Task-Motion Planning with Reinforcement Learning for Adaptable Mobile Service Robots

Yuqian Jiang[1], Fangkai Yang[2], Shiqi Zhang[3], and Peter Stone[1]
[1]Department of Computer Science, University of Texas at Austin, Austin, USA
[2]NVIDIA Corporation, Redmond, WA, USA
[3]Department of Computer Science, SUNY Binghamton, Binghamton, NY, USA
jiangyuqian@utexas.edu, fangkaiy@nvidia.com, szhang@cs.binghamton.edu, pstone@cs.utexas.edu

Abstract

Task-motion planning (TMP) addresses the problem of efficiently generating executable and low-cost task plans in a discrete space such that the (initially unknown) action costs are determined by motion plans in a corresponding continuous space. A task-motion plan for a mobile service robot that behaves in a highly dynamic domain can be sensitive to domain uncertainty and changes, leading to suboptimal behaviors or execution failures. In this paper, we propose a novel framework, *TMP-RL*, which is an integration of TMP and reinforcement learning (RL), to solve the problem of robust TMP in dynamic and uncertain domains. The robot first generates a low-cost, feasible task-motion plan by iteratively planning in the discrete space and updating relevant action costs evaluated by the motion planner in continuous space. During execution, the robot learns via model-free RL to further improve its task-motion plans. RL enables adaptability to the current domain, but can be costly with regards to experience; using TMP, which does not rely on experience, can jump-start the learning process before executing in the real world. TMP-RL is evaluated in a mobile service robot domain where the robot navigates in an office area, showing significantly improved adaptability to unseen domain dynamics over TMP and task planning (TP)-RL methods.

Introduction

Building mobile robots that behave intelligently in real environments is one of the central problems of robotics and artificial intelligence. Future service robots are expected to take general requests such as “deliver coffee to Alice”. To achieve a goal like this, the integration between high-level task planning and low-level motion planning, also known as *task-motion planning* (TMP), has been widely studied for robot manipulators (Erdem et al. 2011; Srivastava et al. 2014; Garrett, Lozano-Perez, and Kaelbling 2018) and navigation tasks in service robot (Lo, Zhang, and Stone 2018) or self-driving cars (Chen et al. 2015). TMP algorithms typically consist of a task planner that generates high-level task sequences in an abstract discrete space, possibly using an AI planning approach (Cimatti, Pistore, and Traverso 2008), and a motion planner that expands each task using algorithms such as Probabilistic Random Map (Kayraki et al. 1996) or Rapidly-exploring Random Trees (LaValle 1998) to generate a collision-free trajectory based on the current status of the environment. This hierarchical approach reduces the complexity of long-horizon motion planning by improving plan feasibility, quality, and scalability.

Despite the progress made on generating feasible and quality *offline plans*, during *execution*, a robot can still face domain uncertainties and changes that are not available at modeling time. This challenge is particularly pervasive for mobile service robots that cohabit with human and serve human requests (Veloso et al. 2015; Khandelwal et al. 2017). Mobile service robots usually have to navigate in building-wide areas, whose environmental dynamics involve, among many things, crowds of people, changing lighting conditions, and changed furniture layout, which are not practical for the motion planner to accurately model. Such dynamic changes may invalidate task-motion plans, leading to suboptimal behaviors and execution failures. Continually learning from execution experience and adapting to the changing domain is therefore crucial for mobile service robots to achieve long-term autonomy. To this end, reinforcement learning (RL) (Sutton and Barto 1998) has been used to build highly-adaptive autonomous agents (Mnih et al. 2015) and improve symbolic plan robustness and adaptability (Yang et al. 2018; Lyu et al. 2019) in various simulation domains, becoming an attractive approach to enable learning adaptive task-motion plans for mobile service robots.

Aiming to *improve adaptability of task-motion plans for mobile service robots*, in this paper, we propose to integrate TMP with RL such that the robot can constantly generate feasible, high-quality task-motion plans and rapidly learn from execution experience to adapt to domain changes. Inspired by PETLON, a recent task-level-optimal TMP algorithm (Lo, Zhang, and Stone 2018), and PEORL, a state-of-the-art task planning-RL framework (Yang et al. 2018), our approach features *two nested planning–reinforcement learning loops*:

- The *inner loop* is a complete TMP algorithm, where a symbolic plan is generated and each symbolic action is evaluated by the motion planner. Iterative learning and plan improvement is performed on rewards derived from motion plan costs.
- The *outer loop* is for learning to generate an optimal task-motion plan to accommodate domain uncertainty, change, and extra reward information. Each task-motion plan generated by the inner loop is executed in the outer loop to

learn from environmental rewards. The inner loop then uses the learned values to generate an improved plan in the next episode. When the outer loop terminates, the robot has learned a task-motion plan that has adapted to the observed domain changes.

In the framework above, the inner loop generates a high quality task-motion plan based on its own discrete and continuous models, leading to a jump-start of plan quality. The outer-loop helps the task-motion planner to fine-tune the plans by learning from the environment, improving the adaptability of TMP facing domain uncertainty and change. The duality between the inner and outer loop allows a seamless integration of TMP with RL such that motion planning in a continuous model and reinforcement learning from the real execution experience can jointly contribute to improving TMP.

