (1P) and then acquires the sample at the bottom-left (2T). It could have been a better decision to take the sample on the right of the first taking picture (7T), but instead, it has decided to

Table 3. Parameters measured for 3 pictures acquisition and 3 samples delivering scenario.

Planner	Search time (ms)	Exec. time (s)	Path (m)	Odometry (m)	Turn (deg)
SGPlan$_{S\theta}$	72	672	41.21	41.71	814
OPTIC$_{S\theta}$	138	582	34.93	35.29	766
UP2TA	5041	480	26.55	26.82	615

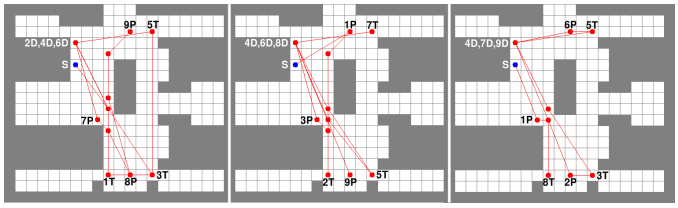


Figure 15. Planned path for 3 pictures and 3 samples delivering scenario for planners (from left to right): SGPlan$_{S\theta}$, OPTIC$_{S\theta}$ and UP2TA. Actions in sequential order: **P**=picture; **T**=take sample; **D**=deliver sample.

move to the south. Then during the traverse to deliver the sample (4D), it acquires the picture at the middle of the map (3P). Then, it acquires the sample at bottom-right (5T) and delivers it (6D). Again, it delays the action of taking a picture close to other goal to the end of the plan (9P). This is a consequence of the domain independent heuristic used by the PDDL planners that do not properly consider the distance travelled during the planning process, delaying goals even when the robot traverses nearby them. In the case of UP2TA, it performs picture acquisitions while moving to the sample acquisition goals or to the delivery position. It starts moving to the south of the map, acquiring the picture in the middle (1P) and then the one in the bottom-middle (2P). After, it takes the sample at the bottom-right (3T) and delivers it (4D). The next goal is the sample at the top-right (5T) but, before delivering it (7D) it acquires a picture (6P). At this point, all pictures goals are achieved and there is only one task sample remaining. UP2TA takes the sample (8T) and delivers it (9D). We can conclude that sharing information between the path-planner and task-planner allows interleaving goals that are close to each other, which results in better solutions.

9. Conclusions

In this paper, we presented the UP2TA planner that integrates task-planning using a PDDL based planner and a path-planning algorithm to obtain feasible paths. The motivation of this new planner is to create a deliberative layer for autonomous robots that can interleave task-planning and path-planning. The aim is to obtain a closer optimal ordering of tasks taking into consideration the path between them. A PDDL model could manage both objectives, but only with small grids and spending several time managing the input files. Another possible approach is to loosely couple task and path-planning interleaving them in different planning phases. This approach provides some advantages, but still lacks of optimality in the solution generated.

Our approach, however, interleaves task-planning and path-planning during the search, sharing information among planners. This allows using domain-dependent heuristics for path-planning, and domain-independent heuristics for task-planning. Using this technique UP2TA can take advantage of each planner features and deal with more complex scenarios ensuring a good performance and an improved quality of the plan generated. The new planner also separates the path search and the task ordering, which means that the task-planner does not need to manage the terrain information. This implies that we can exploit highly specialized path-planning algorithms suitable for different situations to improve the performance and quality of the plans.

We performed an experimental evaluation of the UP2TA planner in two mobile robot exploration domains. The first domain is based on picture acquisition with tasks being independent of each other. The second domain is based on samples delivering with a limited sample capacity. This constraint makes the problem more challenging and costly to optimally solve it. Both domains were tested in automated benchmarks and a real robot platform. Results showed that UP2TA provides better solutions than other approaches in which there is no shared information among planners during search.

About the scalability of the new planner, in the paper we have only presented the results for plans with 6 to 12 tasks and 500x500 grids map size. However UP2TA can handle higher number of tasks, although for real applications, this number is generally low because executing higher numbers of tasks is very time consuming. In relation to the size of the maps, UP2TA can manage 20.000x20.000 grids size in a generic computer with 8GB of RAM. Bigger grids can be handled with more powerful machines. Additionally, independently of the number of obstacles (we have considered maps with blocked cells up to 40%, as commonly reported in the literature), if there exist a path between two points, the path-planning algorithms used will find it.

