Integrating Task-Motion Planning with RL for Robust Decision Making in Mobile Robots

Presenter: Mihir Suvarna

October 25th, 2022

Real-World Mobile Robots

How can we build a mobile robot that is:

- ➔ able to behave intelligently
- ➔ inexpensive
- ➔ active in real environments

Possible Approaches:

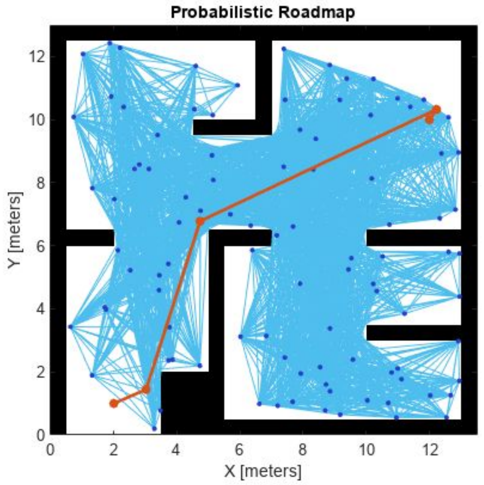
- ➔ Motion Planning
- ➔ Task Planning
- ➔ Reinforcement Learning

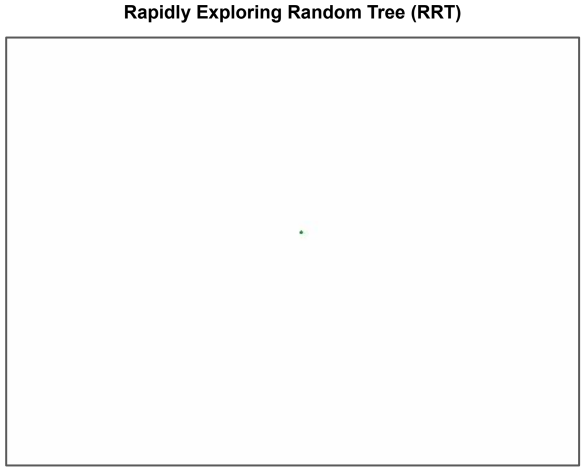
Intro to Motion Planning (MP)

Robot is given a task: “deliver this package”

Must use *motion planning* in continuous action space

Low-level and computationally $$$



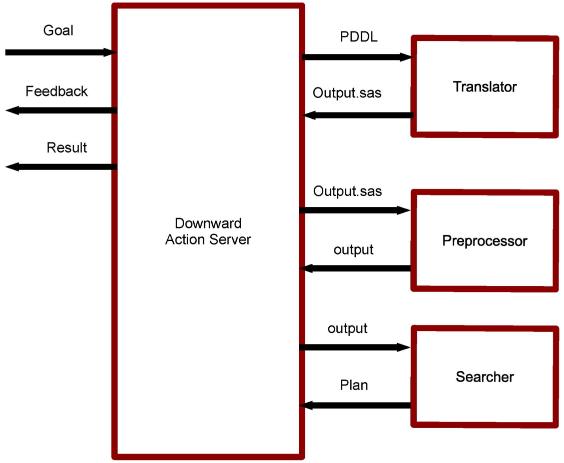


Combining MP with Task Planning (TMP)