Our approach is generic in the sense that a variety of task planning, motion planning, and reinforcement learning approaches can be used. In this paper, we instantiate our approach using the same symbolic planning and reinforcement learning approach as PEORL (Yang et al. 2018), including action language $\mathcal{BC}$ (Lee, Lifschitz, and Yang 2013) for task planning due to its expressiveness and efficient implementation using answer set solver CLINGO, and R-learning (Mahadevan 1996) for reinforcement learning. We have evaluated the approach both in the Gazebo simulator (Koenig and Howard 2004) and on a real service robot that operates in an office area. Compared to PETLON, a recent task-level-optimal TMP algorithm (Lo, Zhang, and Stone 2018) and PEORL, a recent Task Planning (TP)-RL approach (Yang et al. 2018), TMP-RL demonstrates superior adaptability to environmental uncertainties; it achieves better task performance than PETLON, and faster convergence than PEORL. The experiment is extended with a sequence of different scenarios, showing that TMP-RL can smoothly reuse learned information to improve long-term performances.

Related Work

Task planning (Ghallab, Nau, and Traverso 2004) and motion planning (Choset et al. 2005) generate plans in symbolic and continuous spaces respectively. Robots need task planning to accomplish goals that are impossible through individual actions, and need motion planning to generate trajectories that can be executed in the real world. Although robots that operate in the real world need capabilities of both task and motion planning, it is not until recent years that the term of Task and Motion Planning (TMP) was used in the literature to refer to algorithms that integrate both planning paradigms (Dantam et al. 2018; Srivastava et al. 2014; Garrett, Lozano-Perez, and Kaelbling 2018; Lo, Zhang, and Stone 2018; Kaelbling and Lozano-Pérez 2013; Lagriffoul et al. 2014; Chen et al. 2015). Existing research on TMP algorithms have various foci, such as ensuring symbolic actions' feasibility via motion planning (Srivastava et al. 2014), integrated symbolic planning under uncertainty and motion planning (Kaelbling and Lozano-Pérez 2013), and leveraging symbolic search heuristics in motion planning space (Chen et al. 2015; Garrett, Lozano-Perez, and Kaelbling 2018). Recently proposed PETLON (Lo, Zhang, and Stone 2018), which uses sampling-based probabilistic motion planning methods to evaluate costs of task-level actions is most similar to the inner loop of our work, but in our work, we use RL to learn rewards derived from real action costs, whereas PETLON is purely a planning method. While generating feasible, low-cost task-motion plans is the major focus of existing work on TMP, to the best of our knowledge, mixing task-motion planning and learning from execution to accommodate domain uncertainty and change for long range navigation tasks in mobile robots has not been investigated before.

The integration of symbolic planning with reinforcement learning has been studied in a variety of approaches (Parr and Russell 1998; Hogg, Kuter, and Munoz-Avila 2010; Leonetti, Iocchi, and Stone 2016). Recent approaches such as PEORL (Yang et al. 2018) and SDRL (Lyu et al. 2019) utilize closed-loop communication between planning and learning: an optimal symbolic plan is obtained from a mutually beneficial, iterative process of planning and learning. These approaches are mainly evaluated in simulation domains, but an integrated robot system is usually equipped with well-designed motion planners that can be used to evaluate the outcomes of task plans before execution, calling for an integration of TMP with RL. Our approach is inspired by PEORL and PETLON, but generalizes them into two nested loops to capture the complete task-motion planning, execution and learning loop for mobile robots. The two nested loops allow estimates made by motion planner and values learned from the environment to jointly improve the quality of plans. Consequently, TMP-RL is adaptive to real-world changes like PEORL while efficiently leveraging a motion planner to generate economical task plans like PETLON. To the best of our knowledge, our work is the first to apply reinforcement learning to improve adaptability of task-motion plans for service robots, where task planning in discrete spaces and motion planning, execution and learning in continuous spaces are handled in a unified framework.

Preliminaries

In this section, we individually introduce the symbolic planning, motion planning and learning technologies that will be combined in our framework introduced in the next section.

Symbolic Planning

An *action description* $\mathbb{D}$ in the language $\mathcal{BC}$ (Lee, Lifschitz, and Yang 2013) includes *fluent constants* that represent the properties of the world and *action constants*. A fluent atom is an expression of the form $f = v$, where f is a fluent constant and v is an element of its domain. An action description is a finite set of *causal laws* that describe how fluent atoms are related with each other in a single time step, or how their values are changed from one step to another, possibly by executing actions. For instance, $(A$ **if** $A_1, \ldots, A_m)$ is a *static law* that states at a time step, if $A_1, \ldots, A_m$ holds then A is true. Another static law (**default** $f = v$) states that by default, the value of f equals v at any time step. $(a$ **causes** A_0 **if** $A_1, \ldots, A_m)$ is a *dynamic law*, stating that at any time step, if $A_1, \dots, A_m$ holds, by executing action a, A_0 holds in the next step. (**nonexecutable** a **if** $A_1, \dots, A_m$) states that at any step, if $A_1, \dots, A_m$ holds, action a is not executable.

