2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

2021 IEEE International Conference on Robotics and Automation (ICRA) | 978-1-7281-9077-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICRA48506.2021.9561819

Learning and Planning for Temporally Extended Tasks in Unknown Environments

Christopher Bradley[1], Adam Pacheck[2], Gregory J. Stein[3], Sebastian Castro[1], Hadas Kress-Gazit[2], and Nicholas Roy[1]

***Abstract*— We propose a novel planning technique for satisfying tasks specified in temporal logic in partially revealed environments. We define *high-level actions* derived from the environment and the given task itself, and estimate how each action contributes to progress towards completing the task. As the map is revealed, we estimate the cost and probability of success of each action from images and an encoding of that action using a trained neural network. These estimates guide search for the minimum-expected-cost plan within our model. Our learned model is structured to generalize across environments and task specifications without requiring retraining. We demonstrate an improvement in total cost in both simulated and real-world experiments compared to a heuristic-driven baseline.**

I. Introduction

Our goal is to enable an autonomous agent to find a minimum cost plan for multi-stage planning tasks when the agent's knowledge of the environment is incomplete, i.e., when there are parts of the world the robot has yet to observe. Imagine, for example, a robot tasked with extinguishing a small fire in a building. To do so, the agent could either find an alarm to trigger the building's sprinkler system, or locate a fire extinguisher, navigate to the fire, and put it out. Temporal logic is capable of specifying such complex tasks, and has been used for planning in fully known environments (e.g., [2–6]). However, when the environment is initially unknown to the agent, efficiently planning to minimize expected cost to solve these types of tasks can be difficult.

Planning in partially revealed environments poses challenges in reasoning about unobserved regions of space. Consider our firefighting robot in a building it has never seen before, equipped with perfect perception of what is in direct line of sight from its current position. Even with this perfect local sensing, the locations of any fires, extinguishers, and alarms may not be known. Therefore, the agent must envision all possible configurations of unknown space—including the position of obstacles—to find a plan that satisfies the task specification and, ideally, minimizes cost. We can model planning under uncertainty in this case as a Locally Observable Markov Decision Process (LOMDP) [7], a special class of Partially Observable Markov Decision Process (POMDP), which by definition includes our assumption of perfect range-limited perception. However, planning with such a model requires a distribution over possible environment configurations and high computational effort [8–10].

Since finding optimal policies for large POMDPs is intractable in general [10], planning to satisfy temporal logic specifications in full POMDPs has so far been limited to relatively small environments or time horizons [11, 12]. Thus, many approaches for solving tasks specified with temporal logic often make simplifying assumptions about known and unknown space, or focus on maximizing the probability a specification is never violated [13–17]. Accordingly, such strategies can result in sub-optimal plans, as they do not consider the cost of planning in unknown parts of the map.

A number of methods use learned policies to minimize the cost of completing tasks specified using temporal logic [18–22]. However, due to the complexity of these tasks, many are limited to fully observable small grid worlds [18–21] or short time-horizons [22]. Moreover, for many of these approaches, changing the specification or environment requires retraining the system. Recent work by Stein et al. [23] uses supervised learning to predict the outcome of acting through unknown space, but is restricted to goal-directed navigation.

To address the challenges of planning over a distribution of possible futures, we introduce Partially Observable Temporal Logic Planner (PO-TLP). PO-TLP enables real-time planning for tasks specified in Syntactically Co-Safe Linear Temporal Logic (scLTL) [24] in unexplored, arbitrarily large environments, using an abstraction based on a given task and partially observed map. Since completing a task may not be possible in known space, we define a set of dynamic *high-level actions* based on transitions between states in our abstraction and available *subgoals* in the map—points on the boundaries between free and unknown space. We then approximate the full LOMDP model such that actions either successfully make their desired transitions or fail, splitting future beliefs into two classes and simplifying planning. We train a neural network from images to predict the cost and outcome of each action, which we use to guide a variant of Monte-Carlo Tree Search (PO-UCT [25]) to find the best high-level action for a given task and observation.

Our model learns directly from visual input and can generalize across novel environments and tasks without retraining. Furthermore, since our agent continually replans as more of the environment is revealed, we ensure both that the specification is never violated and that we find the optimal trajectory if the environment is fully known. We apply PO-TLP to multiple tasks in simulation and the real world, showing improvement over a heuristic-driven baseline.

This work is supported by ONR PERISCOPE MURI N00014-17-1-2699 and the Toyota Research Institute (TRI). [1] CSAIL, MIT, Cambridge, MA 02142, USA ({cbrad, sebacf, nickroy}@mit.edu). [2] Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14850, USA ({akp84, hadaskg}@cornell.edu). [3] Computer Science Department, George Mason University, Fairfax, VA 22030, USA (gjstein@gmu.edu).

