PETLON: Planning Efficiently for Task-Level-Optimal Navigation

Shih-Yun Lo
Learning Agent Research Group
The University of Texas at Austin
yunl@cs.utexas.edu

Shiqi Zhang
Department of EECS
Cleveland State University
s.zhang9@csuohio.edu

Peter Stone
Learning Agent Research Group
The University of Texas at Austin
pstone@cs.utexas.edu

ABSTRACT

Intelligent mobile robots have recently become able to operate autonomously in large-scale indoor environments for extended periods of time. Task planning in such environments involves sequencing the robot's high-level goals and subgoals, and typically requires reasoning about the locations of people, rooms, and objects in the environment, and their interactions to achieve a goal. One of the prerequisites for optimal task planning that is often overlooked is having an accurate estimate of the actual distance (or time) a robot needs to navigate from one location to another. State-of-the-art motion planners, though often computationally complex, are designed exactly for this purpose of finding routes through constrained spaces. In this work, we focus on integrating task and motion planning (TMP) to achieve task-level optimal planning for robot navigation while maintaining manageable computational efficiency. To this end, we introduce TMP algorithm PETLON (Planning Efficiently for Task-Level-Optimal Navigation) for everyday service tasks using a mobile robot. PETLON is more efficient than planning approaches that pre-compute motion costs of all possible navigation actions, while still producing plans that are optimal at the task level.

KEYWORDS

Task and motion planning; Navigation; Mobile robots; Service robots

ACM Reference Format:
Shih-Yun Lo, Shiqi Zhang, and Peter Stone. 2018. PETLON: Planning Efficiently for Task-Level-Optimal Navigation. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. Andre, S. Koenig (eds.), Stockholm, Sweden, July 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

"Planning", or selecting a sequence of actions to achieve a goal, has been a core focus of interest within the field of Artificial Intelligence (AI) since the field was founded in the 1950's. Initially, the focus of attention was on *task planning* which is concerned with sequencing actions within a symbolic representation of the state space [7]. For example, *if a robot has the goal of obtaining supplies for camping including milk and frozen hot dogs, both of which could spoil if not refrigerated, a symbolic planner, given a domain model that includes coolers, ice, and conditions for spoiling, could identify that the robot should first buy a cooler, then a block of ice, and then milk and hot dogs*. In general, task planners aim to find the shortest plan in terms of number of symbolic actions. If action costs are available, some task planners are also able to identify the lowest cost plan (which in general may be longer in terms of number of actions).

A key limitation of task planning is that it assumes that symbolic actions can be executed "atomically." Continuing our example, it does not reason about how the robot should traverse continuous space in order to travel from its current location to the store. Rather it assumes that the robot can teleport itself to the next location, perhaps roughly estimating how long it would take to move there in the real world. In contrast, a largely independent thread of research exists on *motion planning* that focuses on producing a continuous motion plan while avoiding collisions with obstacles in 2D or 3D continuous space [24]. Traditionally, motion planning has been concerned with computing a path connecting a start configuration to a goal configuration, without any concern for sequencing of subgoals.

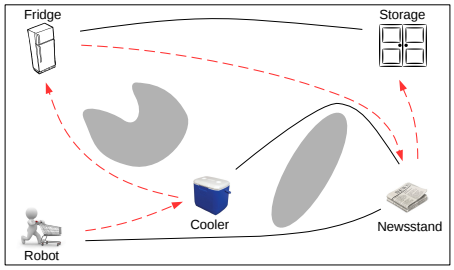
Two Types of TMP Problems. Task and motion planning (TMP) have historically remained mostly (though not entirely — see related work) independent, because physical robots have only been able to execute very short missions that could be solved entirely with motion planning algorithms. TMP for manipulation (TMP-M) has attracted much attention of late, mainly to ensure the geometric feasibility of symbolic plans in highly confined workspaces, with complex kinematic constraints [6, 10, 13, 23, 33]. Despite the great success of these approaches, **TMP for navigation** (TMP-N), i.e., to select task routes considering task-level domain knowledge and navigation costs, presents sufficiently different challenges that different approaches are needed, and has not yet been well addressed in the literature.

Challenges of TMP-N. TMP-N frequently arises in large, knowledge-intensive domains, in which a robot has to reason about many objects and their properties, such as feasible locations to acquire and then to store milk. Moreover, solutions to TMP-N may vary significantly in quality, where suboptimal plans may significantly delay task completion schedule, due to long execution time navigating in large environments.

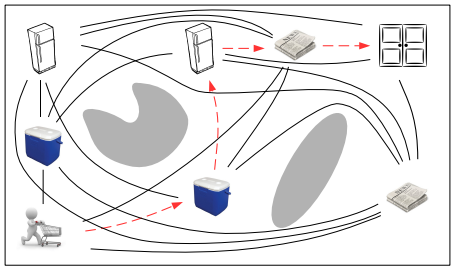
Because of the recent advances in sustainability of long-term autonomy on mobile service robots in large-scale environments [1, 14, 21], there is a pressing need to generate task plans that are fully aware of—and indeed dependent upon—the grounded navigation costs of task actions that can only be determined by motion planning algorithms.

With well-defined physical constraints of symbolic states and a model of the free space and obstacles in the environment, the navigation costs of all possible task actions can be evaluated and then used to select the optimal task route. However, in cases with combinatorially many possible task sequences, doing so can be computationally infeasible.

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. Andre, S. Koenig (eds.),
, July 2018, Stockholm, Sweden.



(a) Small number of objects.



(b) Large number of objects (doubled)

Figure 1: Camping Preparation example: as the number of objects increases, the number of motion cost evaluations increases exponentially.

A Motivating Example: Camping Preparation. Figure 1 illustrates a scenario where a robot needs to prepare items for camping: a hot dog and a newspaper need to be collected and moved to the storage room; and the robot has to have a cooler as the container for hot dogs. In the first setting, to find the optimal task plan, the robot has to evaluate 7 motion costs. In the second setting, the number of cost evaluations grows to 20, even after the task planner rules out plans that do not meet action preconditions (the robot has to collect a cooler before collecting a hot dog from a fridge). Continuing to scale up the number of objects, we can see the prohibitively large number of motion cost evaluations.