A *state* s is a complete set of fluent atoms, and a transition is a tuple $\langle s_1, a, s_2 \rangle$ where s_1, s_2 are states and a is a (possibly empty) set of actions. Let $\mathbb{I}$ be the initial state and $\mathbb{G}$ be goal state. The triple $(\mathbb{I}, \mathbb{G}, \mathbb{D})$ is called a planning problem. A plan can be computed using answer set solver such as CLINGO. Throughout the paper, we use Π to denote both the plan and the transition path by following the plan. Automated planning can be achieved by an answer set solver.

Motion Planning

A configuration space includes a set of all possible, potentially high-dimensional, configurations, where a configuration describes a possible pose of the robot. The output of a motion planner is a collision free trajectory, i.e., a sequence of discrete motions that can be directly passed to the joints of robot for execution. In this work, we consider a mobile robot that moves in 2D spaces. where we directly search in the 2D workspace (instead of higher-dimensional configuration space). 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 $\mathcal{P}^{geo}$ to compute a collision-free trajectory ξ^* (connecting $\mathbf{x}^{init}$ and a pose $\mathbf{x}^{goal} \in \mathbf{X}^{goal}$ 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 satisfactory trajectories. The *optimal* trajectory is $\xi^* = \text{argmin}_{\xi \in \Xi} Len(\xi)$, where $\xi(0) = \mathbf{x}_{init}$ and $\xi(L) = \mathbf{x}_{goal} \in \mathbf{X}_{goal}$. In particular, we use `global_planner`, an off-the-shelf package from the Robot Operating System (ROS) (Quigley et al. 2009) community for motion planning, which generates trajectories using gradient descent together with standard A^* and Dijkstra's search.

R-learning for Finite Horizon Problems

A Markov Decision Process (MDP) is defined as a tuple $(\mathcal{S}, \mathcal{A}, P^a_{ss'}, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the sets of symbols denoting states and actions, the transition kernel $P^a_{ss'}$ specifies the probability of transition from state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$ by taking action $a \in \mathcal{A}$, $r(s,a) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is a reward function bounded by $r_{\max}$, and $0 \leq \gamma < 1$ is a discount factor. A solution to an MDP is a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maps a state to an action. Model-free RL concerns on learning a near-optimal policy by executing actions and observing transitions and rewards.

To evaluate a policy π, R-learning (Mahadevan 1996) applies to the expected un-discounted sum of reward for finite horizon problems. Define $J^\pi_{\text{avg}}(s) = \mathbb{E}[\sum\limits_{t=0}^{T} r_t | s_0 = s]$, and the *gain reward* $\rho^\pi(s)$ reaped by policy π from s as $\rho^\pi(s) = \lim\limits_{T \to \infty} \frac{J^\pi_{\text{avg}}(s)}{T} = \lim\limits_{T \to \infty} \frac{1}{T} \mathbb{E}[\sum\limits_{t=0}^{T} r_t]$. R-learning is a model-free value iteration algorithm that can be used to find the optimal policy for the average reward criterion. At the t-th iteration (s_t, a_t, r_t, s_{t+1}), the following update is performed:

$$\begin{aligned} R_{t+1}(s_t, a_t) &\xleftarrow{\alpha_t} r_t - \rho_t(s_t) + \max_a R_t(s_{t+1}, a) \\ \rho_{t+1}(s_t) &\xleftarrow{\beta_t} r_t + \max_a R_t(s_{t+1}, a) - \max_a R_t(s_t, a), \end{aligned} \tag{1}$$

where α_t, β_t are the learning rates, and $a_{t+1} \xleftarrow{\alpha} b$ denotes the soft update rule $a_{t+1} = (1 - \alpha)a_t + \alpha b$.

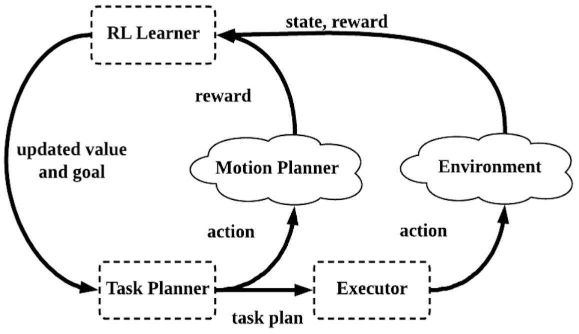


Figure 1: An illustration of our TMP-RL framework

TMP-RL Framework

The TMP-RL framework proposed in this paper is shown in Figure 1. The inner loop consists of a task planner, a motion planner and a reinforcement learner that iteratively performs planning and learning to generate *a feasible and low-cost task-motion plan*. Once the inner loop returns a task-motion plan, it is sent to execution in the outer loop, where the reinforcement learner performs value iteration on the reward derived from execution experience. The learned values are returned into the symbolic planner, and the inner loop runs again to generate an improved task-motion plan leveraging the motion planner and learned experience. The framework is explained in detail below.