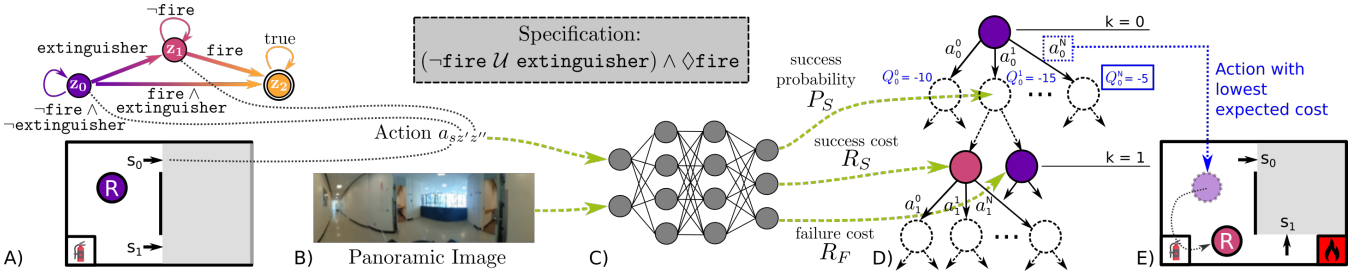

Fig. 1. **Overview.** (A) Given a task (e.g., $(\neg\texttt{fire}\ \mathcal{U}\ \texttt{extinguisher}) \wedge \Diamond\texttt{fire}$), we compute a DFA using [1] to represent the high-level steps needed to accomplish the task. The robot operates in a partially explored environment with *subgoals* between observed (white) and unexplored (gray) space, and regions labeled with propositions relevant to the task, e.g., extinguisher and fire. (B) We define *high-level actions* with a subgoal (e.g., s_0), a DFA state z' (e.g., z_0) which the robot must be in when arriving at the subgoal, and a state z'' (e.g., z_1) which the agent attempts to transition to in unknown space. (C) For each possible action, we estimate its probability of success (P_S) and costs of success (R_S) and failure (R_F) using a neural network that accepts a panoramic image centered on a subgoal, the distance to that subgoal, and an encoding of the transition from z' to z'' (Sec. IV-A). (D) We estimate the expected cost of each action with these estimates of P_S, R_S, and R_F using PO-UCT search. (E) The agent selects an action with the lowest expected cost, and moves along the path defined by that action, meanwhile receiving new observations, updating its map, and re-planning.

II. Preliminaries

Labeled Transition System (LTS): We use an LTS to represent a discretized version of the robot's environment. An LTS $\mathcal{T}$ is a tuple $(X, x_0, \delta_\mathcal{T}, w, \Sigma, l)$, where: X is a discrete set of states of the robot and the world, $x_0 \in X$ is the initial state, $\delta_\mathcal{T} \subseteq X \times X$ is the set of possible transitions between states, the weight $w : (x_i, x_j) \rightarrow \mathbb{R}^+$ is the cost incurred by making the transition (x_i, x_j), Σ is a set of *propositions* with $\sigma \in \Sigma$, and the labeling function $l : X \rightarrow 2^\Sigma$ is used to assign which propositions are true in state $x \in X$. An example LTS is shown in Fig. 2A where $\Sigma = \{\texttt{fire}, \texttt{extinguisher}\}$, $l(x_3) = \{\texttt{extinguisher}\}$ and $l(x_4) = \emptyset$. States $x \in X$ refer to physical locations, and labels like `fire` or `extinguisher` indicate whether there is a fire or extinguisher (or both) at that location. A finite trajectory τ is a sequence of states $\tau = x_0 x_1 \ldots x_n$ where $(x_i, x_{i+1}) \in \delta_\mathcal{T}$ which generates a finite word $\omega = \omega_0 \omega_1 \ldots \omega_n$ where each letter $\omega_i = l(x_i)$.

Syntactically Co-Safe Linear Temporal Logic (scLTL): To specify tasks, we use scLTL, a fragment of Linear Temporal Logic (LTL) [24]. Formulas are written over a set of atomic propositions Σ with Boolean (negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$)) and temporal (next ($\bigcirc$), until ($\mathcal{U}$), eventually ($\Diamond$)) operators. The syntax is:

$$\varphi := \sigma \mid \neg\sigma \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi\ \mathcal{U}\ \varphi \mid \Diamond\varphi,$$

where $\sigma \in \Sigma$, and φ is an scLTL formula. The semantics are defined over infinite words $\omega^{\text{inf}} = \omega_0\omega_1\omega_2 \ldots$ with letters $\omega_i \in 2^\Sigma$. Intuitively, $\bigcirc\varphi$ is satisfied by a given ω^{inf} at step i if φ is satisfied at the next step $(i+1)$, $\varphi_1 \mathcal{U} \varphi_2$ is satisfied if φ_1 is satisfied at every step until φ_2 is satisfied, and $\Diamond\varphi$ is satisfied at step i if φ is satisfied at some step $j \geq i$. For the complete semantics of scLTL, refer to Kupferman and Vardi [24]. For example, $(\neg\texttt{fire}\ \mathcal{U}\ \texttt{extinguisher}) \wedge \Diamond\texttt{fire}$ specifies the robot should avoid fire until reaching the extinguisher and eventually go to the fire.

Deterministic Finite Automaton (DFA): A DFA, constructed from an scLTL specification, is a tuple $\mathcal{D}_\varphi = (Z, z_0, \Sigma, \delta_\mathcal{D}, F)$. A DFA is composed of a set of states, Z, with an initial state $z_0 \in Z$. The transition function $\delta_\mathcal{D} : Z \times 2^\Sigma \rightarrow Z$ takes in the current state, $z \in Z$, and a letter $\omega_i \in 2^\Sigma$, and returns the next state $z \in Z$ in the DFA. The DFA has a set of accepting states, $F \subseteq Z$, such that if the execution of the DFA on a finite word $\omega = \omega_0\omega_1 \ldots \omega_n$ ends in a state $z \in F$, the word belongs to the language of the DFA. While in general LTL formulas are evaluated over infinite words, the truth value of scLTL can be determined over finite traces. Fig. 1A shows the DFA for $(\neg\texttt{fire}\ \mathcal{U}\ \texttt{extinguisher}) \wedge \Diamond\texttt{fire}$.