The aim of this research is to integrate task and motion planning to select the optimal (lowest-cost) task route for robot navigation in a computationally efficient manner. We introduce a novel algorithm, called Planning Efficiently for Task-Level-Optimal Navigation (PETLON), that returns task-level-optimal solutions while significantly reducing the number of motion cost evaluations. We say a task plan, in the form of a sequence of symbolic actions, is task-level-optimal, if its overall action cost is not higher than that of any task plan, where action costs of the task plans are evaluated using a motion planner. PETLON is a general approach that can work with a variety of task and motion planners. In this paper, we evaluate PETLON with mobile robot delivery tasks both in simulation and on a real robot. It significantly improves planning efficiency in most cases, compared with a competitive, task-level-optimal baseline.

2 RELATED WORK

The integration of task and motion planning has a long history (in some sense, Shakey [30], which executed its plans in the real world, is the earliest example). However, it is not until recently that "integrated task and motion planning" has been used as a term to refer to a family of algorithms that use both task and motion planners. These algorithms have been developed under very different assumptions with very different goals, making direct comparisons a challenge. Here we summarize representative algorithms for this problem.

aSyMov, as one of the earliest algorithms on integrated task and motion planning, bridged the gap between task and motion planners via a set of predefined *roadmaps*, one for each action [2]. The search was conducted only at the motion level within and across different roadmaps (not at the task level), so aSyMov cannot guarantee task-level optimality.

Hierarchical task network (HTN) [29], as a framework for task planning, has been integrated with motion planners. Resulting algorithms include SAHTN [37] and HPN [18]. To improve the HTN-level search efficiency, SAHTN uses an *irrelevance* function to rule out irrelevant domain variables (e.g., objects not blocking the way are not modeled in navigation actions), and HPN uses depth-first traversal and interleaves planning and execution. As a result, SAHTN and HPN can solve extremely long-horizon planning problems. Although SAHTN is a "hierarchically optimal" algorithm, i.e., its optimality is conditioned on the hierarchy, it requires an irrelevance function that is frequently unavailable in practice. More recent work has extended HPN to model the uncertainty in action outcomes and observability [19], but neither the original nor the extended HPN algorithms can guarantee task-level optimality.

Another realization of TMP was achieved via introducing symbolic state constraints at the task level [6], where new constraints are added into the task planner when no feasible kinematic solution can be found for the plan generated by the current task planner. Task-level optimal solutions can be found in that work, *only if* costs of all actions are evaluated at the motion level, which can be prohibitively time-consuming in practice (PETLON is specifically designed to avoid this). In the work of [4], this idea was formalized to achieve probabilistically complete TMP, leveraging the incremental solution capability of Satisfiability Modulo Theories (SMT) [5].

Off-the-shelf task and motion planners can be integrated using a planner-independent interface [33]. Their task planner is optimistic at the very beginning and generates very short plans that are frequently infeasible at the motion level. To update the task planner, their interface needs to *explain* the motion-level failures to the task planners. While their approach is applicable to our navigation domains, failure diagnosis is generally a difficult reasoning problem, especially when a relatively large number of objects and their properties need to be considered.

Learning algorithms have been incorporated to guide the search in task and motion planning problems. For instance, the framework produced in [33] was improved by incorporating reinforcement learning (RL) for plan refinement and learning from expert demonstrations to guide the search at task level [3]. Recent work aims at predicting *solution constraints* to reduce the search space in task and motion planning problems [22]. In that work, a new representation was developed to facilitate the transfer of knowledge (in the form of solution constraints) from one problem instance to another to significantly reduce the search space.

Recently, an algorithm called FFRob has been developed for task and motion planning [10]. FFRob does not require a motion planner, but directly conducts task planning over a set of samples generated

in the configuration space. In order to efficiently search in this large sample space, FFRob extends the FastForwad heuristic [16] to generate a set of heuristics (to reason with geometric constraints) to guide the search. FFRob aims at plans of the shortest length (or "mode-sequence" length in their terms), which is different from our focus. This idea has been further extended to plan in factored transition systems for exposing the topology of their solution space [8], and to reason with possibly infinite sequences of object poses and static predicates [9]

Instead of aiming at a general planning framework, researchers have studied, for instance, rejecting inconsistent task actions for kinematically restricted problems [23], objective optimization [36], and a selective sampling approach for efficient exploration [31]. These algorithms focus on one or more components of the TMP problem, instead of aiming at a general solution.

Although the above algorithms are built on very different assumptions, two observations are shared over all of them: *manipulation* has been the main challenge at the motion level, e.g., grasping and ungrasping, although some involve navigation actions; and none of them guarantees task-level optimality.[1] PETLON is an efficient, task-level-optimal algorithm that is applicable to large-scale robot *navigation* domains that include both low-level (navigational) constraints and high-level (task ordering) constraints.

3 ALGORITHM

Within the context of task and motion planning (TMP) problems, existing research has been conducted with very different assumptions and goals (see Section 2). In this section, we formalize TMP for navigation (TMP-N) problems at task and motion levels, and introduce our PETLON algorithm.

3.1 Planning at Task and Motion Levels

Let $\mathcal{D}^t$ specify a task planning domain that includes a set of states, S, and a set of actions, A. We assume a factored state space such that each state $s \in S$ is defined by the values of a fixed set of variables; each action $a \in A$ is defined by its preconditions and effects. A cost function *Cost* maps the state transition to a real number: $Cost(\langle s, a, s' \rangle) \rightarrow \mathbb{R}$, which represents the cost of action a being executed in state s, as in the MDP setting. [2]

Given domain $\mathcal{D}^t$, a task planning problem is defined by an initial state $s^{init} \in S$ and a specification of the goal that corresponds to a set of goal states $S^G \subseteq S$. A plan, $p \in P$, includes a sequence of transitions that can be represented as: $p = \langle s_0, a_0, \cdots, s_{N-1}, a_{N-1}, s_N \rangle$, where $s_0 = s^{init}, s_N \in S^G$ and P is the set of satisfactory plans.