Optimal Task Planning Conditioned on Motion Planning

A task planning problem defines the objective of generating a satisfactory plan Π^τ, i.e., a sequence of actions given a planning problem $(\mathbb{I}^\tau, \mathbb{G}^\tau, \mathbb{D}^\tau)$, where $\mathbb{D}^\tau$ is a domain independent symbolic formulation given by human experts, $\mathbb{I}^\tau$ is an initial state and $\mathbb{G}^\tau$ is a goal state. As in PEORL, $\mathbb{D}^\tau$ consists of causal laws that formulates preconditions and effects of actions, such as *approach* door D_1 causes the robot besides D_1 if currently the robot is beside D_2 and D_1 is accessible from D_2:

approach(D_1) **causes** *besides*(D_1) **if** *beside*(D_2), *acc*(D_2, D_1)

and static relationship on fluents, such as symmetry of accessible relationship: *acc*(D_1, D_2) **if** *acc*(D_2, D_1).

A motion planning problem concerns on generating a collision free trajectory $\xi(\mathbb{I}^m, \mathbb{G}^m)$ given a motion planning problem $(\mathbb{D}^m, \mathbb{I}^m, \mathbb{G}^m)$ where $\mathbb{D}^m$ is a motion planning domain, $\mathbb{I}^m$ is an initial position and $\mathbb{G}^m$ is the goal position, such that the position $\mathbb{I}^m$ is connected with position $\mathbb{G}^m$.

We use a mapping function $f : X = f(s)$ that maps a symbolic state s into a set of feasible poses X in continuous space, for the motion planning 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 $\mathbb{D}^m$. If it is not the case, the state s is declared infeasible. Given a motion planning domain $\mathbb{D}^m$ and a task plan Π^τ for task planning problem $(\mathbb{D}^\tau, \mathbb{I}^\tau, \mathbb{G}^\tau)$, a plan refinement of Π^τ w.r.t motion planner, denoted as Π^m, is a sequence of collision free trajectories obtained by perform motion planning on each navigation actions, i.e., $\Pi^m = \bigcup_{\langle s,a,s'\rangle \in \Pi^\tau} \xi(x, x')$, where $x \in f(s)$, $x' \in f(s')$. The cumulative cost of a task plan Π^τ is obtained by the cumulative length of its motion planning refinement Π^m, i.e., $\mathit{Cost}(\Pi^\tau) = \sum_{\xi \in \Pi^m} \mathit{Len}(\xi)$. An *optimal task plan conditioned on motion plan* is defined as the task plan Π^τ_o such that Π^m_o has the minimal length among all task plans.

TMP with Reinforcement Learning

Reward Given a symbolic transition $\langle s, a, s'\rangle$ where a can be refined by motion planner, we define a reward function r that is negative and inversely proportional to a distance metric of the motion plan that refines the navigation action a, mapped by a function $R : \mathbb{R}^+ \mapsto \mathbb{R}^-$:

$$r(s,a) = R(\mathit{Len}(\xi(x,x')) \propto \frac{1}{\mathit{Len}(\xi(x,x'))},$$

where $x \in f(s), x' \in f(s')$. One simple way to instantiate R is

$$r(s,a) = R(\mathit{Len}(\xi(x,x'))) = -\mathit{Len}(\xi(x,x')).$$

If a motion plan fails for transition $\langle s, a, s'\rangle$, we define $r(s, a) = -\infty$.

Domain Formulation We enrich the domain formulation $\mathbb{D}^\tau$ with the following causal laws formulating the effect of actions on cumulative plan quality:

$$a \textbf{ causes } \mathit{quality} = C + Z \textbf{ if } s, \rho(s,a) = Z, \mathit{quality} = C$$

where s is a state. The ρ-values are initialized optimistically to the upper-bound of gain reward, which is the reward derived from the L_p metric in the configuration space:

$$\textbf{default } \rho(s,a) = \max_{x,x'} R(||x - x'||_p)$$

where $x \in f(s), x' \in f(s'), x \neq x'$, for $\langle s, a, s'\rangle$ in $T(\mathbb{D}^\tau)$, $p \in \mathbb{R}^+$, and L_p metric stands for $||x - x'||_p = \left(\sum_{i=1}^{n} |x_i - x'_i|^p\right)^{-p}$.

Algorithm 1 Task-Motion Planning

```
Require: $(\mathbb{I}^\tau, \mathbb{G}^\tau, \mathbb{D}^\tau, f, \mathbb{D}^m, q_0, P_0)$ where $quality > q_0 \in G^\tau$,
    and an exploration probability $\epsilon$
 1: $t \Leftarrow 0$
 2: while $t < +\infty$ do
 3:     $\Pi^* \Leftarrow \Pi^\tau_t$
 4:     obtain a plan $\Pi^\tau_t \Leftarrow Plan(\mathbb{I}^\tau, \mathbb{G}^\tau, \mathbb{D}^\tau \cup P_t)$
 5:     if $\Pi^\tau_t = \emptyset$ then
 6:         return $\Pi^*$
 7:     end if
 8:     for symbolic transition $\langle s, a, s'\rangle \in \Pi^\tau_t$ do
 9:         if $a$ cannot be refined by motion planner then
10:             continue
11:         end if
12:         obtain initial pose $x = f(s)$ and goal pose $x' = f(s')$
13:         generate motion plan $\xi(x, x')$
14:         calculate reward $r(s,a) = R(Len(\xi(x,x')))$
15:         update $R(s,a)$ and $\rho^a(s)$ using (1).
16:     end for
17:     calculate quality of $\Pi^\tau_t$ by (2).
18:     update planning goal $G \Leftarrow (quality > quality_t(\Pi^\tau_t))$.
19:     update facts $P_t \Leftarrow \{\rho(s,a) = z : \langle s, a, s'\rangle \in \Pi, \rho^a_t(s) =$
        $z\}$
20:     $t \Leftarrow t + 1$
21: end while
```