Product Automaton (PA): A PA is a tuple $(P, p_0, \delta_P, w_P, F_P)$ which captures this combined behavior of the DFA and LTS. The states $P = X \times Z$ keep track of both the LTS and DFA states, where $p_0 = (x_0, z_0)$ is the initial state. A transition between states is possible iff the transition is valid in both the LTS and DFA, and is defined by $\delta_P = \{(p_i, p_j) \mid (x_i, x_j) \in \delta_\mathcal{T}, \delta_\mathcal{D}(z_i, l(x_j)) = z_j\}$. When the robot moves to a new state $x \in X$ in the LTS, it transitions to a state in the DFA based on the label $l(x)$. The weights on transitions in the PA are the weights in the LTS $(w_P(p_i, p_j) = w(x_i, x_j))$. Accepting states in the PA, F_P, are those states with accepting DFA states ($F_P = X \times F$).

Partially/Locally Observable Markov Decision Process: We model the world as a Locally Observable Markov Decision Process (LOMDP) [7]—a Partially Observable Markov Decision Process (POMDP) [26] where space within line of sight from our agent is fully observable—formulated as a belief MDP, and written as the tuple $(B, \mathcal{A}, P, R)$. In a belief MDP, B represents the belief over states in an MDP, which in our case are the states from the PA defined by a given task and environment. We can think of each belief state as distributions over three entities $b_t = \{b_\mathcal{T}, b_x, b_z\}$: the environment (abstracted as an LTS) $b_\mathcal{T}$, the agent's position in that environment b_x, and the agent's progress through the DFA defined by the task b_z. With our LOMDP formulation, we collapse b_x and b_z to point estimates, and consider any states in $\mathcal{T}$ that have been seen by the robot as fully observable, allowing us to treat $b_\mathcal{T}$ as a labeled occupancy grid that is revealed as the robot explores. $\mathcal{A}$ is the set of available actions from a given state in the PA, which at the lowest level of abstraction are the feasible transitions given

by δ_P. $P(b_{t+1}|b_t, a_t)$ and $R(b_{t+1}, b_t, a_t)$ are the transition probability and instantaneous cost, respectively, of taking an action a_t to get from a belief b_t to b_{t+1}.

III. Planning in Unknown Environments with co-safe LTL Specifications

Our objective is to minimize the total expected cost of satisfying an scLTL specification in partially revealed environments. Using the POMDP model, we can represent the expected cost of taking an action using a belief-space variant of the Bellman equation [26]:

$$Q(b_t, a_t) = \sum_{b_{t+1}} P(b_{t+1}|b_t, a_t)\Big[R(b_{t+1}, b_t, a_t) + \min_{a_{t+1}\in\mathcal{A}(b_{t+1})} Q(b_{t+1}, a_{t+1})\Big], \quad (1)$$

where $Q(b_t, a_t)$ is the belief-action cost of taking action a_t given belief b_t. This equation can, in theory, be solved to find the optimal action for any belief. However, given the size of the environments we are interested in, directly updating the belief in evaluation of Eq. (1) is intractable for two reasons. First, due to the *curse of dimensionality*, the size of the belief grows exponentially with the number of states. Second, by the *curse of history*, the number of iterations needed to solve Eq. (1) grows exponentially with the planning horizon.

A. Defining the Set of High-Level Actions

To overcome the challenges associated with solving Eq. (1), we first identify a set of discrete, *high-level actions*, then build off an insight from Stein et al. [23]: that we can simplify computing the outcome of each action by splitting future beliefs into two classes—futures where a given action is successful, and futures where it is not. Note that this abstraction over the belief space does not exist in general, and is enabled by our assumption of perfect local perception.

To satisfy a task specification, our agent must take actions to transition to an accepting state in the DFA. For example, given the specification $(\neg\texttt{fire}\ \mathcal{U}\ \texttt{extinguisher}) \wedge \Diamond\texttt{fire}$ as in Fig. 1 and 2, the robot must first retrieve the extinguisher and then reach the fire. If the task cannot be completed entirely in the known space, the robot must act in areas that it has not yet explored. As such, our action set is defined by *subgoals* $\mathbb{S}$—each a point associated with a contiguous boundary between free and unexplored space—and the DFA. Specifically, when executing an action, the robot first plans through the PA in known space to a subgoal, and then enters the unexplored space beyond that subgoal to attempt to transition to a new state in the DFA.

For belief state b_t, action $a_{sz'z''}$ defines the act of traveling from the current state in the LTS x to reach subgoal $s \in \mathbb{S}_t$ at a DFA state z', and then attempting to transition to DFA state z'' in unknown space. Our newly defined set of possible actions from belief state b_t is: $\mathcal{A}(b_t) = \{a_{sz'z''} \mid s \in \mathbb{S}_t, z' \in Z_{\text{reach}}(b_t, s), z'' \in Z_{\text{next}}(z')\}$ where $Z_{\text{reach}}(b_t, s)$ is the set of DFA states that can be reached while traveling in known space in the current belief b_t to subgoal $s \in \mathbb{S}_t$, and $Z_{\text{next}}(z') = \{z'' \in Z \mid \exists\omega_i \in 2^\Sigma \text{ s.t. } z'' = \delta_{\mathcal{D}}(z', \omega_i)\}$ is the set of DFA states that can be visited in one transition from z'. Fig. 2B illustrates an example of available high-level actions.

