Temporal Logic Path Planning under Localization Uncertainty*

Amit Dhyani[1] and Indranil Saha[2]

***Abstract*— We present a method to find the optimal control strategy for a robot using prior information of localization that maximizes the probability of satisfaction of a temporal logic specification while considering the uncertainty in both motion and sensing, two major causes for localization uncertainty. The specifications are given in the probabilistic computation tree logic (PCTL) formulae over a set of propositions, which capture the presence of the robot in some key locations in the environment. A computation model that can deal with the uncertainty in both motion and sensing is the Partially Observable Markov Decision Process (POMDP), which is computationally expensive. We approximate the underlying POMDP using Augmented Markov Decision Process (AMDP) and present a control synthesis algorithm for AMDP. We carry out numerous experiments on workspaces with sizes up to 100×100 and three different PCTL specifications to evaluate the efficacy of our technique. Experimental results show that our technique for computing robot control policy using localization prior can deal with localization uncertainty effectively and scale to large environments.**

I. Introduction

Traditionally, the path planning problem for a robot deals with reaching a goal location from its initial location while avoiding obstacles, which is known as *reach-avoid specification* [1]. In the recent past, the path planning problem for a mobile robot has been considered for complex specifications, captured in some form of temporal logic. *Linear Temporal Logic* (LTL) [2] has been widely used as the specification language for complex robotic missions, and various algorithms have been proposed for solving the LTL path planning problem in various settings (e.g. [3], [4], [5], [6], [**?**], [7], [8], [9], [10], [11], [12], [13], [14]).

A major limitation of most LTL motion planning algorithms is that they consider the motion model of the robot to be perfect. However, in practice, any mobile robot involves uncertainty in its movement. To deal with this uncertainty in motion, several recent work (e.g. [15], [16], [17], [18]) have devised the probabilistic motion planning approach for a mobile robot, where the uncertain motion model of a robot due to imprecision in action executions is represented as *Markov Decision Process* (MDP) [19], and the specification of the robot is captured in *Probabilistic Computation Tree Logic* (PCTL) [20].

Apart from the uncertainty in the execution of the actions, there is another significant source of uncertainty that is ignored in most of the previous work on probabilistic motion planning for temporal logic specifications. This uncertainty is due to the incapability of state-of-the-art sensing methodologies to provide precise location information during a robot's movement. Like the motion uncertainty, error in sensing also contributes significantly to *localization uncertainty* [19]. If the inaccuracy in sensing is not taken into account during the path planning, the paths followed by the robot may be significantly inefficient.

In this paper, we present a methodology for solving the temporal logic path planning problem for a mobile robot in the presence of localization uncertainty due to the error in both action execution and sensing operation. Partially Observable Markov Decision Process (POMDP) [19] is a widely used model in robot path planning, which can deal with localization error due to both action execution and sensing operation (e.g. [21], [22]). However, solving a path planning problem on a POMDP is computationally expensive [19]. As dealing with temporal logic specifications introduces additional complexity to the path planning problem, POMDP-based algorithms to solve such path planning problems for temporal logic specifications are computationally inefficient.

To deal with the intractability of path planning using POMDP, we design a plan synthesis algorithm based on an approximate model of POMDP, which is called *Augmented MDP* (AMDP) [23], [24]. AMDP lies in-between MDP and POMDP in terms of the computational complexity of the algorithms designed for them. AMDP states are represented using lower dimension statistics. Thus, working with AMDP is computationally efficient and practical as compared to POMDP. In this paper, we show how the probabilistic model checking algorithm for MDP [16] can be extended to AMDP so that we can solve the temporal logic path planning problem for PCTL specifications in the presence of both the motion and the sensing uncertainties.

We have implemented our algorithm in a software tool that we have applied to solve the planning problem for three PCTL specifications on complex environments. Our experimental results show that the AMDP-based planning algorithm can efficiently deal with the localization uncertainty and generate the path in many situations where MDP-based path planning algorithms perform poorly. We also compare our algorithm with a recently developed point-based method for synthesizing policies for POMDPs that shows the potential to solve planning problems in the presence of both the motion and sensing uncertainties [22]. Experimental results establish that our algorithm can provide results equivalent to the point-based POMDP solver, but in significantly less computation time. Moreover, the methodology presented in [22] can deal with only LTL formulae, whereas our AMDP-based method can solve the planning problem for arbitrarily complex PCTL formulae. To the best of our knowledge, this paper, for the first time, presents an efficient algorithmic procedure to solve complex temporal logic path planning problems involving both the motion and the sensing uncertainties.

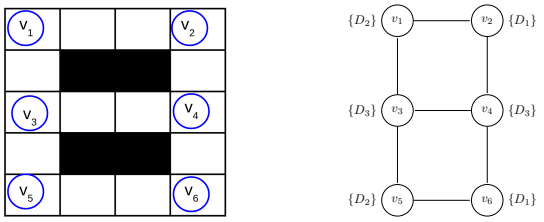
II. Preliminaries

This section provides the background concepts necessary to understand the rest of the paper.

*The authors thankfully acknowledge the Defence Research & Development Organisation (DRDO), India for funding the project through JCBCAT, Kolkata.

[1]Amit Dhyani is with the Department of Computer Science and Engineering, IIT Kanpur, India. dhyani@cse.iitk.ac.in

[2]Indranil Saha is with the Department of Computer Science and Engineering, IIT Kanpur, India isaha@cse.iitk.ac.in



(a) Grid map $\mathcal{G}$ (b) Roadmap $\mathcal{R}$

Fig. 1: (a) Grid map Environment $\mathcal{G}$, which is having six vertices. Each pathways in grid map environment $\mathcal{G}$ are bidirectional. (b) Roadmap $\mathcal{R}$ representation of Environment $\mathcal{G}$.

A. Gridmap and Roadmap

The workspace of a robot is represented as a grid-map, denoted by $\mathcal{G}$. Some of the grid locations are obstacle-free, and others are occupied by obstacles. Let $\mathcal{X}$ denote the set of obstacle-free cells in $\mathcal{G}$. Some of the cells in $\mathcal{X}$, for example, those that represent some intersections, are important for robot navigation. Depending on the specifications, we define a set of atomic propositions AP, which are `true` at some of those important cells. We create a roadmap $\mathcal{R}$ from $\mathcal{G}$ by keeping these important cells as vertices (V) and their connectivity as edges (E). We associate a roadmap with a labeling function $L : V \rightarrow 2^{AP}$ that maps each vertex in the roadmap to a subset of atomic propositions in AP. For a given grid map $\mathcal{G}$, the corresponding roadmap is represented as $\mathcal{R} = (V, E, AP, L)$.