Solving a task planning problem, using optimal planner $\mathcal{P}^t$, produces plan p^* that is optimal among all satisfiable plans:

$$p^* = \text{argmin}_{p \in P} \Sigma_{\langle s,a,s' \rangle \in p} Cost(\langle s, a, s' \rangle). \quad (1)$$

Let $\mathcal{D}^m$ specify a motion planning domain, where we directly search in the 2D workspace, since in this work we focus on only 2D navigation problems for motion planning. Given $\mathcal{D}^m$ and a robot model $\mathcal{M}$, a motion planning problem can be specified by an initial position x^{init} and a goal set X^{goal}. The 2D space is represented as a region in Cartesian space such that the position and orientation of the robot can be uniquely represented as a *pose* $(\mathbf{x}, \theta)$. Some parts of the space are designated as free space, and the rest is designated as obstacle.

The motion planning problem is solved by the motion planner $\mathcal{P}^m$ to compute a collision-free trajectory ξ^* (connecting x^{init} and a pose $x^{goal} \in X^{goal}$ taking into account any motion constraints on the part of the robot) with minimal trajectory length $Len(\xi) = L$. We use Ξ to represent the trajectory set that includes all collision-free trajectories. The *optimal* trajectory is

$$\xi^* = \text{argmin}_{\xi \in \Xi} Len(\xi), \quad (2)$$

where $\xi(0) = x_{init}$ and $\xi(L) = x_{goal} \in X_{goal}$.

A symbolic state s in $\mathcal{D}^t$ corresponds to a geometric constraint in $\mathcal{D}^m$ that can be represented as a set of poses X in the configuration space. For instance, the symbol "beside a table" corresponds to a (infinite) set of positions within a small range of the table. The geometric constraints ensure the motion-level feasibility between task state transitions.

We use a *state mapping function*, $f: X = f(s)$, to map the symbolic state s into a set of feasible poses X in continuous space, for the algorithm to sample from. We assume the availability of at least one pose $x \in X$ in each state s, such that the robot is in the free space of $\mathcal{D}^m$. If it is not the case, the state s is declared *infeasible*.

It should be noted that such state mapping functions break global optimality. We would like to be able to guarantee full motion-level optimality. But in continuous domains, such a guarantee is elusive due to the fundamental difference between representations at the two levels. In line with past research [2, 6, 33], we use a state mapping function, which makes it possible for us to achieve task-level optimality, i.e., optimality conditioned on the motion planner and state mapping function.

3.2 Definition of Task-Level-Optimal TMP-N

The input of a TMP-N problem is a six-tuple

$$\Omega : \langle \mathcal{D}^t, \mathcal{D}^m, s^{init}, S^G, x^{init}, f, \mathcal{M} \rangle$$

where $x^{init} \in f(s^{init})$, meaning that the geometric initial position is consistent with the symbolic initial state.

A satisfactory output of a TMP-N problem is a two-tuple,

$$\langle p, [\xi_0, \xi_1, \cdots, \xi_{N-1}] \rangle$$

that includes a symbolic plan and a set of collision-free trajectories, where $p(0) = s^{init}$, $p(N) \in S^G$, $|p| = N$, $\xi_0(0) = x^{init}$, $\xi_i(0) \in f(s_i)$, and $\xi_i(T_i) \in f(s_{i+1})$ for $i = \{0, 1, \ldots, N-1\}$. T_i is the end time of trajectory ξ_i.

Finally, we define a task-level optimal plan to be a lowest-cost plan p^*, conditioned on a motion planner $\mathcal{P}^m$ and state-mapping function f:

$$p^* = \text{argmin}_{p \in P} \Big(\sum_{0 \le i < |p|} Len(\xi_i) | \mathcal{P}^m, f \Big), \quad (3)$$

where $\xi_i = \mathcal{P}^m(\langle s_{i-1}, a_{i-1}, s_i \rangle, f, \mathcal{D}^m, \mathcal{M})$ is the trajectory returned by $\mathcal{P}^m$ given state transition $\langle s_{i-1}, a_{i-1}, s_i \rangle \in p$.

[1] The work of [6] is an exception, where task-level optimality is achieved in a computationally expensive way.

[2] The overall domain can be modeled as a hierarchical MDP, where the cost depends only on the latest transition: not at all on the transitions that happened beforehand. We use the formulation that depends on s' for ease of connection with the motion-level action evaluation in Section 3.3

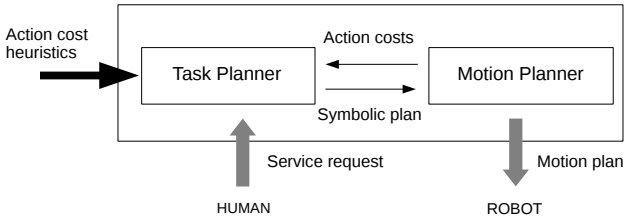


Figure 2: Overview of PETLON to efficiently solve TMP-N problems, with guaranteed task-level optimality.

3.3 PETLON

PETLON (Planning Efficiently for Task-Level-Optimal Navigation), is visualized in Figure 2, where task planner $\mathcal{P}^t$ and motion planner $\mathcal{P}^m$ serve as the two main components. The task planner interacts with humans by taking their service requests and generates symbolic plans. The motion planner generates realistic motion costs of each symbolic action. Our guarantee of task-level optimality relies on an admissible heuristic for motion costs, which can be easily obtained such as straight-line distance in 2D space.

Before introducing the algorithm, it is necessary to first define three cost functions:

- *Heuristic cost function* (h): we use h to represent our admissible heuristic cost function that computes the *Euclidean* distance between x and x' in 2D space.
$$h(x,x') = ||x - x'||_2 \tag{4}$$
- *Evaluated cost function* ($\hat{C}$): function $\hat{C}$ calls our motion planner $\mathcal{P}^m$ to estimate the geometric-level cost of traversing from $x \in f(s)$ to $x' \in f(s')$ given robot workspace $\mathcal{D}^m$:
$$\hat{C}(x,x') = Len\big(\mathcal{P}^m(\langle s,a,s'\rangle, f, \mathcal{D}^m, \mathcal{M})\big). \tag{5}$$
- *Maintained cost function* (*Cost*): values of $Cost(s,a,s')$ are initialized to $h(x,x')$, and then are selectively updated by $\hat{C}$ as the algorithm proceeds.

Overall, h is a very inexpensive operation compared to $\hat{C}$ that relies on calling a motion planner, and h underestimates motion cost. The following relationship among the three cost functions holds throughout the steps in PETLON:
$$h(x,x') \leq Cost(s,a,s') \leq \hat{C}(x,x'). \tag{6}$$

Therefore, PETLON is efficient to the extent that it minimizes the number of times the motion planner is called, while still ensuring that the returned plan is the same as if all state actions had been evaluated by the motion planner.