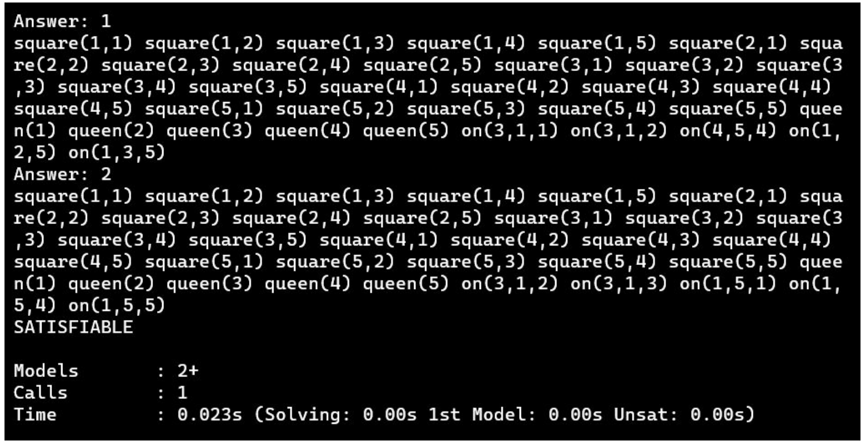
Generate subtasks, each have corresponding motion plan

Motion planner used to check feasibility

Discrete action space!



FastDownward PDDL



```
Answer: 1
square(1,1) square(1,2) square(1,3) square(1,4) square(1,5) square(2,1) squa
re(2,2) square(2,3) square(2,4) square(2,5) square(3,1) square(3,2) square(3
,3) square(3,4) square(3,5) square(4,1) square(4,2) square(4,3) square(4,4)
square(4,5) square(5,1) square(5,2) square(5,3) square(5,4) square(5,5) quee
n(1) queen(2) queen(3) queen(4) queen(5) on(3,1,1) on(3,1,2) on(4,5,4) on(1,
2,5) on(1,3,5)
Answer: 2
square(1,1) square(1,2) square(1,3) square(1,4) square(1,5) square(2,1) squa
re(2,2) square(2,3) square(2,4) square(2,5) square(3,1) square(3,2) square(3
,3) square(3,4) square(3,5) square(4,1) square(4,2) square(4,3) square(4,4)
square(4,5) square(5,1) square(5,2) square(5,3) square(5,4) square(5,5) quee
n(1) queen(2) queen(3) queen(4) queen(5) on(3,1,2) on(3,1,3) on(1,5,1) on(1,
5,4) on(1,5,5)
SATISFIABLE

Models       : 2+
Calls        : 1
Time         : 0.023s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
```

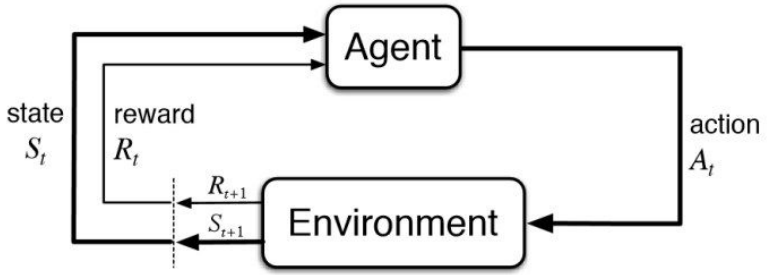
Clingo Answer Set Solver

Reinforcement Learning (RL)

Reinforcement learning has previously been used for unseen domains

But, still need LOTS of data, especially in a *real world setting*

No guarantee for convergence!



Problem at hand

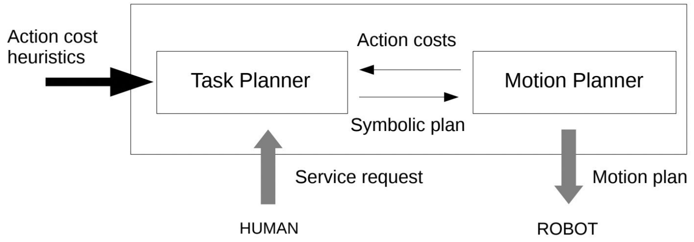
How can we use TMP and RL to develop a mobile robot that is able to:

- ➔ Generate high-quality task motion plans
- ➔ Learn from execution experience
- ➔ Adapt to domain uncertainty and changes
- ➔ Be more robust towards sub-optimal plans

We can use such an autonomous robot in augment with humans!