Example 1: Consider the grid map environment $\mathcal{G}$ shown in Figure 1a, The roadmap $\mathcal{R}$ for the environment $\mathcal{G}$ is shown in Figure 1b. It consists of 6 vertices. The set of vertices is $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$. Let us assume that we are interested in three propositions D_1, D_2, D_3 which are satisfied at some important cells (vertices) of grid map $\mathcal{G}$, as shown in Figure 1b. Therefore $AP = \{D_1, D_2, D_3\}$. The mapping of the vertices to subsets of atomic propositions are as follows: $L(v_1) = \{D_2\}$, $L(v_2) = \{D_1\}$, $L(v_3) = \{D_3\}$, $L(v_4) = \{D_3\}$, $L(v_5) = \{D_2\}$, and $L(v_6) = \{D_1\}$.

B. Robot Model

1) Motion model and sensing model: We assume that the robot has discrete dynamics with `Up`, `Down`, `Left`, and `Right` motion primitives that move the robot to the four neighboring roadmap vertices. For sensing the environment, the robot uses a laser range scanner, which helps the robot localize itself with respect to its surroundings. In the real world, the actuators and sensors of a robot might not be perfect, which causes errors in the motion model and the sensing model of the robot. We assume that the noises present in the motion and sensing model are Gaussian in nature with the standard deviation of σ_m and σ_l respectively. We refer to them as *motion uncertainty* and *sensing uncertainty*, respectively.

2) Markov Localization: Localization is the problem of estimating the state of the robot in a known environment from uncertain sensor measurements. Due to motion and sensing uncertainties, the robot cannot determine its exact state. Instead, it estimates its location as the probability distribution over possible states. This probability distribution is known as robot *belief*. Let the state of the robot at time t be denoted by x_t. The belief of the robot at time t, represented by $bel(x_t)$, is computed with the help of its belief $bel(x_{t-1})$ at time $t-1$, robot control action u_t at time t, and sensor data z_t at time t [19]. Prediction and measurement update are two essential steps in Markov Localization. Markov Localization computes belief $\overline{bel(x_t)}$ known as predicted belief during the prediction step using $bel(x_{t-1})$ and u_t. We cannot directly use this belief because the error in this belief increases over time, and after some iterations, this belief becomes too uncertain. Thus, we have to improve the robot's belief with sensor data. This step is known as *measurement update*. In this step, $bel(x_t)$ is computed with the help of predicted belief $\overline{bel(x_t)}$ and sensor data z_t at time t.

3) Localization Prior: Localization prior information indicates how well a robot can localize itself at a particular grid cell. We compute this prior information using the method described in [25]. Localization prior information can be calculated using the grid cell information about the obstacles and the sensor model of the laser range finder. It computes the covariance matrix for each grid location $x \in \mathcal{X}$, higher value of covariance at a particular grid cell indicates that the robot is less likely to localize itself at that grid cell and vice-versa. Since the environment is static, we compute the localization prior once for each grid location.

4) Control Policy: When a robot arrives at a (concrete or belief) state, it has to decide which action to choose. The policy is the choice to pick a particular action at a given state of the robot. The control policy maps the state space (S) of the robot to the set of available actions or action space (A). Therefore, given the state of the robot $s \in S$, the control policy $\mu(a|s)$, $a \in Act$, is a probability distribution over a set of actions.

C. MDP, POMDP, and AMDP

1) Markov Decision Process (MDP): MDP allows for optimal decision-making in a fully observable environment under only motion uncertainty. In MDP, the robot can compute its location precisely at any time.

Definition 2.1 (MDP): An MDP is defined as a tuple, $M = (S, s_0, Act, P, AP, L)$, where S is a finite set of states, $s_0 \in S$ is the initial state, *Act* is the set of possible actions, P: $S \times Act \times S \rightarrow$ [0, 1] is the probability transition function such that for all states $s \in S$ and action $a \in Act$, $\sum_{s' \in S} P(s, a, s') = 1$, AP is the set of atomic propositions, and L is the labeling function: $S \rightarrow 2^{AP}$.
We represent the cardinality of S and *Act* in an MDP by n and m respectively. The Probability Transition function P is represented using a matrix of dimension $n \times m \times n$, where $P(:, a, :)$ represents all the transitions from state $s_i \in S$ to state $s_j \in S$ on an action $a \in Act$.

2) Partially Observable Markov Decision Process (POMDP): POMDP allows for optimal decision-making under both motion uncertainty and sensor uncertainty [26], [27]. In POMDP, the robot cannot predict its exact location. Instead, it computes the probability distribution over all possible states, also known as belief.

Definition 2.2 (POMDP): A POMDP is defined by a tuple (S', Act, P', R, O, Z), where S' is set of belief states, Act is the set of actions. For $s, s' \in S'$ and $a \in Act$, P' is the transition probability $P'(s, a, s')$, $R(s, a)$ is the reward function. O is set of observations. For $o \in O$, $Z(o, s', a)$ defines the observation probability $Pr(o \mid s', a)$.

3) Augmented Markov Decision Process (AMDP): POMDP considers every possible belief, but many of these beliefs are unlikely in practice. This increases the complexity of the underlying algorithm [19]. AMDP considers both types of uncertainty while maintaining the degree of tractability [24]. AMDP uses augmented state representation to approximate a POMDP as a variant of MDP. The AMDP

states discretize the set of belief states in the POMDP, and explicitly store the uncertainties present in the environment. The states in the augmented MDP is represented as a tuple of the vertex $v \in V$ and variance $\sigma^2 \in W$ [28]. Let Q be the set of augmented states. Formally, we can define Q as:

$$Q = \{(v_i, \sigma_j^2) \mid v_i \in V, \sigma_j^2 \in W\}, \tag{1}$$

where V is the set of vertices in the roadmap $\mathcal{R}$ and W is the set of possible variances that discretizes the possible range of uncertainty present in the environment. Thus, the total number of states in the Augmented MDP is $|V| \times |W|$, where $|V|$ and $|W|$ are the cardinality of the set V and W respectively. Each augmented state $q = (v, \sigma^2)$ represents the Gaussian probability distribution over the cells of grid map $\mathcal{G}$, whose mean is v, and isotropic covariance $\Sigma = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}$. Therefore, by representing the robot's belief using the tuple (v, σ^2), which is independent of the number of grid map cells, the complexity of the underlying algorithm decreases significantly. Each vertex $v \in V$ satisfies zero or some propositions (See example 1). Thus, an augmented state satisfies each proposition in AP with zero or some probability (see Example 2).