Algorithm 1 presents PETLON, taking the following terms as input:

- Initial state s^{init}, initial position x^{init} (in free space of $\mathcal{D}^m$), and goal specification S^G
- State mapping function $f : s \to X$
- Admissible heuristic cost function $h : (x,x') \to \mathbb{R}$ (to initialize the maintained cost function *Cost*)
- Task Domain Description $\mathcal{D}^t$, motion domain description $\mathcal{D}^m$, and robot model $\mathcal{M}$
- Task planner $\mathcal{P}^t : (s^{init}, S^G, Cost, \mathcal{D}^t) \to p$
- Motion planner $\mathcal{P}^m : (\langle s,a,s'\rangle, f, \mathcal{D}^m, \mathcal{M}) \to \xi$

Algorithm 1 PETLON algorithm

Require: s^{init}, x^{init},S^G,f, h, $\mathcal{D}^t$, $\mathcal{D}^m$,$\mathcal{M}$, $\mathcal{P}^t$, $\mathcal{P}^m$
Ensure: Symbolic plan p that is optimal at the task level
1: Initialize function *Cost* with h and sampled poses $x \in f(s)$: $Cost(s,a,s') \leftarrow h(x,x')$
2: Initialize empty state-action-state array: A^{evld}
3: Initialize the *so-far-best* cost: $C^{sfb} \leftarrow Inf$
4: **while true do**
5: $[\hat{p}^*, P] \leftarrow \mathcal{P}^t(s^{init}, S^G, Cost, \mathcal{D}^t)$, where $\hat{p}^*$ is optimal given *Cost*, P includes a set of (near-)optimal plans, and $\hat{p}^* \in P$
6: **if** $\langle s,a,s'\rangle \in A^{evld}, \forall \langle s,a,s'\rangle \in \hat{p}^*$ **then**
7: **return** $\hat{p}^*$
8: **end if**
9: **for each** $p \in P$ **and** $Cost(p) < C^{sfb}$ **do**
10: **for each** $\langle s,a,s'\rangle \in p$ **do**
11: Update motion planner $\mathcal{P}^m$
12: **if** $\langle s,a,s'\rangle \notin A^{evld}$ **then**
13: **while** $x' \notin$ FreeSpace($\mathcal{D}^m|\mathcal{M}$) **do**
14: Re-sample $x' \in f(s')$
15: **end while**
16: Evaluate motion cost: $Cost(s,a.s') \leftarrow \hat{C}(x,x')$
17: Append $\langle s,a,s'\rangle$ to A^{evld}
18: **end if**
19: **end for**
20: $C_{plan} = \Sigma_{\langle s,a,s'\rangle \in p} Cost(s,a,s')$
21: $C^{sfb} = min(C_{plan},\ C^{sfb})$
22: **end for**
23: **end while**

The algorithm starts by initializing: the maintained cost function *Cost* with our (admissible) heuristic function h (Line 1); an empty plan array for recording evaluated state-action-state tuple (Line 2); and a scalar cost value C^{sfb} with *Inf*, indicating the "evaluated" cost of the *so-far-best* plan (Line 3).[3] Entering the first while-loop, PETLON computes a set of symbolic plans P, including the current estimate of optimal plan $\hat{p}^*$, using the maintained cost function *Cost*. If all actions in the current optimal solution have been evaluated, PETLON returns $\hat{p}^*$ as the optimal task-level solution p^*(Line 6-8). If not, PETLON enters the outer for-loop (Lines 9-22), processing one plan $p \in P$ at each iteration. In the inner for-loop, each state-action-state tuple is considered, in a forward order (Lines 10-19). First, $\mathcal{P}^m$ is updated based on post-conditions of the previous action (Line 11), to adapt to potential changes in $\mathcal{M}$, $\mathcal{D}^m$, and feasibility of sampled poses in $\mathcal{P}^m$. If state-action-state tuple $\langle s,a,s'\rangle$ has not been evaluated before, PETLON first checks the feasibility of the sampled end pose x' (Line 13-Line 15, not necessary if considering static configuration spaces), then evaluates the cost value by calling $\hat{C}$ (Line 16), and last appends $\langle s,a,s'\rangle$ to the evaluated state-action-state set A^{evld} (Line 17).

The so-far-best cost C^{sfb} is updated if the current plan has lower evaluated cost value C_{plan} (Line 21). It maintains the lowest evaluated cost value among all evaluated plans.

[3] A "so-far-best" plan is the plan that is currently believed to be optimal using the maintained cost fuction. PETLON keeps updating the so-far-best plan as more action costs are evaluated.

3.4 Two Configurations of PETLON

In Line 5 of Algorithm 1, the task planner returns a plan set P, that includes the optimal plan based on the maintained *Cost* and an arbitrary number of suboptimal (near-optimal) plans.[4] Based on the trade-off between task and motion planning computational expense, PETLON can choose to evaluate the motion costs of actions from more plans in P in order to converge within fewer iterations. We evaluate two extreme configurations as noted in the following:

- **OptOne**: In each iteration, given the maintained cost function *Cost*, only the motion costs of the actions in the optimal $\hat{p}^*$ are evaluated.
- **OptAll**: In each iteration, the actions in all plans in P whose costs are lower than the so-far-best cost C^{sfb} (the lowest cost over all evaluated plans) are evaluated.

OptOne guarantees the minimal number of calls to the motion planner, which is beneficial when the motion planner is computationally expensive. OptAll has computational advantages when the task planner is relatively expensive. It trades more action cost evaluations for fewer iterations to converge compared to OptOne. PETLON's two configurations are evaluated in Section 5 with two different task planners.

3.5 Proof of Task-Level Optimality of PETLON

PROPOSITION 1. *Given motion planner $\mathcal{P}^m$ and state mapping function f, Algorithm 1 returns p^* that has the lowest cost over all satisfactory plans, i.e., the plan returned by Algorithm 1 satisfies Equation 3.*

Proof: Suppose this proposition is false, meaning that there exists at least one plan, p, whose geometric-level cost is lower than that of p^*, the plan returned by PETLON:

$$\sum_{\langle s,a,s'\rangle \in p} \hat{C}(s,a,s') < \sum_{\langle s,a,s'\rangle \in p^*} \hat{C}(s,a,s') \tag{7}$$

where $\hat{C}$ is the *evaluated cost function* as is detailed in Section 3.3 in the main paper.