Planning Goal At any episode t, planning goal $\mathbb{G}^\tau_t$ contains a regular logical constraint describing the goal condition plus a linear constraint of the form

$$\mathit{quality} \geq \mathit{quality}(\Pi^\tau_t) \tag{2}$$

where $\mathit{quality}(\Pi^\tau_t) = \sum_{\langle s,a,s'\rangle \in \Pi^\tau_t} \rho(s,a)$ for some task plan Π^τ_t. In the planning – learning loop, the linear constraint guides the planner to generate a plan with cumulative quality higher than a previous one, measured by learned ρ-values, leading to the iterative process of plan improvement based on reinforcement learning.

Algorithm for TMP Algorithm 1 describes our inner loop of task-motion planning. The input to the algorithm includes a motion planning domain and a task planning problem. q_0 is initialized to be $-\infty$, and $P_0 = \emptyset$. The algorithm first generates a task plan (Line 4). Then it iterates on each symbolic transition in the plan, and for each navigation action, it obtains the initial and goal poses in 2D domain (Line 12), generates motion plan (Line 13) and returns reward (Line 14). Value iteration of R-learning is performed with the reward (Line 15). At the end of this process, plan quality is computed using the learned ρ values (Line 17), and it is used as the new constraint in the planning goal (Line 18), setting a baseline for the planner in the next iteration. The learned ρ values are also updated in the symbolic formulation (Line 19). When the algorithm terminates, it outputs a task plan that cannot be further improved w.r.t the motion planner.

TMP Execution and Learning

Once a task-motion plan is generated, it is sent for execution, which goes to the outer loop of learning from real ex-

Algorithm 2 Task-Motion Planning and Learning

```
Require: $(\mathbb{I}^\tau, \mathbb{G}^\tau, \mathbb{D}^\tau, f, \mathbb{D}^m)$ where $quality > 0 \in G^\tau$, and an
    exploration probability $\epsilon$
 1: $P_0 \Leftarrow \emptyset, \Pi_0^\tau \Leftarrow \emptyset, q_0 = -\infty, t = 0$
 2: while $t < +\infty$ do
 3:     $\Pi^* \Leftarrow \Pi_t^\tau$
 4:     obtain a task-motion plan by calling Algorithm 1 $\Pi_t^\tau \Leftarrow$
        $TMP(\mathbb{I}^\tau, \mathbb{G}^\tau, \mathbb{D}^\tau, f, \mathbb{D}^\mu, q_t, P_t)$.
 5:     if $\Pi_t^\tau = \emptyset$ then
 6:         return $\Pi^*$
 7:     end if
 8:     for symbolic transition $\langle s, a, s' \rangle \in \Pi_t^\tau$ do
 9:         execute $a$ and obtain true reward $r(s, a)$.
10:         update $R(s, a)$ and $\rho^a(s)$ using (1).
11:     end for
12:     calculate quality of $\Pi_t^\tau$ by (2).
13:     update plan quality $q_t \Leftarrow quality_t(\Pi_t^\tau)$.
14:     update facts $P_t \Leftarrow \{\rho(s, a) = z : \langle s, a, s' \rangle \in \Pi_t^\tau, \rho_t^a(s) =$
        $z\}$
15:     $t \Leftarrow t + 1$
16: end while
```

ecution experience, where Algorithm 1 becomes the planning subroutine (Line 4) in Algorithm 2. In Algorithm 2, each action is executed in the environment, and the true reward is obtained (Line 9). The value iteration performed on the true reward received during execution further rewrites the value learned through motion planner and feed back into the TMP algorithm (Line 14) to iteratively generate a task-motion plan that is adaptable to domain change.

The difference between Algorithm 2 and Algorithm 1, is in Line 4: Algorithm 1 makes a task planning call and Algorithm 2 makes a TMP call. Such duality brings a unification of refining task plans through motion planner and through learning from the environment: the quality of task plans are learned in the same framework and the learned values are propagated back into a symbolic representation so that motion planners and execution experience can jointly improve the task plan.

Experiments

We evaluate our approach using a simulated mobile service robot (Figure 2a) in an office building floor (Figure 2b) in Gazebo, and later demonstrate it on a real robot. The simulation is created to closely match the real robot platform (shown in Figure 2c) and the environment it operates in (shown in Figure 2d).

We compare the performance of the proposed TMP-RL algorithm (Algorithm 2) with PETLON (Lo, Zhang, and Stone 2018), a TMP algorithm, and PEORL (Yang et al. 2018), a TP-RL approach which iteratively generates symbolic task plans and performs reinforcement learning during execution. The plan generated by each algorithm is executed, and we compare the curves of actual reward received in each episode. Our hypothesis is that TMP-RL outperforms TMP when real action rewards are unexpected, and it outperforms TP-RL with higher quality exploration and faster adaptation to domain changes.