Related Work

The PETLON Algorithm to Plan Efficiently for Task-Level-Optimal Navigation

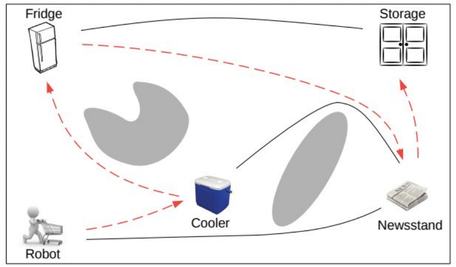


PETLON

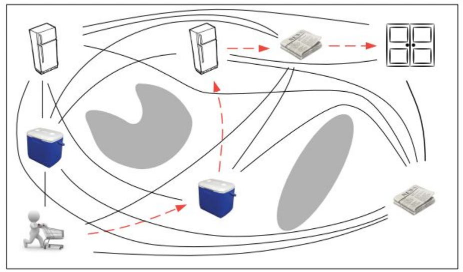
Sampling-based method for evaluating cost of task-level actions

Key Limitations

- ➔ Purely a planning method
- ➔ No experimentation done with mixing task-motion plans and execution
- ➔ Assumes that domain does *not* change



(a) Small number of objects.

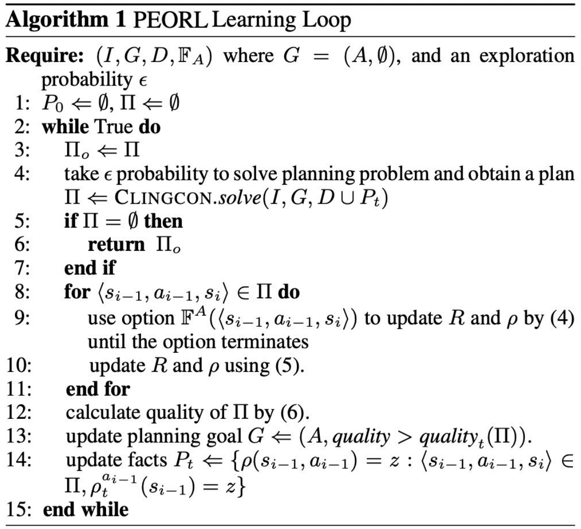


(b) Large number of objects (doubled)

More related work

PEORL

Combining symbolic plans with HRL (hierarchical reinforcement learning)

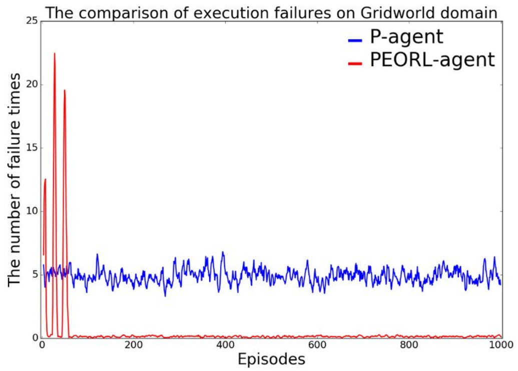


Algorithm 1 PEORL Learning Loop

```
Require: $(I, G, D, \mathbb{F}_A)$ where $G = (A, \emptyset)$, and an exploration
    probability $\epsilon$
 1: $P_0 \Leftarrow \emptyset, \Pi \Leftarrow \emptyset$
 2: while True do
 3:     $\Pi_o \Leftarrow \Pi$
 4:     take $\epsilon$ probability to solve planning problem and obtain a plan
        $\Pi \Leftarrow$ CLINGCON.solve$(I, G, D \cup P_t)$
 5:     if $\Pi = \emptyset$ then
 6:         return $\Pi_o$
 7:     end if
 8:     for $\langle s_{i-1}, a_{i-1}, s_i \rangle \in \Pi$ do
 9:         use option $\mathbb{F}^A(\langle s_{i-1}, a_{i-1}, s_i \rangle)$ to update $R$ and $\rho$ by (4)
            until the option terminates
10:         update $R$ and $\rho$ using (5).
11:     end for
12:     calculate quality of $\Pi$ by (6).
13:     update planning goal $G \Leftarrow (A, quality > quality_t(\Pi))$.
14:     update facts $P_t \Leftarrow \{\rho(s_{i-1}, a_{i-1}) = z : \langle s_{i-1}, a_{i-1}, s_i \rangle \in$
        $\Pi, \rho_t^{a_{i-1}}(s_{i-1}) = z\}$
15: end while
```