In Algorithm 1, the task planner uses function *Cost*, the *maintained cost function*, and ensures that the returned plan is optimal given *Cost*:

$$\sum_{\langle s,a,s'\rangle \in p^*} Cost(s,a,s') \leq \sum_{\langle s,a,s'\rangle \in p} Cost(s,a,s'). \tag{8}$$

where p is an arbitrary satisfactory plan.

Algorithm 1 (Lines 12-18) also ensures that all action costs of the returned plan are evaluated by the motion planner:

$$\sum_{\langle s,a,s'\rangle \in p^*} Cost(s,a,s') = \sum_{\langle s,a,s'\rangle \in p^*} \hat{C}(s,a,s'). \tag{9}$$

We also know that the maintained cost function, *Cost*, cannot return a value that is higher than the evaluated cost, because function *Cost* is initialized by our heuristic cost function, h, that never overestimates the cost (Inequality 6):

$$\sum_{\langle s,a,s'\rangle \in p} Cost(s,a,s') \leq \sum_{\langle s,a,s'\rangle \in p} \hat{C}(s,a,s') \tag{10}$$

From Inequalities 8 and 10, and Equation 9, we draw conclusion:

$$\sum_{\langle s,a,s'\rangle \in p^*} \hat{C}(s,a,s') \leq \sum_{\langle s,a,s'\rangle \in p} \hat{C}(s,a,s') \tag{11}$$

Inequality 7 therefore contradicts with Inequality 11, proving the task-level optimality guarantee of p^*. ■

PROPOSITION 2. *Given motion planner $\mathcal{P}^m$ and state mapping function f, Algorithm 1 returns p^* in finite steps.*

Proof: Algorithm 1 terminates when the returned plan p has all its action costs evaluated by the motion planner, as suggested in Line 6. PETLON conducts at least one action cost evaluation at each iteration before termination. Given that the state space at the task level is finite, we can guarantee the convergence of Algorithm 1 within finite steps. ■

3.6 Highly Constrained TMP Problems

Algorithm and implementation details in the sampling step of PETLON ($x \in f(s)$) can influence the motion-level completeness. In particular, it presents a challenge on kinematically or dynamically constrained platforms that are relatively more common in TMP-M problems [10, 23, 33]. We do not specify this sampling process in PETLON, as it can be independently developed; for this purpose, we refer to existing research on efficiently sampling in the configuration space of highly confined domains [31, 32].

In highly constrained TMP domains, existing research has produced methods focusing on the motion-level feasibility of task-level actions, where approaches such as efficient re-sampling [6, 33] or backtracking task actions for resampling [23, 34] are commonly used. For instance, action backtracking methods enable reconsidering the end poses of previous task actions, in order to sample new initial poses for the currently infeasible action. These methods can be leveraged to make PETLON applicable to kinematically more challenging domains, though most TMP-N domains do not fall into this category.

Theoretically, if action backtracking is triggered to re-sample a pose for symbolic state s', we need to reevaluate the motion costs of related task-level actions, i.e., costs of all $\langle s,a,s'\rangle$ tuples in A^{evld}, so as to ensure the task-level optimality suggested in Proposition 1 (Section 3.5). This process ensures that the costs of navigation actions in A^{evld} are up to date. As for our test domains, the resampling process (Line 14 in Algorithm 1) is seldom reached, therefore action backtracking is rarely a concern.

4 ALGORITHM INSTANTIATION

Our task planner has been implemented using two declarative languages: Answer Set Programming (ASP) [12, 27] and Planning Domain Definition Language (PDDL) [28]. ASP is a popular general knowledge representation and reasoning (KRR) language, and has been used for solving task planning problems [26]. PDDL was developed for the International Planning Competition (IPC) and has been maintained by the IPC community. A recent empirical comparison has shown that PDDL-based planners perform better when tasks require long solutions, and ASP-based planners perform better when tasks require complex reasoning [17]. The goal of implementing both ASP-based and PDDL-based planners is to provide evidence

[4] Since solving a task planning problem can take a long time, some modern task planners can output some suboptimal plans before returning the optimal. Such plans are collected in plan set P.

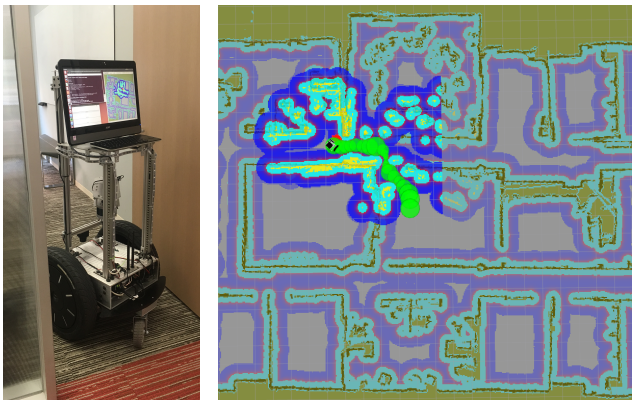


Figure 3: Left: Robot platform used in this research. Right: Occupancy-grid map (inflated) and a motion trajectory, indicated by a sequence of green planned for navigating through an office door.

for our hypothesis that PETLON is not sensitive to task planner selection.

At the task level, static predicates such as `has_door` and `inside` describe the spacial constraints on room accessibility and person locations respectively. We model three actions (`moveto`, `fetch`, and `deliver`), and represent the task states through non-static predicates (`delivered, loaded, at`). To map the task state to continuous space, we include a set of *geometric instances* using predicate `beside`, to describe task states in a small spacial area. For instance, `beside(fridge)` can be realized through function f as a pose within small area of the target fridge.

We use PRM* [20] to implement our motion planner. Compared to planners using a tree structure, such as RRT [25], the graph structure of PRM* is preferable due to the fact that we reuse the graph for multiple path evaluations with different starting points. Further, its asymptotic optimality, meaning almost-sure convergence to the optimal solution, ensures higher-quality motion plans, at least when using high sampling density. Here, we average two sample configurations per square meter, and the resultant plan quality has small variations (mostly within one meter) among trials. It should be noted that PETLON is not restricted to specific task or motion planning algorithms. The higher complexity the motion planner has, the greater the potential advantage PETLON can bring by saving computation for motion evaluation.