Implementation

Our system uses the answer set solver Clingo for task (symbolic) planning[1]. The task planning domain models navigation actions (*approach*, *open_door*, *go_through*), as well as non-navigation actions (such as *pick_up*, *put_down*). Only navigation actions are required in this experiment.

Path planning is implemented using the navigation stack of Robot Operating System (ROS) (Quigley et al. 2009). In TMP and TMP-RL implementations, the global path planner is called to generate a trajectory for each navigation action, and the motion costs are estimated by the sum of distances between waypoints on the trajectory. The plan quality constraints are implemented for T(M)P-RL algorithms. Learning of the ρ values is implemented using Equation (1) with learning rates $\alpha = 0.1, \beta = 0.5$. The default ρ-values of *approach* actions are implemented as the Euclidean distance between the target location and the landmark closest to the robot. The default reward of an *open_door* action is -3. All other state-action pairs that do not have an evaluated or default ρ value are assumed to have reward of -1.

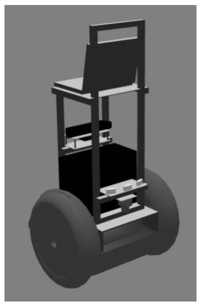
Gazebo Simulation

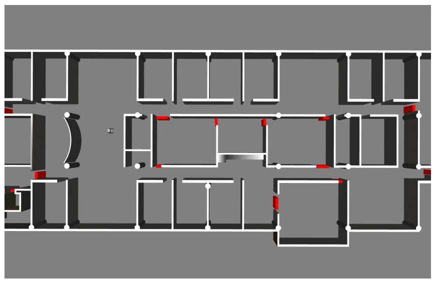
In this experiment, the robot's task is to go to a room where its service is requested. The robot starts near a landmark in an open space and the robot's end position can be anywhere in the target room. The reward is defined as the negative of the execution time. The room has three initially closed doors that are available for entrance. The task planner determines which entrance the robot will use. Figure 3 shows the experiment set-up and three competitive task plans. With 30 regions and 12 doors in the domain, many other feasible plans may be generated by task planner, and some plans involve significant detour. Thus, generating feasible, low-cost plans efficiently and quickly adapting to domain uncertainties is quite challenging in this domain. For instance, plan (1) uses the top door along the blue path and has 3 symbolic actions: $approach(top\_door)$, $open\_door(top\_door)$, $go\_through(top\_door)$.

Table 1 shows the task plan length, motion plan length, and average execution time of the competitive plans. Among the three plans, plan (2) features the shortest navigation distance, but it takes 9 actions and requires crossing 3 doors. Plan (3) has 3 actions and the second shortest navigation distance. In this environment, opening door takes longer than what the robot expects, and opening the bottom door is particularly expensive. Precisely, the duration of executing an *open_door* action is sampled from a normal distribution with a standard deviation of (10 seconds). Opening the bottom door takes 60 seconds on average, while the mean open time is 20 seconds for other doors. Therefore, plan (1) which uses the top door has the lowest expected execution time. This example shows one situation where all three levels of capability are required to efficiently find the optimal real-world plan.

Evaluation of TMP, TP-RL, TMP-RL We use PETLON (TMP) and PEORL (TP-RL) for comparison in this eval-

[1]https://github.com/potassco/clingo/releases.


(a) Simulated robot


(b) Gazebo simulation environment

(c) Real robot

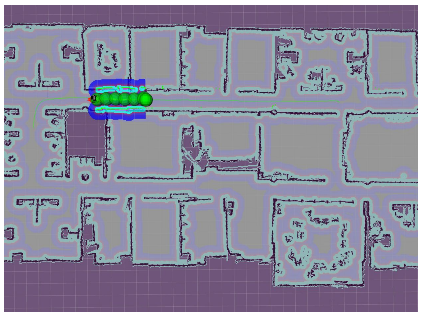

(d) Visualization of the robot in a real map

Figure 2: Our experiments use Gazebo to simulate an office environment and a service robot operating in this environment. The simulation matches the real robot and the office environment it operates in.

Plan	Task Plan Length	Motion Plan Cost	Average Execution Cost
(1)	3	60.8	**80.6**
(2)	9	**45.5**	126.9
(3)	3	53.1	116.6

Table 1: Plan costs at different levels of abstraction.

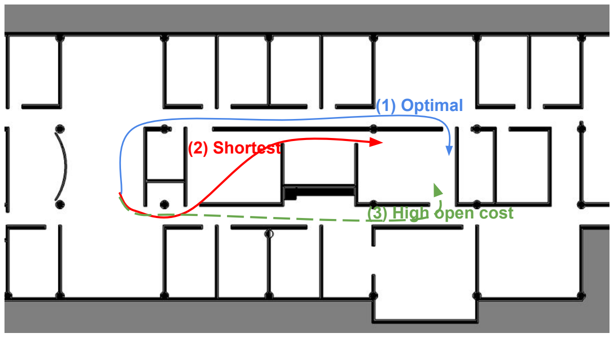


Figure 3: Three competitive plans for the task.