Key Limitations

- ➔ Artificial domain knowledge (ex: can die in Atari before learning not to in game, how to replicate on real robots?)
- ➔ Needs a pre-mapped environment on integrated robot system (can be $$$) to evaluate task outcomes
- ➔ Relies heavily on answer set solver…this can hang!



TMP-RL Framework

Combines RL, TP, and MP!

Framework is defined as follows:

1. Inner loop is a task planner, motion planner, and reinforcement learner.
    a. Able to generate a **cheap** and **feasible** plan.
2. Plan sent to execution in outer loop; reinforcement learner performs value iteration from execution experience.
3. Learned values sent back to symbolic planner in inner loop.
4. Returns best possible task-motion plan!

Let's dive a little deeper.

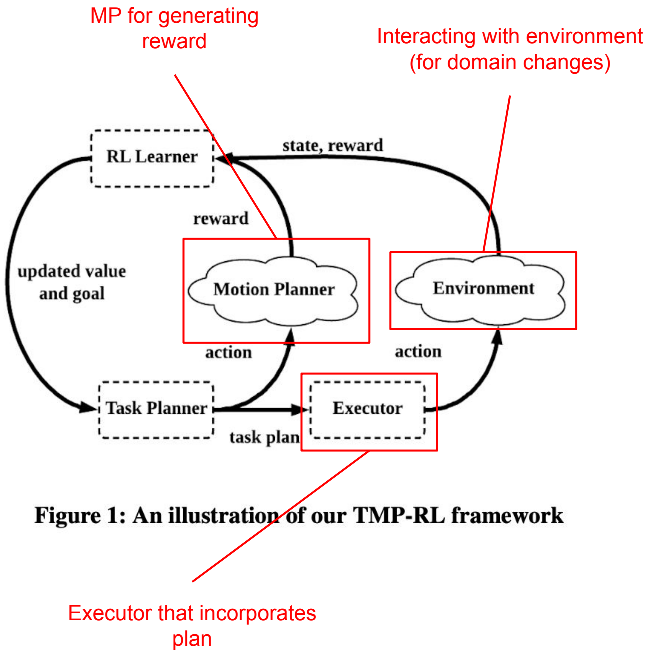


Figure 1: An illustration of our TMP-RL framework

Theoretical Assumptions

Motion planning, for optimal trajectory is defined as: $\xi^* = \mathrm{argmin}_{\xi \in \Xi} Len(\xi)$. This also works under the assumption that there is *no collisions.* Gradient descent and search algorithms (A* and Djikstra's) are used to generate paths.

For learning the underlying MDP, R-learning for finite horizon problems are adopted:

$$\rho^\pi(s) = \lim_{T\to\infty} \frac{\boxed{J^\pi_{\text{avg}}(s)}}{T} = \lim_{T\to\infty} \frac{1}{T} \boxed{\mathbb{E}[\sum_{t=0}^{T} r_t]}.$$

Model-free value iteration

$$J^\pi_{\text{avg}}(s) = \mathbb{E}[\sum_{t=0}^{T} r_t | s_0 = s]$$

The gain reward $\rho^\pi(s)$ is reaped from policy π to s.

Approximating a Continuous Domain

By using a mapping function, symbolic states are mapped into feasible poses in the continuous space.