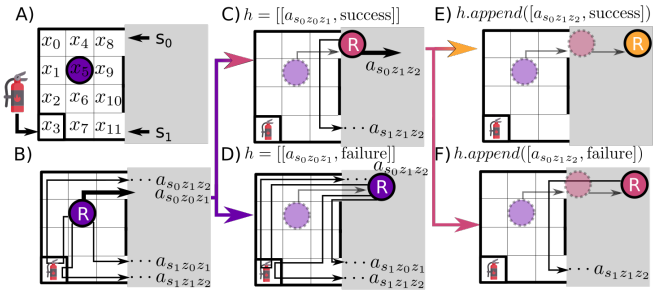

Fig. 2. **High-level actions.** The robot ("R") is in a partially explored environment abstracted into an LTS with two subgoals s_0 and s_1 (A). As the robot considers high-level actions at different stages of its plan (B), its color indicates belief of the DFA state (according to Fig. 1A). In this rollout, the robot considers action $a_{s_0z_0z_1}$, and outcomes where it succeeds (C), and fails (D). If successful, the robot has two actions available and considers $a_{s_0z_1z_2}$, which in turn could succeed (E) or fail (F).

When executing action $a_{sz'z''}$, the robot reaches subgoal s in DFA state z', accumulating a cost $D(b_t, a_{sz'z''})$ which is computed using Dijkstra's algorithm in the known map. Once the robot enters the unknown space beyond the subgoal, the action has some probability P_S of successfully transitioning from z' to z'', denoted as: $P_S(b_t, a_{sz'z''}) \equiv \sum_{b_{t+1}} P(b_{t+1}|a_{sz'z''}, b_t)\mathbb{I}[Z(b_{t+1}) = z'']$, where $Z(b_{t+1})$ refers to b_z at the next time step and $\mathbb{I}[Z(b_{t+1}) = z'']$ is an indicator function for belief states where the agent has reached the DFA state z''. Each action has an expected cost of success $R_S(b_t, a_{sz'z''})$ such that $R_S(b_t, a_{sz'z''}) + D(b_t, a_{sz'z''}) \equiv \frac{1}{P_S}\sum_{b_{t+1}} P(b_{t+1}|b_t, a_{sz'z''})R(b_{t+1}, b_t, a_{sz'z''})\mathbb{I}[Z(b_{t+1}) = z'']$ and expected cost of failure $R_F(b_t, a_{sz'z''})$, which is equivalently defined for $Z(b_{t+1}) \neq z''$ and normalized with $\frac{1}{1-P_S}$. By estimating these values via learning (discussed in Sec. IV), we can express the expected instantaneous cost of an action as $\sum_{b_{t+1}} P(b_{t+1}|b_t, a_{sz'z''})R(b_{t+1}, b_t, a_{sz'z''}) = D + P_S R_S + (1 - P_S)R_F$.

B. Estimating Future Cost using High-Level Actions

We now examine planning to satisfy a specification with minimum cost using these actions, remembering to consider both the case where an action succeeds—the robot transitions to the desired state in the DFA—and when it fails. We refer to a complete simulated trial as a *rollout*. Since we cannot tractably update and maintain a distribution over maps during a rollout, we instead keep track of the *rollout history* $h = [[a_0, o_0], \ldots, [a_n, o_n]]$ (a sequence of high-level actions a_i considered during planning and their simulated respective outcomes $o_i = \{\texttt{success}, \texttt{failure}\}$). Recall that, during planning, we assume that the agent knows its position x in known space $\mathcal{T}_{\text{known}}$. Additionally, conditioned on whether an action $a_{sz'z''}$ is simulated to succeed or fail, we assume no uncertainty over the resulting DFA state, collapsing b_z to z'' or z', respectively. Therefore, for a given rollout history, the agent knows its position in known space and its state in the DFA. These assumptions, while not suited to solving POMDPs in general, fit within the LOMDP model.

During a rollout, the set of available future actions is informed by actions and outcomes already considered in h. For example, if we simulate an action in a rollout, and it fails, we should not consider that action a second

Algorithm 1 PO-TLP

Function PO-TLP(θ): // θ: Network parameters
  $b \leftarrow \{b_{\mathcal{T}_0}, b_{x_0}, b_{z_0}\}$, $Img \leftarrow Img_0$
  while *True* **do**
    if $b_z \in F$ **then** // In accepting state
      return *SUCCESS*
    $a^*_{sz'z''} \leftarrow$ HIGHLEVELPLAN($b, Img, \theta, \mathcal{A}(b)$)
    b, $Img \leftarrow$ ACTANDOBSERVE($a^*_{sz'z''}$)
Function HIGHLEVELPLAN(b, *Img*, θ, $\mathcal{A}(b)$):
  for $a \in \mathcal{A}(b)$ **do**
    $\underline{a.P_S}, \underline{a.R_S}, \underline{a.R_F} \leftarrow$ ESTPROPS($b, a, Img \mid \theta$)
  $a^*_{sz'z''} \leftarrow$ PO-UCT($b, \mathcal{A}(b)$)
  return $a^*_{sz'z''}$

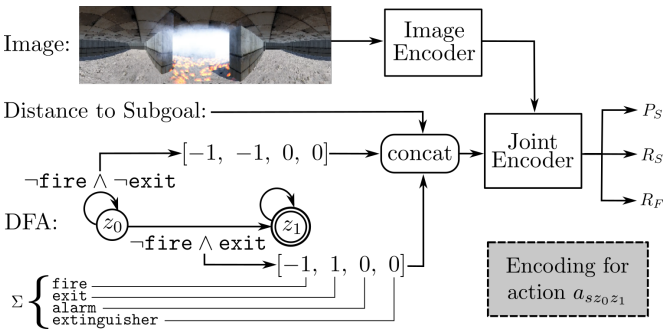

Fig. 3. **Neural network inputs and outputs.** Estimating P_S, R_F, and R_S for action $a_{sz_0z_1}$: attempting to reach an exit while avoiding fire.

time. Conversely, we know a successful action would succeed if simulated again in that rollout. In Fig. 2F, the robot imagines its first action $a_{s_0z_0z_1}$ succeeds, while its next action $a_{s_0z_1z_2}$ fails, making the rollout history $h = [[a_{s_0z_0z_1}, \texttt{success}], [a_{s_0z_1z_2}, \texttt{failure}]]$. When considering the next step of this rollout, the robot knows it can always find an extinguisher beyond s_0, and there is no fire beyond s_0. To track this information during planning, we define a *rollout history-augmented belief* $b_h = \{\mathcal{T}_{\text{known}}, x, z, h\}$, which augments the belief with the actions and outcomes of the rollout up to that point. To reiterate, we maintain the history-augmented belief b_h only during planning, to avoid the complexity of maintaining a distribution over possible future maps from possible future observations during rollout.