We implemented our ASP-based and PDDL-based task planners using award-winning solvers of Clingo [11] and FastDownward (FD) [15] respectively. Since IPC penalizes solvers that output suboptimal solutions, IPC solvers (including FD) do not output any plan until the optimal is found. As a result, we can only evaluate the OptOne configuration of PETLON using the FD solver. We use a laptop equipped with 2.2GHz i7 processor and 16GB RAM on OS X for all reported results.

5 EXPERIMENTS

PETLON has been implemented on a real robot as shown in Figure 3 (Left). The right of the figure shows the occupancy-grid map and the robot planning to navigate through an office door. The robot uses an RMP 110 mobile platform, onboard auxiliary battery, desktop computer (with touchscreen), and Velodyne VLP-16 for perception [21].

The test domain is our office environment, with the map prescanned and constructed by running the SLAM algorithm [35] (loaded before robot planning). Part of the domain is shown in Figure 3 on the right, containing seven rooms, four people, and four types of items (accordingly four types of containers). Each type of container has two to three instances. This test domain is later referred to as the **base domain**, on which we create variant domains for evaluating PETLON in different categories.

5.1 Baseline Methods

The goal of PETLON is to significantly reduce overall planning time while guaranteeing task-level optimality. Therefore, we evaluate PETLON based on both computational time and resulting plan quality by comparing PETLON (two configurations) to baselines with the following action cost formulations:

- *Constant cost:* Task actions are assumed to share the same unit cost. As a result, task planners generate plans with the fewest actions ($Cost = 1$). Our hypothesis was that this baseline would perform the worst in plan quality and the best in efficiency (due to the absence of a motion planner).
- *Heuristic cost:* Task actions are assumed to have cost equivalent to the Euclidean distance traveled (the motion planner is never called, and $Cost{=}h$ all the time).
- *Brute force:* Costs of all task actions are evaluated by the motion planner beforehand ($Cost{=}\hat{C}$). Our hypothesis was that this baseline would produce task-level-optimal solutions, but does not perform as well as PETLON in efficiency.

All three baselines use optimal task planners. However, depending on the motion cost evaluation strategy, these baselines produce trajectories of dramatically different quality. Only the brute-force baseline and PETLON (both configurations) generate task-level-optimal plans. Their computational times are then compared to demonstrate the improvement in planning efficiency introduced by PETLON. The other two baselines produce task-level-suboptimal plans.

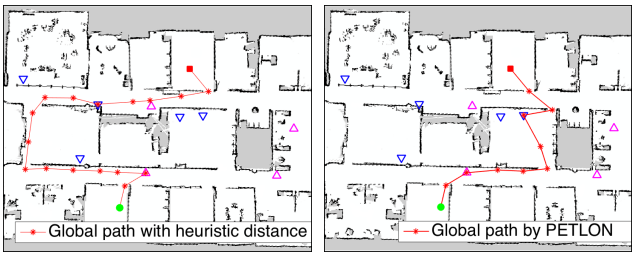
5.2 Illustrative Example

A plan quality comparison between the output from PETLON and the output from the *heuristic cost* baseline is shown in Figure 4. In this example, the robot needs to deliver a bottle of juice (5 instances marked as blue downward triangles) and a newspaper (4 instances marked as magenta upward triangles) to a target person (solid green circle). The initial state is specified using `at(corridor)` and `beside(init_pos)`, and the goal state is specified using the following four literals:

```
delivered(alice,n). newspaper(n).
delivered(alice,j). juice(j).
```

where `n,j` are available locations of newspapers and juices.

While considering an environment with low visibility from one location to another, such as an office domain, heuristic cost functions (such as suggested in Equation 4) may greatly underestimate the true



(a) Heuristic cost baseline (b) PETLON

Figure 4: The heuristic cost baseline uses only the Euclidean distance for plan cost value estimate (28.4m) and results in the suboptimal solution with actual length as 50m. The optimal solution by PETLON has higher heuristic cost value estimate (31.3m) but shorter actual length as 37m, compared to the heuristic-cost baseline.

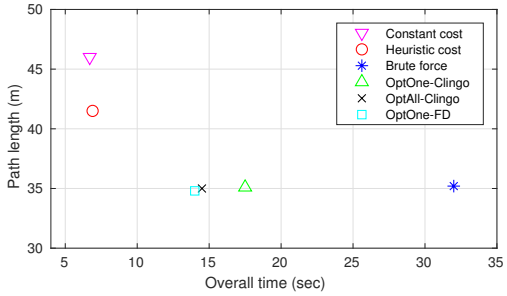


Figure 5: Plan quality (path length) vs. overall planning time. We compare six different planning algorithms using the base domain. PETLON (three versions) significantly reduces the overall planning time, while ensuring task-level optimality.

motion cost value, resulting in suboptimal plans (50m vs. 37m in travel length in this case). In this example, the geometric instances in the task planning domain are of four types: `fridge`, `newsstand`, `door`, and `person`. The task planner decides the order of the subtasks (such as `fetch(newspaper)`), and which instance to fulfill the action preconditions (such as `beside(newsstand1)`).

This pairwise example illustrates the necessity of cost evaluations using a motion planner (the baseline does not do so) and the importance of task-level optimality (the suboptimal solution causes a significant delay of task completion).

5.3 Experimental Results

Our central hypothesis is that PETLON is more efficient than planning approaches that pre-compute motion costs of all possible navigation actions, while still producing task-level-optimal solutions. Accordingly, we conducted the following four sets of experiments focusing on evaluating the performance of PETLON in efficiency and plan quality under different conditions.

Overall Performance of Six Planning Strategies. The two PETLON configurations with Clingo and FD implementations are compared with the baselines, on the task of delivering two specified

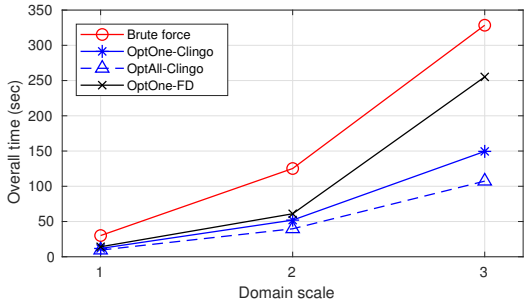


Figure 6: Overall planning time given different domain scale-up. PETLON (three versions) is more efficient than the baseline in every implementation.

kinds of items to a target person. As explained in Section 4, FD does not output suboptimal solutions, so we can only evaluate the OptOne configuration of PETLON using the FD solver. Each data point corresponds to an average over eight trials, and there are six strategies in total in this set of experiments.