Acknowledgments

Pablo Muñoz is supported by the European Space Agency under the Networking and Partnering Initiative *Cooperative systems for autonomous exploration missions* project 4000106544 /12/NL/PA. This work was partially supported by UAH project 2015/00297/001, MINECO EphemeCH TIN2014-56494-C4-4-P and Junta de Comunidades de Castilla-La Mancha project PEII2014-015-A. Authors want to thank Héctor Franco Gregorio and Diego López Pajares for their contributions on the UP2TA and TurtleBot implementation and the reviewers for their valuable comments.

References

[1] Heutger, M. and Kückelhaus, M. Self-driving vehicles in logistics. Tech. report, 2014.

[2] McDermott, D. The PDDL Planning Domain Definition Language. The AIPS-98 Planning Competition Committee, 1998.

[3] Muñoz, P., R-Moreno, M. D. and Martínez, A. A first approach for the autonomy of the Exomars rover using a 3-Tier architecture, In Procs. of the 11th ESA Symposium on Advanced Space Technologies for Robotics and Automation. Noordwijk, The Netherlands, 2011.

[4] Muñoz, P. and R-Moreno, M. D. Model-Based Architecture on the ESA 3DROV simulator, In Procs. of the Application Showcase, 23rd International Conference on Automated Planning and Scheduling. Rome, Italy, 2013.

[5] Yap, P. Grid-Based Path-Finding, Advances in Artificial Intelligence, vol. 2338, pp. 44–55, Lecture Notes in Computer Science, Springer Berlin, 2002.

[6] Hart, P., Nilsson, N. J. and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, vol. 4, pp. 100–107, 1968.

[7] Daniel, K., Nash. A., Koening, S. and Felner, A. Theta*: Any-Angle Path Planning on Grids. Journal of Artificial Intelligence Research, vol. 39, pp. 533–579, 2010.

[8] Botea, A., Muller, M. and Schaeffer, J. Near Optimal Hierarchical Path-Finding. Journal of Game Development, vol. 1(1), pp. 1–22, 2004.

[9] Ferguson, D. and Stentz, A. Field D*: An Interpolation-based Path Planner and Replanner, In Procs. of the 12th International Symposium on Robotics Research (ISRR), San Francisco, CA, USA, 2005.

[10] Nash, A., Koenig, S. and Likhachev, M. Incremental Phi*: Incremental Any-Angle Path Planning on Grids. In Procs. of the International Joint Conference on Artificial Intelligence (IJCAI), Pasadena, CA, USA, 2009.

[11] Nash, A., Koenig, S. and Tovey, C. Lazy Theta*: Any-Angle Path Planning and Path Length Analysis in 3D. In Procs. of the 24th AAAI Conference on Artificial Intelligence (AAAI), Atlanta, GE, USA, 2010.

[12] Muñoz, P. and R-Moreno, M. D. S-Theta*: low steering path-planning algorithm. In Procs. of the 32nd SGAI International Conference on Artificial Intelligence, Cambridge, UK, 2012.

[13] Muñoz, P. and R-Moreno, M. D. Improving Efficiency in Any-Angle Path-Planning Algorithms. In Procs. of the 6th IEEE International Conference on Intelligent Systems, Sofia, Bulgaria, 2012.

[14] Bonet, B. and Geffner, H. Planning as Heuristic Search: New results. In Procs. of the 1999 European Conference on Planning (ECP-99), Durham, UK, 1999.

[15] Haslum, P. and Geffner, H. Admissible Heuristics for Optimal Planning. In Procs. of the of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS). Breckenridge, Colorado, USA, 2000

[16] Haslum, P., Bonet, B., and Geffner, H. New Admissible Heuristics for Domain-Independent Planning. In Procs. of the of the 19th AAAI Conference on Artificial Intelligence (AAAI). Pittsburgh, Pennsylvania, USA, 2005

[17] Hoffmann, J. and Nebel, B. The FF Planning System: Fast Plan Generation Through Heuristic Search. Journal of Artificial Intelligence Research, vol. 14 pp. 253–302, 2001.

[18] Helmert, M. The Fast Downward planning system. Journal of Artificial Intelligence Research, vol. 26, pp. 191–246, 2006.