Using b_h, we also define a *rollout history-augmented action set* $\mathcal{A}(b_h)$, in which actions known to be impossible based on h are pruned, and with it, a *rollout history-augmented success probability* $P_S(b_h, a_{sz'z''})$ which is identically one for actions known to succeed. Furthermore, because high-level actions involve entering unknown space, instead of explicitly considering the distribution over possible robot states, we define a *rollout history-augmented distance* function $D(b_h, a_{sz'z''})$, which takes into account physical location as a result of taking the last action in b_h. If an action leads to a new subgoal ($s_{t+1} \neq s_t$), the agent accumulates the success cost R_S of the previous action if that action was simulated to succeed and the failure cost R_F if it was not.

By planning with b_h, the future expected reward can be written so that it no longer directly depends on the full future belief state b_{t+1}, allowing us to approximate it as follows:

$$\sum_{b_{t+1}} P(b_{t+1}|b_t, a_{sz'z''}) \min_{a' \in \mathcal{A}(b_{t+1})} Q(b_{t+1}, a') \approx P_S(b_h, a_{sz'z''}) \min_{a_S \in \mathcal{A}(b_{h_S})} Q(b_{h_S}, a_S) + [1 - P_S(b_h, a_{sz'z''})] \min_{a_F \in \mathcal{A}(b_{h_F})} Q(b_{h_F}, a_F), \quad (2)$$

where $h_S = h.\texttt{append}([a_{sz'z''}, \texttt{success}])$ is the rollout history conditioned on a successful outcome (and h_F is defined similarly for failed outcomes).

Our high-level actions and rollout history allow us to approximate Eq. (1) with Eq. (3), a finite horizon problem. Given estimates of P_S, R_S, and R_F—which are learned as discussed in Sec. IV—we avoid explicitly summing over the distribution of all possible PAs, thereby reducing the computational complexity of solving for the best action.

C. Planning with High-Level Actions using PO-UCT

Eq. (3) demonstrates how expected cost can be computed exactly using our high-level actions, given estimated values of P_S, R_S, and R_F. However, considering every possible ordering of actions as in Eq. (3) still involves significant computational effort—exponential in both the number of subgoals and the size of the DFA. Instead, we adapt Partially Observable UCT (PO-UCT)[25], a generalization of Monte-Carlo Tree-Search (MCTS) which tracks histories of actions and outcomes, to select the best action for a given belief using sampling. The nodes of our search tree correspond to belief states b_h, and actions available at each node are defined according to the rollout history as discussed in Sec. III-B. At each iteration, from this set we sample an action and its outcome according to the Bernoulli distribution parameterized by P_S, and accrue cost by R_S or R_F.

This approach prioritizes the most promising branches of the search space, avoiding the cost of enumerating all states. By virtue of being an anytime algorithm, PO-UCT also enables budgeting computation time, allowing for faster online execution as needed on a real robot. Once our agent has searched for the action with lowest expected cost, $a^*_{sz'z''}$, it generates a motion plan through space to the subgoal associated with the chosen action. While moving along this path, the agent receives new observations of the world, updates its map, and replans (see Fig. 1 and Algorithm 1).

IV. Learning Transition Probabilities and Costs

To plan with our high-level actions, we rely on values for P_S, R_S, and R_F for arbitrary actions $a_{sz'z''}$. Computing

$$Q(b_h, a_{sz'z''}) = D(b_h, a_{sz'z''}) + \underline{P_S(b_h, a_{sz'z''})} \times \left[\underline{R_S(b_h, a_{sz'z''})} + \min_{a_S \in \mathcal{A}(b_{h_S})} Q(b_{h_S}, a_S)\right] + \left[1 - \underline{P_S(b_h, a_{sz'z''})}\right] \times \left[\underline{R_F(b_h, a_{sz'z''})} + \min_{a_F \in \mathcal{A}(b_{h_F})} Q(b_{h_F}, a_F)\right] \quad (3)$$