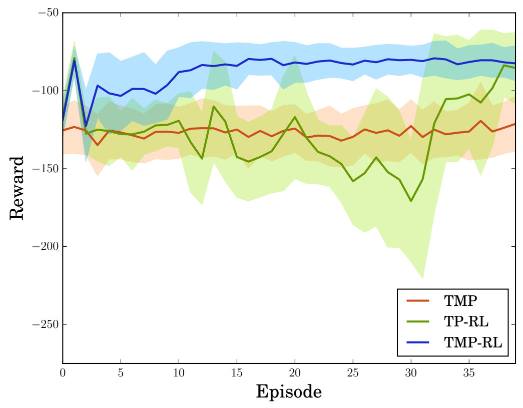
uation. For every approach we conducted 50 runs with 40 episodes in each run. The variability among the trials are caused by noisy action costs of navigation and opening doors. For RL-based methods, they can generate different plans depending on experiences in previous episodes.

Figure 4a plots the learning curves for reward received in 40 episodes, averaged over 50 runs with the shaded regions representing one standard deviation from the mean. Figure 4b shows for each approach, the average number of episodes that the three competitive plans and other feasible plans are executed. Equipped with the reinforcement learner to refine their plans, TMP-RL and TP-RL converge to the practically optimal plan, but TMP-RL converges significantly faster. Using motion plan costs in task plan evaluation ensures that TMP-RL makes steady improvements after the first two episodes, while TP-RL learns everything from executing the plans in the environment. TP-RL has low variances in the first two episodes, because the task planner first selects the plans with the smallest number of actions (plans (1) and (3) in Figure 3), but much higher variances afterwards. As shown in Figure 4b, TP-RL had to execute many task plans that are logically valid but significantly worse in quality directly in the environment, which is expensive and potentially dangerous for real robots. TMP executes the plan with the shortest navigation distance in every episode (plan (2)), without learning any information from execution.

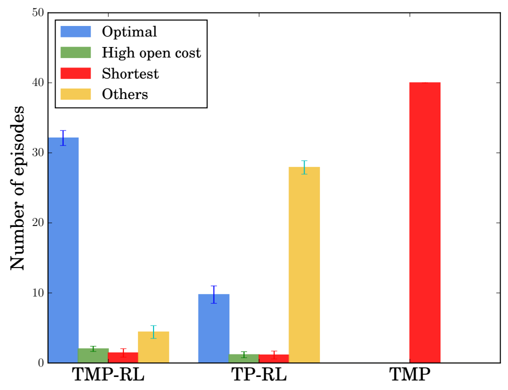
In terms of planning time, each call of the answer set solver is timed-out after 5 seconds for all three approaches. The "anytime" property of PETLON allows it to find plans of good quality given early termination. This property also holds for Algorithms 1 and 2 since they incrementally tighten the bound of plan quality. When the solver fails to find a better plan within the time-out, the algorithms return the current best plan.

Evaluation of TMP-RL in Multiple Tasks In long-term deployments, the robot can be asked to achieve the same end goal from different starting positions. For example, the robot may start in the mail room and deliver mail to an office in the morning, and deliver coffee from the kitchen to the same office in the afternoon. Since the initial states are different, the task planner and motion planner have to solve them as different problems, but TMP-RL can leverage the learned ρ-values to speed up exploration in later tasks. In order to demonstrate TMP-RL's ability to generalize learned values to different scenarios, we extend the previous experiment with two more tasks, each with a different starting position of the robot (shown in Figure 5a).

In this scenario, we compared continuously running TMP-RL for all three tasks against using TMP-RL to learn the second and the third tasks from scratch. In the former setting, the robot explored the first task for 15 episodes, and then switched to the second position and the third position while keeping the learned values. In the latter setting, the robot started at episode 15 and performed the second task, or

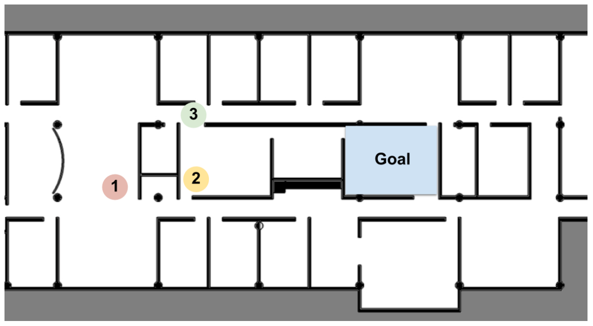


(a) Learning curves.

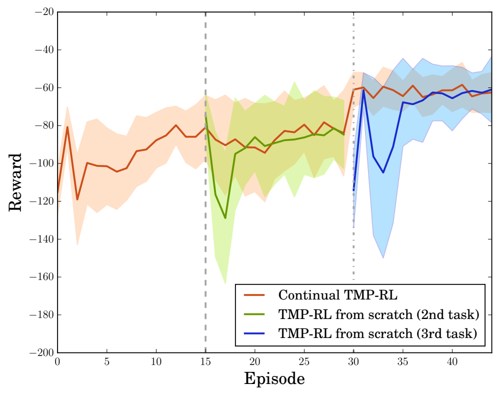


(b) Distribution of plan executions.