Figure 5 reports their overall performance. We can see that PETLON (all three versions) significantly reduces the planning time (x-axis) to less than 20 seconds, in comparison to the brute-force baseline that took more than 30 seconds, while ensuring the best-quality solution (y-axis). It should be noted that, our domain is not very knowledge-intensive in the sense of the numbers of objects and their properties. Real-world applications are frequently much more knowledge-intensive than the brute-force baseline is able to handle; in domains with many objects that are irrelevant to the tested task request, the brute-force runtime grows exponentially (in the number of objects), and hence is not applicable for relatively large domains. The outputs of the other two baselines, *Constant cost* and *Heuristic cost*, are much worse in terms of quality due to their ignorance of true action costs.

Efficiency (Planning Time) in Domains of Different Sizes. In this experiment, with the same task specification, we scale up the domain size in terms of both the number of objects (adding complexity for $\mathcal{P}^t$), and map size (adding complexity for $\mathcal{P}^m$) by appending N copies of the base map to one another (left to right, then top to down), following its same structure. The initial positions of the robot are randomly selected in the three domains, to increase the likelihood that the robot will traverse newly-appended map areas.

The overall planning time includes the time consumed by both the task planner (Clingo-based or FD-based) and the motion planner. The task planners and motion planner are sensitive to the increasing number of objects and map size respectively. It should be noted that we add more objects and increase domain sizes, but the delivery task, as delivering a juice and a newspaper, does not change. Correspondingly, the length of the generated symbolic plan does not change.

Figure 6 shows the results of overall planning time given different levels of domain scale-up ($N = 2$ and $N = 3$). We see PETLON performs significantly better than the brute-force baseline, because brute-force has to evaluate costs of a combinatorially growing number of actions. Table 1 shows the number of motion cost evaluations

Table 1: The average number of cost evaluations conducted in the motion planner, in different domain scales.

	Domain scale		
	1	2	3
OptOne-Clingo	10.75	9.00	11.00
OptOne-FD	13.60	11.00	12.00
OptAll-Clingo	16.00	17.50	25.75
brute-force baseline	325.00	1295.00	2850.00

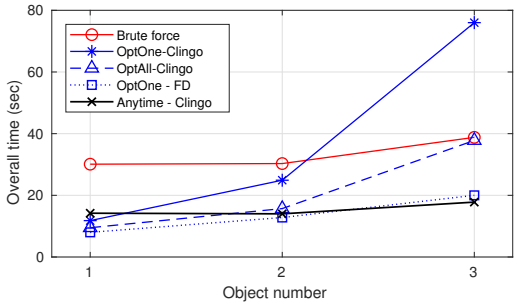


Figure 7: Overall planning time given tasks of different numbers of object to deliver. PETLON (OptOne configuration) that uses the FD-based task planner performs the best.

in different domain scales. From the last row, we can see that in the domain setting of $3x$ scale-up, brute-force conducts 2850 motion cost evaluations, whereas PETLON evaluates fewer than 26.

Planning Time Given Different Tasks. In this set of experiments, we vary the tasks by increasing the number of objects that need to be delivered using the base domain setup. The goal is to evaluate how sensitive the planning algorithms are given more complex tasks, i.e., tasks that require more task-level actions. PETLON may take substantially many iterations to converge given a task that requires many actions, so the computational expense of task planning can become a concern. We used both Clingo and FD task planners in our experiment. Intuitively, Clingo can be relatively more sensitive to plan length as it is a general-purpose reasoner not fine-tuned for planning tasks, whereas the FD planner is developed specifically for efficiently computing plans that include many actions.

The results are shown in Figure 7. As plan length increases (x-axis), OptOne-FD begins outperforming all other implementations, whereas the Clingo task planner is very sensitive to plan length. In such scenarios, making more calls to the task planner may not trade off favorably against motion evaluation; for that case, we introduce the "Anytime" implementation of PETLON, which trades off the optimality guarantee with superior efficiency without much loss of plan quality.

Anytime Property of PETLON, an Illustrative Example. In situations with strict time bounds, it can be useful for a planner to have an "anytime" property, meaning that the algorithm can terminate early while outputting monotonically improved solutions over time. In our case, we would like to see that PETLON produces good-quality plans (sequence of actions) given an early termination.

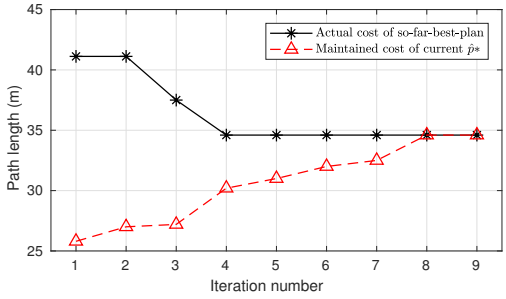


Figure 8: PETLON is an anytime algorithm that can return a valid solution even if it is interrupted before termination.

Figure 8 shows how the so-far-best cost C^{sfb} and the cost of $\hat{p}^*$ computed using *Cost* evolve over nine iterations until convergence. Note that C^{sfb} reaches the optimal value of PETLON (OptOne on the base domain) by iteration 4, while PETLON continues evaluating other potential satisfiable plans and finally, at iteration 9, it reports that the plan found at iteration 4 is optimal. This illustrative trial demonstrates PETLON's good anytime performance. We collected the time lengths of PETLON finding the optimal solution (after that, PETLON continues to evaluate other plans to ensure the optimality). The computational results are reported along with other implementations in Figure 7, where we can see Anytime-Clingo performs the best in comparison to all other planning strategies.

Note that, Clingo is actually capable of outputting all feasible plans, or all plans with costs lower than a certain value. With those two implementations, PETLON guarantees optimality within two calls of the task planner, by setting the second call to output all plans lower than the so-far-best cost; or, to output all feasible plans in the first iteration, and evaluate the rest without more calls of the task planner. The experimental setup in Figure 7 is purely to demonstrate potential issues to consider while using PETLON, given different characteristics brought by the choices of different planners.