Underline denotes terms we estimate via learning

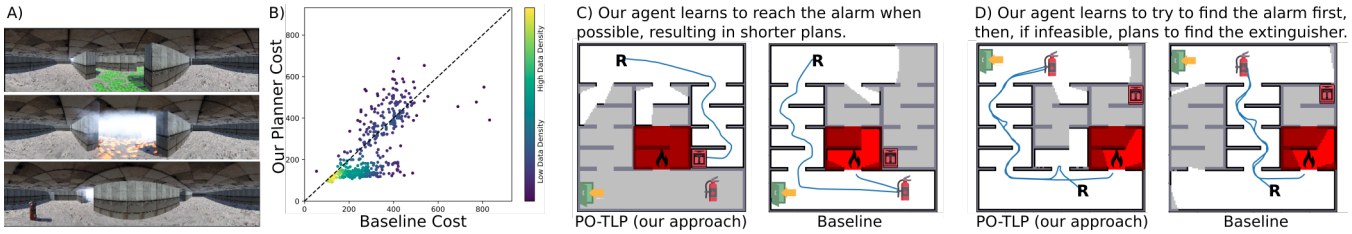

Fig. 4. A comparison between our planner and the baseline for 500 simulated trials in the Firefighting environment with the specification $(\neg\texttt{fire}\ \mathcal{U}\ \texttt{alarm}) \vee ((\neg\texttt{fire}\ \mathcal{U}\ \texttt{extinguisher}) \wedge \Diamond\texttt{fire})$. The robot ("R") learns to associate green tiling with the alarm, and hallways emanating white smoke with fire (A), leading to a 15% improvement for total net cost for this task (B). Our agent learns it is often advantageous to search for the alarm (C), so in cases where the alarm is reachable, we generally outperform the baseline (highlighted by the cluster of points in the lower half of B). Our method is occasionally outperformed when the alarm can't be reached (D), though we perform better in the aggregate and always satisfy the specification.

these values explicitly from the belief (as defined in Sec. III-A) is intractable, so we train a neural network to estimate them from visual input and an encoding of the action.

A. Encoding a Transition

A successful action $a_{sz'z''}$ results in the desired transition in the DFA from z' to z'' occurring in unknown space. However, encoding actions directly using DFA states prevents our network from generalizing to other specifications without retraining. Instead, we use an encoding that represents formulas over the set of propositions Σ in negative normal form [27] over the truth values of Σ, which generalizes to any specification written with Σ in similar environments.

To progress from z' to z'' in unknown space, the robot must travel such that it remains in z' until it realizes changes in proposition values that allow it to transition to z''. We therefore define two n-element feature vectors $[\phi(z', z'), \phi(z', z'')]$ where $\phi \in \{-1, 0, 1\}^n$, which serve as input to our neural network. For the agent to stay in z', if the ith element in $\phi(z', z')$ is 1, the corresponding proposition must be true at all times; if it is -1, the proposition must be false; and if it is 0, the proposition has no effect on the desired transition. The values in $\phi(z', z'')$ are defined similarly for the agent to transition from z' to z''. Fig. 3 illustrates this feature vector for a task specification example.

B. Network Architecture and Training

Our network takes as input a 128×512 RGB panoramic image centered on a subgoal, the scalar distance to that subgoal, and the two n-element feature vectors ϕ describing the transition of interest, as defined in Sec. IV-A. The input image is first passed through 4 convolutional layers, after which we concatenate the feature vectors and the distance to the subgoal to each element (augmenting the number of channels accordingly), and continue encoding for 8 more convolutional layers. Finally, the encoded features are passed through 5 fully connected layers, and the network outputs the properties required for Eq. (3)—P_S, R_S, and R_F. We train our network with the same loss function as in [23].

To collect training data, we navigate through environments using an autonomous, heuristic-driven agent in simulation, and teleoperation in the real world. We assume the agent has knowledge of propositions in its environments, so it can generate the feature vectors that encode actions for subgoals it encounters. As the robot travels, we periodically collect images and the true values of P_S (either 0 or 1), R_S, and R_F for each potential action from the underlying map.

V. Experiments

We perform experiments in simulated and real-world environments, comparing our approach with a baseline derived from Ayala et al. [13]. The baseline solves similar planning-under-uncertainty problems using boundaries between free and unknown space to define actions, albeit with a non-learned selection procedure and no visual input. Specifically, we compare the total distances traveled using each method.

A. Firefighting Scenario Results

Our first environment is based on the firefighting robot example, simulated with the Unity game engine [28] and shown in Fig. 4. The robot is randomly positioned in one of two rooms, and the extinguisher and exit in the other. One of three hallways connecting the rooms is randomly chosen to be a dead end with an alarm at the end of it, and is visually highlighted by a green tiled floor. A hallway (possibly the same one) is chosen at random to contain a fire, which blocks passage and emanates white smoke. Our network learns to associate the visual signals of green tiles and smoke to the hallways containing the alarm and the fire, respectively.

We run four different task specifications—using the same network without retraining to demonstrate its reusability:

1) $(\neg\texttt{fire}\ \mathcal{U}\ \texttt{alarm}) \vee ((\neg\texttt{fire}\ \mathcal{U}\ \texttt{extinguisher}) \wedge \Diamond\texttt{fire})$: avoid the fire until the alarm is found, or avoid the fire until the extinguisher is found, then find the fire.
2) $\neg\texttt{fire}\ \mathcal{U}\ (\texttt{alarm} \wedge \Diamond\texttt{exit})$: avoid the fire until the alarm is found, then exit the building.

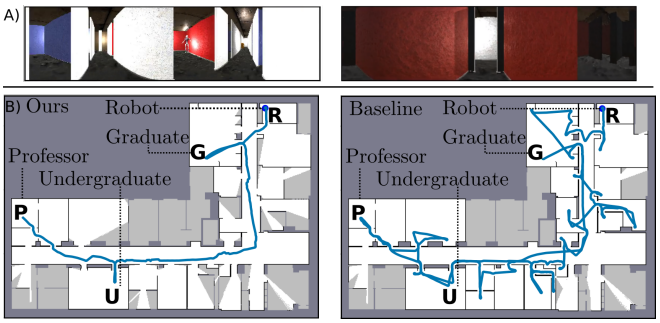

Fig. 5. A) Visual scenes from our Delivery scenario in simulation. The rooms which can contain professors, graduate students, and undergraduates are colored differently and illuminated when occupied. B) A comparison between our approach (left) and the baseline (right) for one of several simulated trials for the task $\Diamond\texttt{professor} \wedge \Diamond\texttt{grad} \wedge \Diamond\texttt{undergrad}$.