Transition function $P(q'|q, a)$ defines the probability to reach augmented state q' given that action a is executed at augmented state q. We can compute the transition function $P(q'|q, a)$ between the augmented states q ad q' in three steps using the method proposed in [28]. Firstly, using the localization prior information (See Section II-B.3), we calculate the probability distribution about the position of the robot given that the robot executes an action a from a vertex v, represented by $p(x|v, a)$ ($x \in \mathcal{X}$), and refer it as *posterior from vertex*. In the second step, using *posterior from a vertex*, we compute $p(x|q, a)$ termed as *posterior from a state*, which is defined as the probability distribution of the robot (or belief of the robot) about its position given that robot executes an action a from an augmented state q. Finally, we compute the transition function $P(q'|q, a)$ by mapping the *posterior from a state* $p(x|q, a)$ to one of the AMDP states.

We now present the formal definition of AMDP.

Definition 2.3 (AMDP): An AMDP is defined as a tuple, $\mathcal{A} = (Q, q_0, Act, P, AP, L')$, where Q is a finite set of belief states, $q_0 \in Q$ is the initial belief state, *Act* is the set of possible actions, P: $Q \times Act \times Q \rightarrow$ [0, 1] is the probability transition function such that for any belief state $q \in Q$ and action $a \in Act$, $\sum_{q' \in Q} P(q, a, q') = 1$, AP is the set of atomic propositions, and L' is the probability labeling function: $Q \times AP \rightarrow$ [0, 1].

Example 2: We construct the augmented MDP $\mathcal{A}$ (shown in Figure 2) of the roadmap $\mathcal{R}$ (shown in Figure 1b) using the method described in [28]. Consider the roadmap environment $\mathcal{R}$, which contains six vertices ($|V| = 6$). Assume that $|W| = 2$, i.e., $W = \{\sigma_1^2, \sigma_2^2\}$, where $\sigma_1^2 = 0.2\text{m}^2$ and $\sigma_2^2 = 1.0\text{m}^2$. Then the number of augmented states $|Q|$ in the augmented MDP $\mathcal{A}$ is 12. Each state in Augmented MDP is represented by (v_i, σ_j^2), where $v_i \in V$ and $\sigma_j^2 \in W$. The transition function between the augmented states is computed using the method proposed in [28] (implementation details discussed in [28]). Assume that the probability distribution represented by an augmented state q over the vertices of roadmap $\mathcal{R}$ contains $p_1, p_2, p_3, p_4, p_5, p_6$, which represent the probability of the robot to be at vertex v_1, v_2, v_3, v_4, v_5, and v_6, respectively.

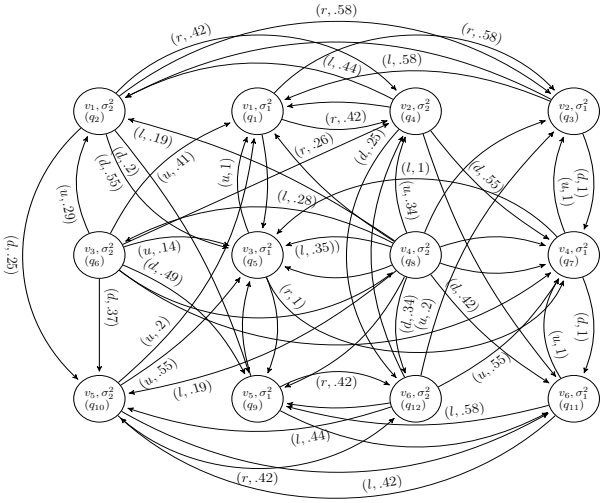


Fig. 2: Augmented MDP '$\mathcal{A}$' of Environment $\mathcal{G}$ with 2 cardinality of W, i.e., $W = \{\sigma_1^2, \sigma_2^2\}$, where $\sigma_1^2 = 0.2\text{m}^2$ and $\sigma_2^2 = 1.0\text{m}^2$. $Act = \{u, d, l, r\}$, where u, d, l, r represent action `Up`, `Down`, `Left`, and `Right` respectively.

Since proposition D_1 is true at vertex v_2 and v_6, the augmented state q satisfies proposition D_1 with probability $p_2 + p_6$. Similarly, q satisfies proposition D_2 and D_3 with probability $p_1 + p_5$ and $p_3 + p_4$, respectively. Therefore, from the calculated belief represented by each augmented state, we obtain that proposition D_1 is satisfied by the augmented states $[q_2, q_3, q_4, q_6, q_8, q_{10}, q_{11}, q_{12}]$ with probability $[0.0097, 1, 0.87, 0.0023, 0.21, 0.0097, 1, 0.87]$. Similarly, proposition D_2 and D_3 are satisfied by the augmented state $[q_1, q_2, q_4, q_6, q_8, q_9, q_{10}, q_{12}]$, and $[q_2, q_4, q_5, q_6, q_7, q_8, q_{10}, q_{12}]$ respectively with probability, $[1, 0.87, 0.0097, 0.21, 0.0023, 1, 0.87, 0.0097]$, and $[0.12, 0.12, 1, 0.79, 1, 0.79, 0.12, 0.12]$.