[19] Richter, S. and Helmert, M. Preferred operators and deferred evaluation in satisfying planning. In Procs. of the 9th International Conference on Automated Planning and Scheduling (ICAPS), Thessaloniki, Greece, 2009.

[20] Hoffmann, J., Porteous, J. and Sebastia, L. Ordered landmarks in planning. Journal of Artificial Intelligence Research, vol. 22, pp. 215–278, 2004.

[21] Richter, S. Landmark-Based Heuristics and Search Control for Automated Planning. PhD thesis at the Institute of Intelligent and Integrated Systems. Faculty of Science, Environment, Engineering and Technology. Grith University, Brisbane, Australia. November 2010,

[22] Richter, S. and Westphal, M. The LAMA planner: guiding cost-based anytime planning with landmarks. Journal of Artificial Intelligence Research, vol. 39, pp. 127–177, 2010.

[23] Benton, J., Coles, A. and Coles, A. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In Procs. of the International Conference on Automated Planning and Scheduling. Atibaia, So Paulo, Brazil, 2012.

[24] Fox, M. and Long, D. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research, vol. 20, pp. 61–124, 2003.

[25] Gerevini, A. and Long, D. Plan Constraints and Preferences in PDDL3, The Language of the Fifth International Planning Competition, Italy, 2005.

[26] Hoffmann, J. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. Journal of Artificial Intelligence Research, vol. 20 pp. 291–341, 2003.

[27] Hsu, C.W. and Wah, B.W.. The SGPlan Planning System in IPC-6. In Procs. of the 6th International Planning Competition (IPC), 2008, September, Sydney, Australia.

[28] Zacharias, F., Borst, C. and Hirzinger G. Bridging the Gap Between Task Planning and Path Planning. of the 2006 IEEE International Conference on Robotics and Systems (IROS), Beijing, China, 2006.

[29] Cambon, S., Alami, R., and Gravot, F. A Hybrid Approach to Intricate Motion, Manipulation and Task Planning. International Journal of Robotics Research, vol. 28(1), pp. 104–126, 2009.

[30] Hauser, K. and Latombe, J. Integrating Task and PRM Motion Planning: Dealing with Many Infeasible Motion Planning Queries. In Procs. of the 9th International Conference on Automated Planning and Scheduling (ICAPS), Thessaloniki, Greece, 2009.

[31] Akinc, M., Bekris, K.E., Chen, B.Y., Ladd, A.M., Plaku, E. and Kavraki, L.E. Probabilistic Roadmaps of Trees for Parallel Computation of Multple Query Roadmaps. In Algorithmic Foundations of Robotics VI, Springer Tracts in Advanced Robotics, vol. 14, pp. 80–89, 2005.

[32] Wolfe, J., Marthi, B. and Russell, S. Combined Task and Motion Planning for Mobile Manipulation. In Procs. of the 10th International Conference on Automated Planning and Scheduling (ICAPS), Toronto, Canada, 2010.

[33] laValle, S.M. and Kuffner, J.J. Rapidly-exploring Random Trees: Progress and Prospects. In Algorithmic and Computational Robotics: New Directions, pp. 293–308, 2001.

[34] Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V. and Uras T. Combining High-Level Casual Reasoning with Low-Level Geometric Reasoning and Motion Planning for Robotic Manipulation. In Procs. of the 2011 IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, 2011.

[35] Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S. and Abbeel, P. Combined Task and Motion Planning Through an Extensible Planner-Independent Interface Layer. In Procs. of the 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 2014.

[36] Kautz, H., Selman, B. and Hoffman, J. SatPlan: Planning as Satisfiability. Abstracts of the 5th International Planning Competition (IPC-5), hosted at the International Conference on Automated Planning and Scheduling (ICAPS), Cumbria, UK, 2006.

[37] Muñoz, P., Barrero, D. F., and R-Moreno, M. D. A Statistically Rigorous Analysis of 2D Path-Planning Algorithms. The Computer Journal 2014.

[38] Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A.Y. ROS: an Open-source Robot Operating System. In Procs. of the 2009 ICRA Workshop on Open Source Software, Kobe, Japan, 2009.

[39] Dowek, G., Muñoz, C. and Pasareanu, C. A Formal Analysis Framework for PLEXIL. In Procs. of the 3rd Workshop on Planning and Plan Execution for Real-World Systems, Rhode Island, USA, 2007.