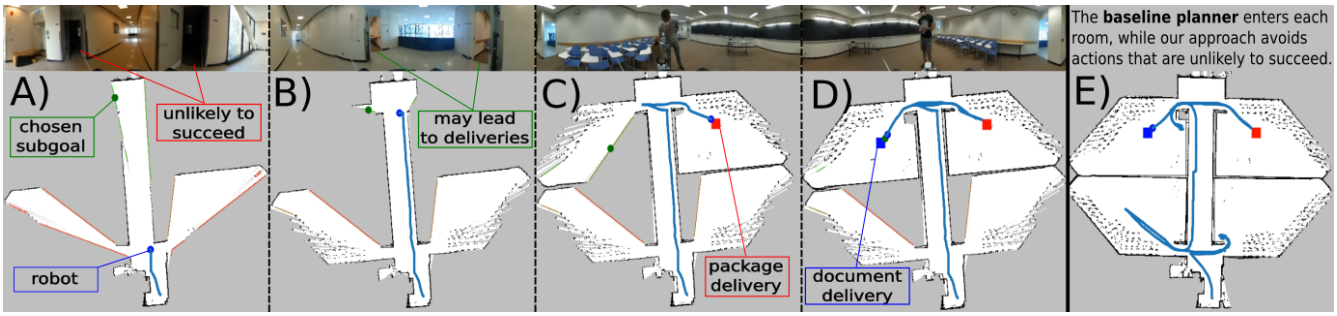

Fig. 6. A comparison between our approach (A-D) and the baseline (E) for our real-world Delivery scenario. Our agent (blue dot) correctly predicts actions likely to fail (e.g., dark rooms in A) and succeed (e.g., completing a delivery in an illuminated room in B). Once the robot identifies delivery actions that lead to DFA transitions in known space, it executes them (C-D). Conversely, the baseline fully explores space before completing the task (E).

3) $(\neg\texttt{fire}\ \mathcal{U}\ \texttt{extinguisher}) \wedge \Diamond(\texttt{fire} \wedge \Diamond\texttt{exit})$: avoid the fire until the extinguisher is found, then put out the fire and exit the building.
4) $\neg\texttt{fire}\ \mathcal{U}\ \texttt{exit}$: avoid the fire and exit the building.

Over ~3000 trials across different simulated environments and these four specifications, we demonstrate improvement for our planner over the baseline (see Table I). Fig. 4 gives a more in depth analysis of the results for specification 1.

TABLE I

	Average Cost			Percent Savings	
Spec	Known Map	Baseline	Ours	Net Cost	Per Trial (S.E.)
1	187.0	264.6	226.9	15%	14.4% (1.3)
2	392.1	696.0	461.2	34%	28.4% (1.0)
3	364.3	539.3	471.3	13%	6.1% (1.3)
4	171.2	269.1	203.3	25%	14.6% (1.0)

B. Delivery Scenario Results in Simulation

We scale our approach to larger simulated environments using a corpus of academic buildings containing *labs*, *classrooms*, and *offices*, all connected by hallways. Our agent must deliver packages to three randomly placed individuals in these environments: one each of a professor, graduate student, and undergraduate, using the specification $\Diamond\texttt{professor} \wedge \Diamond\texttt{grad} \wedge \Diamond\texttt{undergrad}$. Professors are located randomly in offices, graduate students in labs, and undergraduates in classrooms, which have differently colored walls in simulation. Rooms that are occupied have the lights on whereas other rooms are not illuminated.

Our agent is able to learn from these visual cues to navigate more efficiently. Over 80 simulations across 10 environments, the mean per-trial cost improvement of our agent compared with the baseline is 13.5% (with 6.1% standard error). Our net cost savings, summed over all trials, is 7.8%. Fig. 5 illustrates our results.

C. Delivery Scenario Results in the Real World

We extend our Delivery scenario to the real world using a Toyota Human Support Robot (HSR) [29] with a head-mounted panoramic camera [30] in environments with multiple rooms connected by a hallway. The robot must deliver a package and a document to two people, either unordered $(\Diamond\texttt{DeliverDocument} \wedge \Diamond\texttt{DeliverPackage})$ or in order $(\Diamond(\texttt{DeliverDocument} \wedge \Diamond\texttt{DeliverPackage}))$. As in simulation, rooms are illuminated only if occupied.

We ran 5 trials for each planner spanning both specifications and 3 different target positions in a test environment different from the one used to collect training data. We show improved performance over the baseline in all cases with a mean per-trial cost improvement of 36.6% (6.2% standard error), and net cost savings, summed over all trials, of 36.5%. As shown in Fig. 6, the baseline enters the nearest room regardless of external signal, while our approach prioritizes illuminated rooms, which are more likely to contain people.

VI. Related Works

Temporal logic synthesis has been used to generate provably correct controllers, although predominantly in fully known environments [2–6]. Recent work has looked at satisfying LTL specifications under uncertainty beyond LOMDPs [11, 12], yet these works are restricted to small state spaces due to the general nature of the POMDPs they handle. Other work has explored more restricted sources of uncertainty, such as scenarios where tasks specified in LTL need to be completed in partially explored environments [13–17] or by robots with uncertainty in sensing, actuation, or the location of other agents [31–34]. When these robots plan in partially explored environments, they take the best possible action given the known map, but either ignore or make naive assumptions about unknown space; conversely, we use learning to incorporate information about unexplored space.