D. Probabilistic Computation Tree Logic (PCTL)

PCTL is a probabilistic extension of Computation Tree Logic (CTL) [2]. A PCTL state formula over a given set AP of atomic propositions is recursively defined as :

$$\phi = \mathsf{true} \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid \mathcal{P}_J(\psi),$$

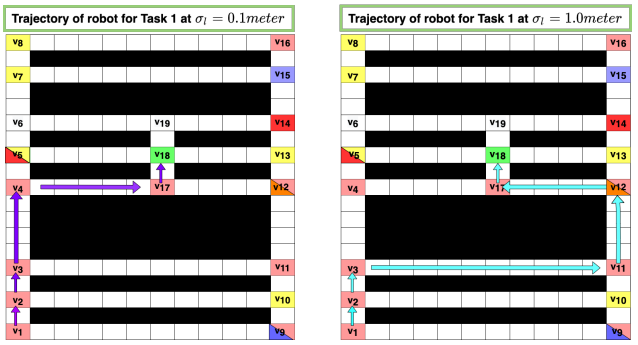
where $a \in AP$, ψ is a path formula, $\wedge$ is the Boolean *conjunction* operator, $\neg$ is Boolean *negation* operator, and J is an interval between [0,1]. A PCTL path formulae is defined as:

$$\psi = \mathsf{X}\phi \mid \phi_1 \mathsf{U} \phi_2 \mid \phi_1 \mathsf{U}^{\leq k} \phi_2,$$

where ϕ_1, ϕ_2 and ϕ are state formulae, X is Temporal Logic *Next* operator, U is *Unbounded Until* operator, $\mathsf{U}^{\leq k}$ is *Bounded Until* operator and $k \in \mathbb{N}$.

III. Problem

Let us consider a gridmap $\mathcal{G}$ with the corresponding roadmap $\mathcal{R} = (V, E, AP, L)$. During navigation on the gridmap, when the robot executes an action, the robot's Markov localization system generates a belief $bel(x)$ about its position over the cells of the gridmap. When the maximum probability in the belief corresponds to a vertex in the roadmap, the robot considers itself to be present at that vertex. The robot keeps on executing the same action from a roadmap vertex till it reaches a new vertex. When the



(a) at $\sigma_l = 0.1\text{m}$ (b) at $\sigma_l = 1.0\text{m}$

Fig. 3: Environment $\mathcal{G}_1$ having grid size 19×12, and size of each grid cell is $1\text{m} \times 1\text{m}$. $\mathcal{G}_1$ has 19 vertex. We are interested in 6 atomic propositions, which are satisfied at the vertices of the environment. Different atomic propositions are marked with different color: Service Station 1 (S_1, Red) $\{v_5, v_{14}\}$, Service station 2 (S_2, Purple) $\{v_9, v_{15}\}$, Department Store (D, Green) $\{v_{18}\}$, Warehouse (W_h, Orange) $\{v_{12}\}$, Super Market (M, Yellow) $\{v_5, v_7, v_8, v_{10}, v_{13}, v_{18}\}$ and Checkpoint (C, Pink) $\{v_1, v_2, v_3, v_4, v_9, v_{11}, v_{12}, v_{16}, v_{17}\}$. Violet and Cyan arrow are showing the trajectory of robot for *Task* 1 at sensor noise having standard deviation of $\sigma_l = 0.1\text{m}$ and $\sigma_l = 1.0\text{m}$ respectively.

robot detects itself to be present at some vertex, it maps the belief $bel(x)$ to one of the augmented state $q_{min} \in Q$, which has the minimum Bhattacharyya distance (D_b) [29] with the belief $bel(x)$, i.e.,

$$q_{min} = \arg\min_{q_i \in Q} \left(D_b(bel(x), q_i) \right), \tag{2}$$

and then executes the action corresponding to q_{min}.

Thus, the execution of a control policy leads to a sequence of roadmap vertices $\xi = v_0\, v_1 \ldots$ through which the robot moves. A trace corresponding to a trajectory $\xi = v_0\, v_1 \ldots$, $v_i \in V$, is the sequence $\pi = L(v_0)\, L(v_1) \ldots$ of the subset of atomic propositions that are `true` at the corresponding vertices in the trajectory. Our goal in this paper is to synthesize a control policy $\mu : Q \rightarrow Act$ for a robot that leads to a trace π maximizing the probability of satisfaction of a PCTL specification.

For a PCTL path formula ψ, we write $\mathcal{P}_{max}(\psi)$ to denote the maximum probability of reaching the states that satisfy the path formula ψ. For synthesis, it is meaningful to define a PCTL formula as $\mathcal{P}_{max}(\psi)$ instead of $\mathcal{P}_J(\psi)$.

Now, we define our problem formally as follows:

Problem 3.1: Given a roadmap $\mathcal{R}$ corresponding to an environment $\mathcal{G}$ and a task in PCTL formula $\phi = \mathcal{P}_{max}(\psi)$ that uses the atomic propositions AP defined on $\mathcal{R}$, find the control policy for the robot that leads to a trace π having the maximum probability of satisfying the PCTL path formula ψ under both motion and sensing uncertainties.

Let us illustrate the problem with an example.

Example 3: Consider the gridmap $\mathcal{G}_1$ shown in Figure 3 (ignore the arrows at this moment). The dimension of gridmap $\mathcal{G}_1$ is 19×12, where size of each grid cell is $1\text{m} \times 1\text{m}$. The roadmap $\mathcal{R}_1$ corresponding to $\mathcal{G}_1$ has 19 vertices which have been marked in the figure. The propositions `true` at different vertices have been provided in the caption of the figure. Now let us consider the following PCTL task on roadmap $\mathcal{R}_1$:

Task 1: The robot should visit the Department Store within 8 steps, and before visiting the department store, it should avoid visiting Service Station 1 and Service Station 2. Formally, $\phi_1 = \mathcal{P}_{max}(\neg S_1 \wedge \neg S_2)\ \mathsf{U}^{\leq 8}\ D$.

Assume that the initial position of the robot is v_1. Let us compare the trajectories of the robot to satisfy *Task* 1 at low sensor noise (at $\sigma_l = 0.1\text{m}$) and high sensor noise (at $\sigma_l = 1.0\text{m}$). The results are shown in Figure 3.