6 CONCLUSIONS

In this paper, we introduce a novel algorithm, PETLON, that fully integrates task and motion planning for mobile robot service tasks. PETLON stands for "Planning Efficiently for Task-Level-Optimal Navigation" and is designed to produce task-level optimal plans while maintaining efficient planning time. PETLON has been evaluated using maps modeled after a real office environment, and has also been implemented on a real robot. Results show that PETLON significantly reduces the overall planning time compared to a baseline that pre-computes motion costs of all actions, while still maintaining task-level optimality.

This work opens a number of new research directions on task and motion planning (TMP). In the future, we intend to extend the work to dynamic environments, where cost estimates reflect plan quality during real-world execution. We also intend to extend the work with an exploration mechanism, where a robot can interact with the real world to learn costs of navigation actions.

7 ACKNOWLEDGMENT

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (IIS-1637736, IIS-1651089, IIS-1724157), Intel, Raytheon, and Lockheed Martin. Peter Stone serves on the Board of Directors of Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

REFERENCES

[1] Joydeep Biswas and Manuela Veloso. 2016. The 1,000-km challenge: Insights and quantitative and qualitative results. *IEEE Intelligent Systems* 31, 3 (2016), 86–96.

[2] Stephane Cambon, Rachid Alami, and Fabien Gravot. 2009. A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research* 28, 1 (2009), 104–126.

[3] Rohan Chitnis, Dylan Hadfield-Menell, Abhishek Gupta, Siddharth Srivastava, Edward Groshev, Christopher Lin, and Pieter Abbeel. 2016. Guided search for task and motion plans using learned heuristics. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 447–454.

[4] Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. 2018. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research* (2018).

[5] Leonardo De Moura and Nikolaj Bjørner. 2011. Satisfiability modulo theories: introduction and applications. *Commun. ACM* 54, 9 (2011), 69–77.

[6] Esra Erdem, Kadir Haspalamutgil, Can Palaz, Volkan Patoglu, and Tansel Uras. 2011. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *International Conference on Robotics and Automation (ICRA)*.

[7] Richard E Fikes and Nils J Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2, 3-4 (1971), 189–208.

[8] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2017. Sample-Based Methods for Factored Task and Motion Planning. In *Robotics: Science and Systems*.

[9] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2017. STRIPS Planning in Infinite Domains. *arXiv preprint arXiv:1701.00287* (2017).

[10] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2018. FFRob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research* 37, 1 (2018), 104–136.

[11] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. 2014. Clingo= ASP+ control: Preliminary report. *arXiv preprint arXiv:1405.3694* (2014).

[12] Michael Gelfond and Yulia Kahl. 2014. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press.

[13] Fabien Gravot, Stephane Cambon, and Rachid Alami. 2005. aSyMov: a planner that deals with intricate symbolic and geometric problems. In *Robotics Research. The Eleventh International Symposium*. Springer, 100–110.

[14] N Hawes, C Burbridge, F Jovan, L Kunze, B Lacerda, L Mudrová, J Young, J Wyatt, D Hebesberger, T Körtner, et al. 2016. The STRANDS Project: Long-Term Autonomy in Everyday Environments. *IEEE Robotics and Automation Magazine* (2016).

[15] Malte Helmert. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.(JAIR)* 26 (2006), 191–246.

[16] Jörg Hoffmann and Bernhard Nebel. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14 (2001), 253–302.

[17] Yuqian Jiang, Shiqi Zhang, Piyush Khandelwal, and Peter Stone. 2018. An Empirical Comparison of PDDL-based and ASP-based Task Planners. *arXiv* (2018).

[18] Leslie Pack Kaelbling and Tomás Lozano-Pérez. 2011. Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 1470–1477.

[19] Leslie Pack Kaelbling and Tomás Lozano-Pérez. 2013. Integrated task and motion planning in belief space. *International Journal of Robotics Research* 32, 9-10 (2013), 1194–1227.

[20] Sertac Karaman and Emilio Frazzoli. 2011. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research* 30, 7 (2011), 846–894.

[21] Piyush Khandelwal, Shiqi Zhang, Jivko Sinapov, Matteo Leonetti, Jesse Thomason, Fangkai Yang, Ilaria Gori, Maxwell Svetlik, Priyanka Khante, Vladimir Lifschitz, et al. 2017. BWIBots: A platform for bridging the gap between ai and human–robot interaction research. *The International Journal of Robotics Research* 36, 5-7 (2017), 635–659.

[22] Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2017. Learning to guide task and motion planning using score-space representation. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2810–2817.

[23] Fabien Lagriffoul, Dimitar Dimitrov, Julien Bidot, Alessandro Saffiotti, and Lars Karlsson. 2014. Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research* 33, 14 (2014), 1726–1747.

[24] Jean-Claude Latombe. 2012. *Robot motion planning*. Springer Science & Business Media.

[25] Steven M LaValle. 1998. Rapidly-Exploring Random Trees A New Tool for Path Planning. (1998).

[26] Vladimir Lifschitz. 2002. Answer set programming and plan generation. *Artificial Intelligence* 138, 1 (2002), 39–54.

[27] Vladimir Lifschitz. 2008. What is answer set programming?. In *Proceedings of the 23rd national conference on Artificial intelligence-Volume 3*. AAAI Press, 1594–1597.

[28] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. PDDL-the planning domain definition language. (1998).

[29] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. 2003. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research* 20, 1 (2003), 379–404.

[30] Nils J Nilsson. 1984. *Shakey the robot*. Technical Report. DTIC Document.

[31] Erion Plaku and Gregory D Hager. 2010. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 5002–5008.

[32] Erion Plaku, Lydia E Kavraki, and Moshe Y Vardi. 2010. Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Transactions on Robotics* 26, 3 (2010), 469–482.

[33] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 639–646.

[34] Mike Stilman and James J Kuffner. 2005. Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics* 2, 04 (2005), 479–503.

[35] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. 2005. *Probabilistic robotics*. MIT press.

[36] Marc Toussaint. 2015. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence*.

[37] Jason Wolfe, Bhaskara Marthi, and Stuart Russell. 2010. Combined Task and Motion Planning for Mobile Manipulation. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*. AAAI Press, 254–257.