Figure 4: Comparisons between TMP-RL and baselines of TMP and TP-RL in the task of room-to-room navigation.



(a) Extended experiment with three different initial states.



(b) TMP-RL with continuous transfer vs. learning from scratch.

Figure 5: Evaluation of TMP-RL in multiple tasks

started at episode 30 and performed the third task. Figure 5b presents the learning curves averaged over 40 runs in these three settings, showing that learning the first task leads to faster, lower variance learning in the later tasks, in comparison with learning from scratch, indicating that the learned values can be transferred to accelerate learning other tasks.

Real Robot Demonstration

We also implement TMP-RL on a real robot (shown in Figure 2c), and demonstrate that it achieves the same performance as the results in the simulation experiments. Since the real office area closely matches the simulation environment with corresponding rooms, corridors and doors, it is possible to have a highly similar task setup to the one shown in Figure 3 and Table 1. We ran TMP-RL repeatedly on this task, and it successfully converged to the practically optimal plan in only eight episodes. A demonstration video showing the setup and the results is available[2].

[2]https://youtu.be/EyoqrpO3Qkk

Conclusion

We introduce a novel TMP-RL framework integrating task-motion planning (TMP) and reinforcement learning (RL) for adaptable robot sequential decision making. The framework mixes task planning, motion planning, and reinforcement learning in a closed loop with iterative improvements on plan quality over the course of execution. Experimental results show that TMP-RL combines the strengths of the individual paradigms: task-motion planning generates high-quality plans without performing costly learning in the real environment, and reinforcement learning refines task-motion planning in dynamic domains and generalizes learned information to new scenarios. Therefore, this framework provides important properties for long-term deployments of robots in dynamic environments. Future work includes using TMP-RL to improve the long-term performance of service robots in a variety of tasks.

References

Chen, C.; Gaschler, A.; Rickert, M.; and Knoll, A. 2015. Task planning for highly automated driving. In *Intelligent Vehicles Symposium*, 940–945.

Choset, H.; Lynch, K. M.; Hutchinson, S.; Kantor, G.; Burgard, W.; Kavraki, L.; and Thrun, S. 2005. Principles of robot motion: Theory, algorithms, and implementations.

Cimatti, A.; Pistore, M.; and Traverso, P. 2008. Automated planning. In van Harmelen, F.; Lifschitz, V.; and Porter, B., eds., *Handbook of Knowledge Representation*. Elsevier.

Dantam, N. T.; Kingston, Z. K.; Chaudhuri, S.; and Kavraki, L. E. 2018. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research*.

Erdem, E.; Haspalamutgil, K.; Palaz, C.; Patoglu, V.; and Uras, T. 2011. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 4575–4581. IEEE.

Garrett, C. R.; Lozano-Perez, T.; and Kaelbling, L. P. 2018. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research* 37(1):104–136.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Hogg, C.; Kuter, U.; and Munoz-Avila, H. 2010. Learning methods to generate good plans: Integrating htn learning and reinforcement learning. In *AAAI*.

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *The International Journal of Robotics Research* 32(9-10):1194–1227.

Kayraki, L.; Svestka, P.; Latombe, J.; and Overmars, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configurations space. *Proc IEEE Trans Robot Autom* 12(4):566–80.

Khandelwal, P.; Zhang, S.; Sinapov, J.; Leonetti, M.; Thomason, J.; Yang, F.; Gori, I.; Svetlik, M.; Khante, P.; Lifschitz, V.; and Stone, P. 2017. Bwibots: A platform for bridging the gap between ai and human–robot interaction research. *The International Journal of Robotics Research* 36(5-7):635–659.

Koenig, N., and Howard, A. 2004. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *International Conference on Intelligent Robots and Systems (IROS)*.

Lagriffoul, F.; Dimitrov, D.; Bidot, J.; Saffiotti, A.; and Karlsson, L. 2014. Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research* 33(14):1726–1747.

LaValle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning.

Lee, J.; Lifschitz, V.; and Yang, F. 2013. Action Language $\mathcal{BC}$: A Preliminary Report. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Leonetti, M.; Iocchi, L.; and Stone, P. 2016. A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artificial Intelligence* 241:103–130.

Lo, S.-Y.; Zhang, S.; and Stone, P. 2018. Petlon: Planning efficiently for task-level-optimal navigation. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 220–228. International Foundation for Autonomous Agents and Multiagent Systems.

Lyu, D.; Yang, F.; Liu, B.; and Gustafson, S. 2019. Sdrl: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *AAAI*.

Mahadevan, S. 1996. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning* 22:159–195.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Parr, R., and Russell, S. J. 1998. Reinforcement learning with hierarchies of machines. In *Advances in neural information processing systems*, 1043–1049.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 5. Kobe, Japan.

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

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*. Cambridge University Press.

Veloso, M.; Biswas, J.; Coltin, B.; and Rosenthal, S. 2015. CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In *Proceedings of IJCAI'15, the International Joint Conference on Artificial Intelligence*.

Yang, F.; Lyu, D.; Liu, B.; and Gustafson, S. 2018. Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. In *International Joint Conference of Artificial Intelligence (IJCAI)*.