When the sensor noise is low (at $\sigma_l = 0.1\text{m}$), the robot's trajectory for *Task* 1 starting at vertex v_1 is as follows (shown in Figure 3a with violet color). The robot reaches vertex v_4 via vertex v_2, v_3, and then it reaches v_{18} via vertex v_{17} by taking a right turn at vertex v_4. However, at a high sensor noise (when $\sigma_l = 1.0\text{m}$), the Markov localization system computes more corrupted beliefs about the robot's position. In this case, the chances of reaching vertex v_5 by missing vertex v_4 increases, and from vertex v_5, *Task* 1 cannot be satisfied as the robot has already satisfied proposition S_1. In this situation, the robot should follow another trajectory to maximize its chance of satisfying the given specification. Such a path is shown in Figure 3b with Cyan color. The robot reaches vertex v_3 via vertex v_2 and then takes a right turn at vertex v_3 to reach vertex v_{11}. Then it reaches vertex v_{18} via vertices v_{12}, v_{17}, which is shown in Figure 3(b).

In the next section, we will describe how we can generate such trajectories algorithmically using Augmented MDP.

IV. AMDP-Based Algorithm

This section presents our approach for solving the path planning problem for a PCTL formula ϕ in the presence of both motion uncertainty and sensor uncertainty.

A. Control Synthesis for PCTL formula on Augmented MDP

In this subsection, given an AMDP $\mathcal{A}$ and a PCTL formula $\phi = \mathcal{P}_{max}(\psi)$, we will discuss how to find the robot control policy corresponds to the maximum probability of satisfying the path formula ψ. Our algorithm for finding the control policy is motivated by [16]. However, we cannot directly apply the equations described in [16] because, in [16], the goal is to find the control policy for an MDP, where the robot knows its position accurately at any point in time. Consequently, by having accurate information about its position, the robot deterministically knows which propositions are true or false at any time. On the other hand, the robot does not know its position accurately in our current setting. Therefore, at any point, the robot cannot accurately decide which propositions are true or false. It can only compute the satisfaction of a proposition with some probability. So, we need to modify the equations described in the [16]. In the next subsections, we will discuss the modified equations for the next operator, the bounded until operator, and the unbounded until operator for the PCTL formula ϕ.

We categorize the PCTL formula into two types: simple PCTL formula and complex PCTL formula. Simple PCTL formulae are those formulas that contain at most one $\mathcal{P}$-operator, which is discussed in section IV-A.1. On the other hand, complex PCTL formulae are those which contain more than one path formulae. Section IV-A.2 discusses the control synthesis for the complex PCTL formula.

1) Control Synthesis for simple PCTL formula:

a) Next Operator: This algorithm is used to find the control policy for the PCTL formula: $\phi = \mathcal{P}_{max}(\mathsf{X}\phi_1)$, where $\mathsf{X}\phi_1$ is a PCTL path formula. Consider the Augmented MDP $\mathcal{A}$ shown in Figure 2 and PCTL state formula $\phi = \mathcal{P}_{max}(\mathsf{X}\phi_1)$. We have to find the action at each augmented state such that the probability of satisfying ϕ_1 at the next state is maximum. Thus, we need to consider only the immediate action at each state. We can find $Pr_{q_i}(\phi)$, the maximum

probability of satisfaction ϕ for state q_i, and $\mu^*(q_i)$, the corresponding control, using the below equations.

$$Pr_{q_i}(\phi) = \max_{a \in Act(q_i)} \left(\sum_{q_j \in Sat(\phi_1)} P(q_i, a, q_j) \cdot (\overline{\phi_1})_{q_j} \right) \quad (3)$$

$$\mu^*(q_i) = \arg \max_{a \in Act(q_i)} \left(\sum_{q_j \in Sat(\phi_1)} P(q_i, a, q_j) \cdot (\overline{\phi_1})_{q_j} \right) \quad (4)$$

where Sat(ϕ_1) denotes the set of states which satisfy the PCTL formula ϕ_1 with probability greater than zero, P is the probability transition function for the AMDP, and $(\overline{\phi_1})_{q_j}$ is the probability to satisfy ϕ_1 at state q_j. In the matrix-vector notation, the above probability and optimal policy can be computed with constant matrix-vector multiplication and one maximization operation. Let $\overline{\phi_1}$ be the state indexed vector, whose dimension is $n \times 1$, where n denotes the number of augmented states in Augmented MDP. For each action $a \in A$, we multiply the probability transition function represented as matrix $P(:, a, :)$ with the state vector $\overline{\phi_1}$. As a result, we get m vectors, where m is the number of actions possible in the AMDP.

Example 4: Consider the Augmented MDP given in Figure 2 and the PCTL formula $\phi = \mathcal{P}_{max}(\mathsf{X} D_3)$. For action `Up` represented as 'u', Equation 3 can be written in matrix-vector form as follow:

$$Pr_u(\phi) = P(:, up, :) \times \overline{\phi_1} \quad (5)$$

By putting the values, we obtain vector $Pr_u(\phi)$ as:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.41 & 0.29 & 0 & 0 & 0.14 & 0.16 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.12 & 0.098 & 0.31 & 0.23 & 0 & 0 & 0.12 & 0.12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.2 & 0.25 & 0 & 0 & 0.55 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.2 & 0.25 & 0 & 0 & 0.55 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0.12 \\ 0 \\ 0.12 \\ 1 \\ 0.79 \\ 1 \\ 0.79 \\ 0 \\ 0.12 \\ 0 \\ 0.12 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.3 \\ 0 \\ 0.25 \\ 1 \\ 0.58 \\ 1 \\ 0.58 \end{pmatrix}$$

Similarly, we can find resultant vectors for action `Down`, `Left`, and `Right` represented by $Pr_d(\phi)$, $Pr_l(\phi)$ and $Pr_r(\phi)$ respectively.

$Pr_d(\phi) = [1 \quad 0.58 \quad 1 \quad 0.58 \quad 0 \quad 0.3 \quad 0 \quad 0.25 \quad 0 \quad 0 \quad 0 \quad 0]^\intercal$,
$Pr_l(\phi) = [0 \quad 0 \quad 0.05 \quad 0.31 \quad 0 \quad 0 \quad 1 \quad 0.61 \quad 0 \quad 0 \quad 0.05 \quad 0.31]^\intercal$,
$Pr_r(\phi) = [0.05 \quad 0.31 \quad 0 \quad 0 \quad 1 \quad 0.61 \quad 0 \quad 0 \quad 0.05 \quad 0.31 \quad 0 \quad 0]^\intercal$.