Thus, if we are given a motion planning domain $\mathbb{D}^m$ and a task plan Π^τ for the problem $(\mathbb{D}^\tau, \mathbb{I}^\tau, \mathbb{G}^\tau)$, we can generate optimal trajectories on each navigation action:

$$\Pi^m = \bigcup_{\underbrace{\boxed{\langle s, a, s' \rangle \in \Pi^\tau}}_{\text{Motion planning states}}} \underbrace{\boxed{\xi(x, x')}}_{\text{Navigation actions}}$$

This defines the **optimal task plan** conditioned on a **motion plan**. This lies in the family of PSPACE-complete.

Let's combine this approach with reinforcement learning.

TMP and Learning Algorithms

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:     $\Pi^* \Leftarrow \Pi_t^\tau$
 4:     obtain a plan $\Pi_t^\tau \Leftarrow Plan(\mathbb{I}^\tau, \mathbb{G}^\tau, \mathbb{D}^\tau \cup P_t)$
 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:         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 $\boxed{(1).}$
16:     end for
17:     calculate quality of $\Pi_t^\tau$ by (3).
18:     update planning goal $G \Leftarrow (quality > quality_t(\Pi_t^\tau))$.
19:     update facts $P_t \Leftarrow \{\rho(s, a) = z : \langle s, a, s' \rangle \in \Pi, \rho_t^a(s) = z\}$
20:     $t \Leftarrow t + 1$
21: end while
```

Policy Plan Refinement

Reinforcement Learning Updates

Reference:

$$R_{t+1}(s_t, a_t) \xleftarrow{\alpha_t} r_t - \rho_t(s_t) + \max_a R_t(s_{t+1}, a)$$
$$\rho_{t+1}(s_t) \xleftarrow{\beta_t} r_t + \max_a R_t(s_{t+1}, a) - \max_a R_t(s_t, a), \quad \boxed{(1)}$$

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 $\boxed{\text{Algorithm 1 } \Pi_t^\tau}$ $\Leftarrow$
        $TMP(\mathbb{I}^\tau, \mathbb{G}^\tau, \mathbb{D}^\tau, f, \mathbb{D}^\mu, q_t, P_t)$.
 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 $\boxed{(1).}$
11:     end for
12:     calculate quality of $\Pi_t^\tau$ by (3).
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
```

Inner TMP loop

Quality and factset updates using state and action

Reference:

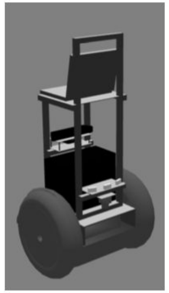
$$quality \geq quality(\Pi_t^\tau) \quad \boxed{(3)}$$

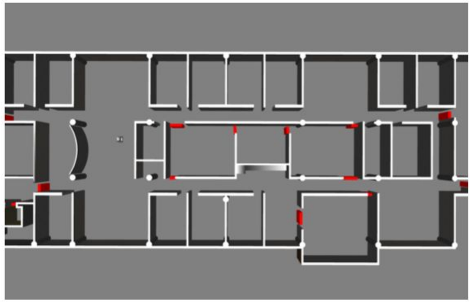
where

$$quality(\Pi_t^\tau) = \sum_{\langle s, a, s' \rangle \in \Pi_t^\tau} \rho(s, a)$$

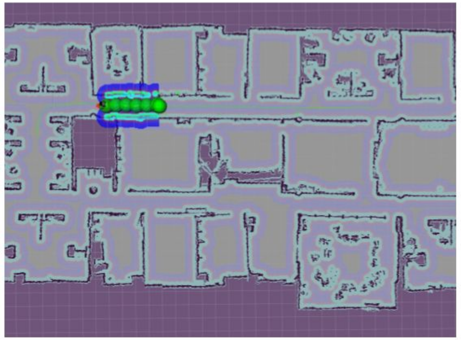
Testing out TMP-RL

Mobile robots were used as office assistants at a university in a simulation. These robots can find people, answer questions, and deliver objects. Thus, the robots need to navigate long distances *efficiently*.