To minimize the cost of satisfying LTL specifications, other recent works have used learning-based approaches [18, 20, 21, 36], yet these methods are limited to relatively small, fully observable environments. Sadigh et al. [19] and Fu and Topcu [20] apply learning to unknown environments and learn the transition matrices for MDPs built from LTL specifications. However, these learned models do not generalize to new specifications and are demonstrated on relatively small grid worlds. Paxton et al. [22] introduce uncertainty during planning, but limit their planning horizon to 10 seconds, which is insufficient for the specifications explored here. Carr et al. [37] synthesize a controller for verifiable planning in POMDPs using a recurrent neural network, yet are limited to planning in small grid worlds.

VII. Conclusion and Future Work

In this work, we present a novel approach to planning to solve scLTL tasks in partially revealed environments. Our approach learns from raw sensor information and generalizes to new environments and task specifications without the need to retrain. We hope to extend our methods to real-world planning domains that involve manipulation and navigation.

REFERENCES

[1] A. Duret-Lutz et al. "Spot 2.0 — a framework for LTL and ω-automata manipulation". In: *ATVA*. Springer, 2016.

[2] H. Kress-Gazit et al. "Synthesis for Robots: Guarantees and Feedback for Robot Behavior". In: *Annual Review of Control, Robotics, and Autonomous Systems* (2018).

[3] G. E. Fainekos et al. "Temporal logic motion planning for dynamic robots". In: *Automatica* (2009).

[4] A. Bhatia et al. "Sampling-based motion planning with temporal goals". In: *ICRA*. 2010.

[5] B. Lacerda et al. "Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications". In: *IROS*. 2014.

[6] S. L. Smith et al. "Optimal path planning for surveillance with temporal-logic constraints". In: *IJRR* 30.14 (2011).

[7] M. Merlin et al. "Locally Observable Markov Decision Processes". In: *ICRA 2020 Workshop on Perception, Action, Learning*. 2020.

[8] L. P. Kaelbling et al. "Planning and Acting in Partially Observable Stochastic Domains". In: *Artificial Intelligence* (1998).

[9] M. L. Littman et al. "Learning policies for partially observable environments: Scaling up". In: *Machine Learning Proceedings*. 1995.

[10] O. Madani. "On the Computability of Infinite-Horizon Partially Observable Markov Decision Processes". In: *AAAI* (1999).

[11] M. Bouton et al. "Point-Based Methods for Model Checking in Partially Observable Markov Decision Processes." In: *AAAI*. 2020.

[12] M. Ahmadi et al. "Stochastic finite state control of POMDPs with LTL specifications". In: *arXiv* (2020).

[13] A. I. M. Ayala et al. "Temporal logic motion planning in unknown environments". In: *IROS*. 2013.

[14] M. Guo et al. "Revising Motion Planning Under Linear Temporal Logic Specifications in Partially Known Workspaces". In: *ICRA*. 2013.

[15] M. Guo and D. V. Dimarogonas. "Multi-agent plan reconfiguration under local LTL specifications". In: *IJRR* 34.2 (2015).

[16] M. Lahijanian et al. "Iterative Temporal Planning in Uncertain Environments With Partial Satisfaction Guarantees". In: *TRO* (2016).

[17] S. Sarid et al. "Guaranteeing High-Level Behaviors While Exploring Partially Known Maps". In: *RSS*. 2012.

[18] R. Toro Icarte et al. "Teaching Multiple Tasks to an RL Agent Using LTL". In: *AAMAS*. 2018.

[19] D. Sadigh et al. "A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications". In: *CDC*. 2014.

[20] J. Fu and U. Topcu. "Probably Approximately Correct MDP Learning and Control With Temporal Logic Constraints". In: *RSS*. 2015.

[21] M. L. Littman et al. "Environment-independent task specifications via GLTL". In: *arXiv* (2017).

[22] C. Paxton et al. "Combining neural networks and tree search for task and motion planning in challenging environments". In: *IROS*. 2017.

[23] G. J. Stein et al. "Learning over Subgoals for Efficient Navigation of Structured, Unknown Environments". In: *CoRL*. 2018.

[24] O. Kupferman and M. Y. Vardi. "Model checking of safety properties". In: *Formal Methods in System Design*. 2001.

[25] D. Silver and J. Veness. "Monte-Carlo Planning in Large POMDPs". In: *Advances in Neural Information Processing Systems*.

[26] J. Pineau and S. Thrun. *An integrated approach to hierarchy and abstraction for POMDPs*. Tech. rep. CMU, 2002.

[27] J. Li et al. "On the Relationship between LTL Normal Forms and Büchi Automata". In: *Theories of Programming and Formal Methods*. 2013.

[28] Unity Technologies. *Unity Game Engine*. `unity3d.com`. 2019.

[29] T. Yamamoto et al. "Development of HSR as the research platform of a domestic mobile manipulator". In: *ROBOMECH* (2019).

[30] *Ricoh Theta S Panoramic Camera*.

[31] C.-I. Vasile et al. "Control in Belief Space with Temporal Logic Specifications". In: *CDC*. 2016.

[32] Y. Kantaros et al. "Reactive Temporal Logic Planning for Multiple Robots in Unknown Environments". In: *ICRA*. 2020.

[33] Y. Kantaros and G. Pappas. "Optimal Temporal Logic Planning for Multi-Robot Systems in Uncertain Semantic Maps". In: *IROS*. 2019.

[34] M. Svoreňová et al. "Temporal logic motion planning using POMDPs with parity objectives: case study paper". In: *HSCC*. 2015.

[35] K. Horák and B. Bošanskỳ. "Solving Partially Observable Stochastic Games with Public Observations". In: *AAAI*. 2019.

[36] X. Li et al. "A formal methods approach to interpretable reinforcement learning for robotic planning". In: *Science Robotics* (2019).

[37] S. Carr et al. "Verifiable RNN-Based Policies for POMDPs Under Temporal Logic Constraints". In: *IJCAI*. 2020.