At the end, we have to perform the maximization operation on each state. For example, for state q_4: $Pr_{q_4}(\phi) = \max(0, 0.58, 0.31, 0)$, which is equals to 0.58 and $\mu^*(q_4)$={`d`}. Similarly, for other states, we can find the optimal probability and its corresponding action. It should be noted that if the probability is zero for some state, then there is no way the robot can satisfy the task from that state.

b) Bounded Until Operator: Now we will discuss how to deal with the formula of the form $\phi = \mathcal{P}_{max}(\phi_1 U^{\leq k} \phi_2)$, where ϕ_1 and ϕ_2 contain only Boolean operators or propositions. Assume that $Pr^k_{q_i}(\phi)$ represent probability to satisfy ϕ in k steps at augmented state q_i. The overall probability of satisfaction of ϕ can be computed recursively using the following equations :

$$Pr^k_{q_i}(\phi) = \max_{a \in Act(q_i)} \left((\overline{\phi_2})_{q_i} + (1 - \overline{\phi_2})_{q_i} (\overline{\phi_1})_{q_i} x^k_{(q_i,a)} \right) \quad (6)$$

$$\mu^*_k = \arg \max_{a \in Act(q_i)} \left((\overline{\phi_2})_{q_i} + (1 - \overline{\phi_2})_{q_i} (\overline{\phi_1})_{q_i} x^k_{(q_i,a)} \right) \quad (7)$$

where we can compute $x^k_{(q_i,a)}$ using

$$x^k_{(q_i,a)} = \textstyle\sum_{q_j \in Q} P(q_i, a, q_j) Pr^{k-1}_{q_j}(\phi) \quad (8)$$

Here, $(\overline{\phi_1})_{q_i}$ represents the probability of satisfaction of ϕ_1 at augmented state q_i, $(\overline{\phi_2})_{q_i}$ is probability to satisfy ϕ_2 at augmented state q_i, P is probability transition function.

Example 5: To demonstrate this method, let us consider the Augmented MDP given in Figure 2 and PCTL formula $\phi = \mathcal{P}_{max}(D_3 \ \mathsf{U}^{\leq 2} \ D_1)$. Now, we have to recursively apply Equation (6) for each possible action. For action `up`, Equation (8) can be written in matrix-vector form as follow:

$$x^k_{up} = P(:, up, :) \times Pr^{k-1}(\phi) \quad (9)$$

Assume that $diag(v)$ represents a square diagonal matrix with the elements of vector v on the main diagonal. For action `up` and $k = 1$, Equation (6) can be written in matrix-vector form as follow:

$$Pr^1_u(\phi) = \overline{D_1} + diag(1 - \overline{D_1}) \times diag(\overline{D_3}) \times x^1_{up} \quad (10)$$

By putting the values of $\overline{D_1}$, $\overline{D_3}$ and x^1_{up} in Equation (10), we obtain:

$Pr^1_u(\phi) = [0 \ 0.0097 \ 1 \ 0.87 \ 0 \ 0.0048 \ 1 \ 0.55 \ 0 \ 0.01 \ 1 \ 0.88]^\intercal$

Similarly, we can compute $Pr^1_d(\phi)$, $Pr^1_l(\phi)$, $Pr^1_r(\phi)$ as follows:

$Pr^1_d(\phi) = [0 \ 0.01 \ 1 \ 0.88 \ 0 \ 0.0048 \ 1 \ 0.55 \ 0 \ 0.0097 \ 1 \ 0.87]^\intercal$,

$Pr^1_l(\phi) = [0 \ 0.0097 \ 1 \ 0.87 \ 0 \ 0.0023 \ 0 \ 0.21 \ 0 \ 0.0097 \ 1 \ 0.87]^\intercal$,

$Pr^1_r(\phi) = [0 \ 0.09 \ 1 \ 0.87 \ 0 \ 0.3 \ 0 \ 0.21 \ 0 \ 0.09 \ 1 \ 0.87]^\intercal$.

In the end, $Pr^1(\phi)$ can be computed using one maximization operation at each state. For example, for state q_6, $Pr^1_{q_6}(\phi) = \max(0.0048, 0.0048, 0.0023, 0.3)$, which is equals to 0.3 and its corresponding control policy at first time step $\mu^1(q_6) =$ {`r`}. Similarly, we can find the optimal probability and its corresponding control policy for other states. In the next iteration, when $k = 2$, we can compute $Pr^2(\phi)$, using the above equations. For example, for state q_5, $Pr^2_{q_5}(\phi) = 1.0$ and its corresponding control policy at second time step $\mu^2(q_5) =$ {`r`}.

c) Unbounded Until Operator: Now we compute the control policy of satisfaction for the PCTL formula, which is of the form $\phi = \mathcal{P}_{max}(\phi_1 \ \mathsf{U} \ \phi_2)$, where ϕ_1 and ϕ_2 contain only Boolean operators or propositions. The until operator is the same as U^k as $k \to \infty$. We use the algorithm of bounded until operator discussed in Section IV-A.1.b. The iteration terminates when the solutions converge sufficiently.

Example 6: To demonstrate the above method, consider the PCTL formula $\phi = \mathcal{P}_{max}(D_3 \ \mathsf{U} \ D_1)$, we obtain the optimal probability $Pr(\phi)$=[0 0.12 1 0.89 1 0.68 1 0.66 0 0.12 1 0.89 0]$^\intercal$, and the corresponding policy at states : $\mu^2(q_5) = \mu^2(q_6) =$ {`r`}, $\mu^2(q_7) = \mu^2(q_8) =$ {`u`}.