(a) Simulated robot


(b) Gazebo simulation environment

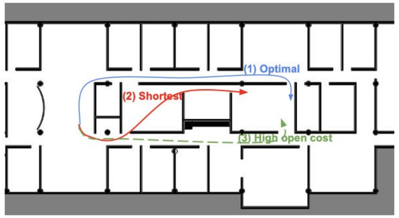
(c) Real robot


(d) Robot navigating in real environment

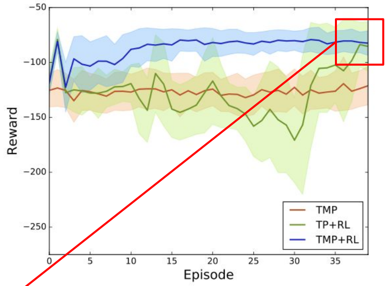
The dynamic environment makes this a real challenge (crowded hallways, closed doors, etc.) and forces the robot to reuse what it learns in order to stay robust.

Results

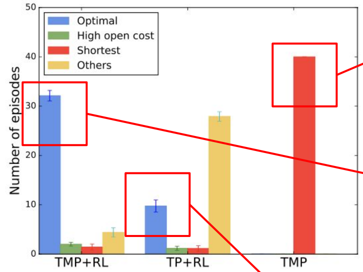
TMP-RL was tested against **TMP-only** and **TP-RL** (symbolic task planner).



(a) Three competitive plans for the task.



(b) Learning curves.



(c) Distribution of plan executions.

No exploration!

Best solution overall

Use of RL = convergence

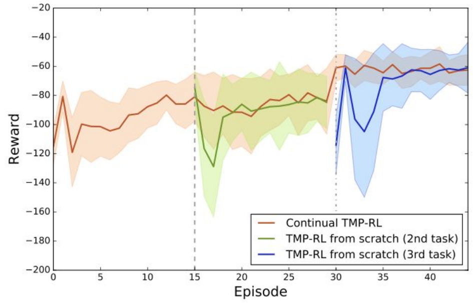
Needs more exploration

Plan	Task Plan Length	Motion Plan Length	Average Execution Time in the Real World
(1)	3	60.8	80.6
(2)	9	45.5	126.9
(3)	3	53.1	116.6

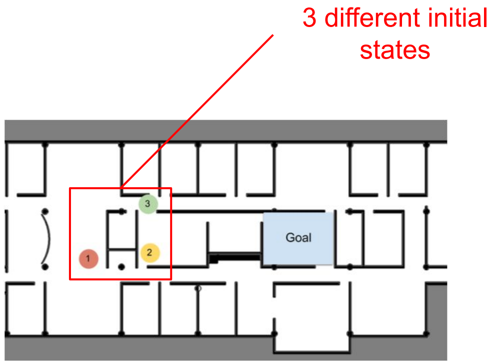
Table 1: Plan costs at different levels of abstraction.

Additional Results

Reward was defined as negative of execution time. 50 runs were conducted, with 40 episodes in each. Experiences in previous episodes caused different learning curves (see image to right).



(b) Learning curves of TMP-RL with continuous transfer vs. exploration from scratch.



(a) Extended experiment with three different initial states.

What if initial states were changed? **TMP-RL** was *still* able to generalize the learned values, regardless of scenario.

Robust and cheap!

Discussion of Results

TMP-RL was most optimal. **TP-RL** needed a lot more time ($$$) to explore learning plans and **TMP** always used the same plan without exploration.

In dynamic domains, **TMP-RL** is still the most resilient to change. If the answer set solver can't find a new best plan, the current best is returned, preventing time-outs and hangs.

With different starting points, **TMP-RL** is also able to generalize to new tasks, and use transfer learning to accelerate training from previous episodes. No need to learn from scratch!