2) Control synthesis for complex PCTL formula: Complex PCTL formulae are the ones that contain more than one path formulae. We can construct a complex PCTL formula by nesting the probabilistic operator, which specifies more complex tasks. We have followed the same approach for the complex PCTL formula described in [16]. The only difference is that we use the equations described in section IV-A.1 instead of the equations mentioned in [16]. In case when the outermost temporal operator is bounded until or unbounded until, the complex nested PCTL formula can be written as:

$$\phi_1 = \mathcal{P}_{max}(\phi_L \ \mathsf{U} \ \phi_R) \quad (11)$$

$$\phi_2 = \mathcal{P}_{max}(\phi_L \ \mathsf{U}^{\leq k} \ \phi_R) \quad (12)$$

where ϕ_L and ϕ_R are PCTL formulas, and at least one of them contains the $\mathcal{P}$-operator. We first compute the set of states Q_{ϕ_R} satisfying the PCTL formula ϕ_R and its corresponding control policy μ_{ϕ_R} using the algorithm described in section IV-A.1. Similarly, we compute the control policy μ_{ϕ_L} for the PCTL formula ϕ_L. In order to satisfy the PCTL formula ϕ_1 and ϕ_2, the robot must reach a state in Q_{ϕ_R} only through the states present in the Q_{ϕ_L}. Therefore, we construct another AMDP $\mathcal{A}' \subset \mathcal{A}$ by eliminating all the actions from $\mathcal{A}$ which are not present in μ_{ϕ_L}. If, in this process, any AMDP states contain no outgoing transitions, a self-loop is inserted in that AMDP state to avoid having any blocking AMDP state. Finally, we find the control policy μ_ϕ for the outermost temporal operator on the modified Augmented MDP $\mathcal{A}'$ using the algorithm described in section IV-A.1.

When the outermost operator is Next operator, the complex nested PCTL formula can be written as:

$$\phi_3 = \mathcal{P}_{max}(\mathsf{X}\ \phi_R) \tag{13}$$

where ϕ_R must contain a probabilistic operator. In this case also, we apply the same procedure as described above. We first compute the set of states Q_{ϕ_R} and its corresponding control policy μ_{ϕ_R}, then find the control policy μ_ϕ for the outermost Next operator discussed in section IV-A.1.a. Therefore, the overall control policy μ^* is as follows: Apply the control policy μ_ϕ until a robot reaches a state in Q_{ϕ_R}, then apply control policy μ_{ϕ_R}.

V. Evaluation

This section describes our experience in applying the algorithm described in the previous section to solve various path planning problems under motion and sensor uncertainties.

A. Experimental Setup

We have implemented our path planning algorithm in C++. The implementation is available in the following repository: `https://github.com/iitkcpslab/RTPlan`. The results shown in this section are obtained on a system with a 3.40 GHz octa-core processor with 32 GB RAM. Most of the experiments presented in this section are carried out in the grid map environment $\mathcal{G}_1$ shown in Figure 3.

1) PCTL Tasks: Along with Task 1 introduced in Example 3, we consider the following two PCTL tasks for our experiments. These properties are also defined for the grid map environment $\mathcal{G}_1$ shown in Figure 3.

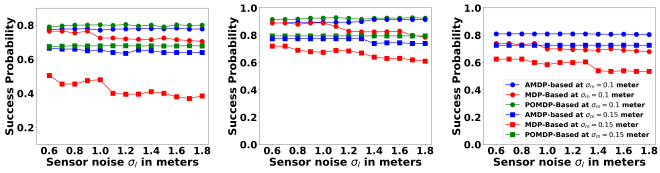
Task 2: The robot will eventually visit the Department store starting from vertex v_8 without going through any checkpoints. Formally, $\phi_2 = \mathcal{P}_{max}(\neg C\ \mathsf{U}\ D)$.

Task 3: The robot first reaches the Department store only through the Super Market and Checkpoint while avoiding Service Station 1. After reaching the Department store, the robot reaches the Warehouse with a probability greater than 0.1 while avoiding Service Station 1 and the Super Market. $\phi_3 = \mathcal{P}_{max}((\neg S_1 \wedge (M \vee C))\mathsf{U}(D \wedge P_{>0.1}((\neg S_1 \wedge \neg M)\mathsf{U}\ W_h)))$.

2) Success probability: For a given temporal logic specification ϕ, we define *success probability* as the simulation probability for the robot to satisfy ϕ. Formally, we define *success probability* as follow:

$$success\_probability\ (\phi) = \frac{H}{N}, \tag{14}$$

where H is the number of simulations when the robot completes the task ϕ successfully, and N is the total number of simulations.



(a) Task 1 (b) Task 2 (c) Task 3

Fig. 4: Success probability of AMDP ($|W| = 5$), MDP and POMDP-based approach at motion noise having standard deviation $\sigma_m = 0.1$m and $\sigma_m = 0.15$m, with varying sensor noise

B. Baseline for Comparison

We compare the simulation result of our AMDP-based approach with both the MDP and POMDP-based approaches.

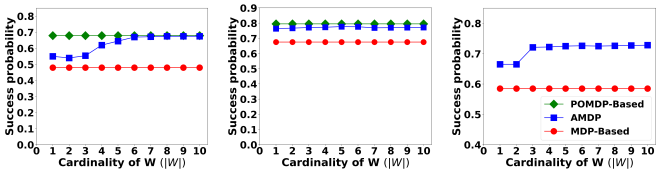
To solve Problem 3.1 using the MDP-based approach, we construct the MDP in which states are vertices V of the environment $\mathcal{G}_1$, and the transitions between the vertices of the environment are defined by the transition function. Since MDP assumes that the robot's position is known exactly and ignores the uncertainty in the belief of the robot, the generated policy may lead to a wrong path when there is significant uncertainty in the robot's belief.

To solve Problem 3.1 using POMDP, we first construct the POMDP by following the procedure in [22]. We have created a partially observable grid world similar to $\mathcal{G}_1$. The robot can use actions `up`, `down`, `left`, `right` to move on desired cell. We intentionally added the noise in the movement of the robot to make the motion model of the robot noisy. The robot uses a noisy laser range scanner to localize itself in the grid world. Then, we use SARSOP [30] as a POMDP solver. The precision of the SARSOP solver is set to 1×10^{-2}. Since [22] can deal with only LTL path formulae, we cannot solve the plan synthesis problem for the specification ϕ_3 as it contains the inner Probability term.

C. Results

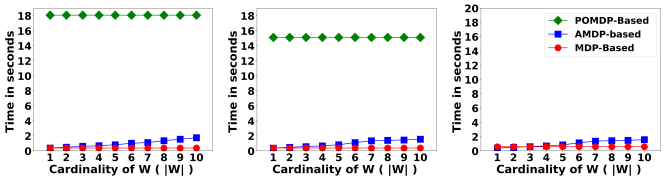
We now present our experimental results in detail.