Open Issues

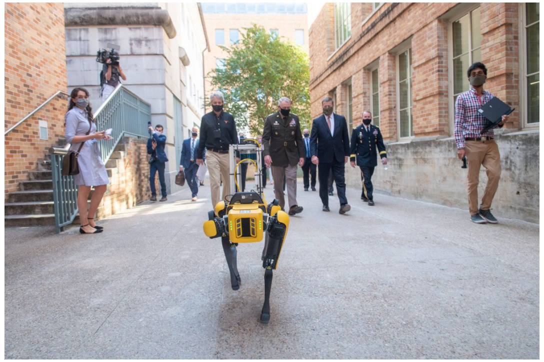
So, what’s left to worry about?

Long-term planning in crowded environments. Still an open challenge…

Transfer generalized information into new domains; how can we learn in a new domain?

What if there is no “best plan”? Maybe the robot needs to compromise and miss out on one objective to achieve the main goal…

Critique and Future Works

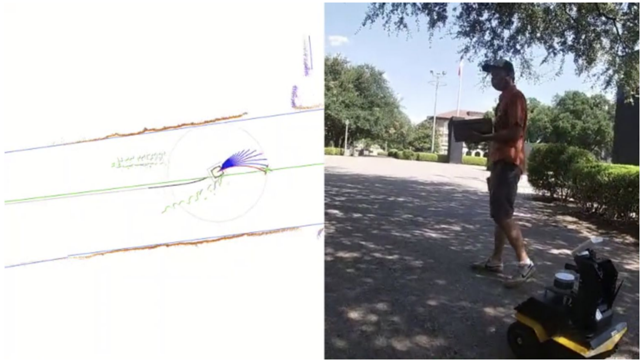


Allow mobile robot testing using **TMP-RL** and measure how well the framework can adapt to the environment.

Some sort of learning period, followed by a fully autonomous testing period where mobile robot needs to complete all tasks.

Test in the real world! All experiments in the paper were done in a *simulated environment* (meant to model the world).

But, test it out in real domain…UT's **AMRL** does mobile robot testing, could talk about this with them.



Extended Readings

PEORL: Integrating Symbolic Planning and Hierarchical RL for Robust Decision-Making. F. Yang et. al. (2018)

- ➔ Combining RL with symbolic planning
- ➔ Good for uncertainties in dynamic environments

What is Answer Set Programming? V. Lifschitz (2008)

- ➔ Introduction to ASP, how it can be used to generate symbolic solutions
- ➔ Idea of how to create fast and quick solutions by analyzing problem

SDRL: Interpretable and Data-efficient Deep Reinforcement Learning Leveraging Symbolic Planning. D. Lyu (2019)

- ➔ Ties in deep RL with the idea of a planner that breaks an objective into tasks
- ➔ Great segue into the pipeline for **TMP-RL** (objective → task → motion and task plan)

Integrated Task and Motion Planning in Belief Space. L. P. Kaelbling et. al. (2013)

- ➔ Symbolic operators with task-oriented perception using a belief space on a real PR2 robot

Summary

How can we build a mobile robot that **behaves intelligently and can make robust decisions**?

Can be very useful for integrating robots into dynamic environments. But, it can be **costly** and **challenging**!

Previous state-of-the-art *Task Planning* and *Motion Planning* algorithms addressed symbolic plan generation; *reinforcement learning* algorithms, separately, helped domain uncertainties.

TMP-RL framework incorporates *Task-Motion Planning* in the inner loop for improvised task plans, while the outer loop uses *Reinforcement Learning* to learn from execution and accommodate for any changes in the environment. This framework is exemplary! It outperforms existing **TMP** algorithms and reacts well to unseen domains.

Iterative improvements over the course of execution build the *best policy.* Combines the strengths of both worlds: high-quality plans from **TMP** and **RL** refines the plan, even applying transfer learning for new integrated scenarios. Overall, **TMP-RL** is a robust and effective iterative learning framework for mobile robots.

Any questions?

Thanks!