1) Effect of sensor noise: This experiment compares the AMDP-based approach's performance with that of the MDP and the POMDP-based approaches for different sensor noises having a standard deviation ranging from 0.6m to 1.8m. We first construct our AMDP for Environment $\mathcal{G}_1$ (shown in Figure 3) using the set $W = \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and run our algorithm 1000 times for each task and each sensor noise to find the success probability. The results of this experiment are shown in Figure 4. When the sensor noise is low, the MDP-based approach's performance is high, but when sensor noise increases, the MDP-based approach's performance decreases rapidly. The reason behind this behavior is as follows. When the sensor noise increases, the Markov localization system generates a corrupted belief in the robot. Thus, in the case of the MDP-based approach, when the robot reaches some vertex with an enormous corrupted belief, the chances of misclassifying the robot's position increase. Due to this, the probability of completing the task decreases. However, in the AMDP-based approach, we approximate the robot's belief with the augmented states. We explicitly store the uncertainty (variance) in the set W. Therefore, the chance of misclassification of robot position decreases, and, consequently, the AMDP-based approach performs much better than the MDP-based approach at high sensor noise. This experiment also shows that the performance of the AMDP-based approach is



(a) Task 1 (b) Task 2 (c) Task 3

Fig. 5: Success probability of AMDP, MDP and POMDP-based approaches with respect to cardinality of W at motion noise having standard deviation $\sigma_m = 0.15\text{m}$ and sensor noise with standard deviation $\sigma_l = 1.0\text{m}$



(a) Task 1 (b) Task 2 (c) Task 3

Fig. 6: Computation time for *Task* 1, 2, 3 for POMDP, AMDP and MDP at motion noise having standard deviation $\sigma_m = 0.15\text{m}$ and sensor noise having standard deviation $\sigma_l = 1.0\text{m}$.

almost the same as that of the POMDP-based approach at different sensor and motion noise.

2) Effect of cardinality of W: We conduct experiments to measure the effect of the cardinality of W on the performance of the Augmented-MDP approach. The results are shown in Figure 5. In this experiment, we add motion noise having standard deviation $\sigma_m = 0.15\text{m}$ and sensor noise in the robot's sensors with standard deviation $\sigma_l = 1.0\text{m}$. We start with two variances in W ($|W| = 2$), such that $W = \{0.1, 0.2\}$, then we gradually add more variances into the set W (e.g., when $|W| = 4$, the set $W = \{0.1, 0.2, 0.3, 0.4\}$) to increase the cardinality of W. We perform 1000 runs for each task for each cardinality of W. The experimental results show that the success probability of the AMDP-based approach increases with an increase in the cardinality of W. The reason for this behavior is as follows: Due to high sensor noise, the Markov localization system computes the corrupted belief of the robot, so when the cardinality of W is low, then the set of variances in W cannot approximate the corrupted robot's belief correctly, which results in misclassification of the robot's position. However, when the cardinality of W increases, we add more variances in W, which approximates the robot's belief with a more appropriate augmented state. As expected, MDP based approach shows the poorest performance, and POMDP based approach shows the best performance in terms of the success probability. However, by increasing the cardinality of $|W|$ in AMDP, it is possible to achieve the performance of POMDP.

3) Computation time: We also compare the computation time of AMDP with the computation time of MDP and POMDP at different cardinalities of set W. The *computation time* is the sum of time taken to construct the AMDP/MDP/POMDP and the time taken to generate the policy. The results are shown in Figure 6 and Table I. Figure 6 shows that the computation time of AMDP increases as the cardinality of W increases because the computation time depends on the number of states in the AMDP. The number of states in AMDP depends on the number of vertices in the roadmap and the cardinality of set W. When the number of states in the augmented MDP increases, the total cost of matrix-matrix and matrix-vector multiplication increases. Consequently, the computation time increases. We can also conclude from Figure 6 that the computation time for the AMDP is much lower than POMDP based approach and slightly more than MDP based approach. Therefore, our approach provides similar results as the POMDP-based approach in a reasonable time.

4) Scalability: We have created a 50×50 grid map (shown in Figure 7) and a 100×100 grid map (similar to the 50×50 one), having 49 and 144 vertices in the corresponding road maps respectively. In this experiment, we measure the success probability and the computation time for these grid

TABLE I: Time taken by AMDP, POMDP and MDP for different task

AMDP/MDP/POMDP	Construction time (in s)	Policy generation time (in s)		
		Task 1	Task 2	Task 3
AMDP ($\|W\| = 2$)	0.382	0.072	0.082	0.105
AMDP ($\|W\| = 4$)	0.45	0.168	0.176	0.225
AMDP ($\|W\| = 6$)	0.772	0.252	0.243	0.35
AMDP ($\|W\| = 8$)	0.852	0.325	0.32	0.463
AMDP ($\|W\| = 10$)	1.12	0.398	0.42	0.529
POMDP	8.25	10.28	11.56	–
MDP	0.35	0.092	0.101	0.24

map environments at different cardinalities of set W ($|W| = 1, 4, 8$). Then we compare the results with the POMDP approach. The computation time and success probability for both the environment are shown in Figure 8. From Figure 8, it is clearly seen that the success probability using the AMDP-based approach is similar to that of the POMDP-based approach. However, it is evident that the computation time for the POMDP is significantly more as compared to the AMDP based approach. From the experiments, we can conclude that, unlike POMDP based technique, our technique can solve the temporal logic motion planning problem considering both motion and sensor uncertainty scalably without compromising on the performance.

5) ROS+Gazebo simulation: In this experiment, we perform simulations with Turtlebot3 [31] on ROS+Gazebo [32], [33]. The video of the simulation has been submitted as a supplementary material and is also available at `https://youtu.be/7WBqyoo4V5M`. Turtlebot3 is equipped with a laser range scanner mounted at the top. We added sensor noise with standard deviation $\sigma_l = 0.6\text{m}$ in the range sensor readings, and motion noise having standard deviation of $\sigma_m = 0.1\text{m}$ for each movement of the robot. Consider the grid map environment $\mathcal{G}_1$ shown in Figure 3. Assume that the initial position of the robot is v_8. We run 50 Gazebo

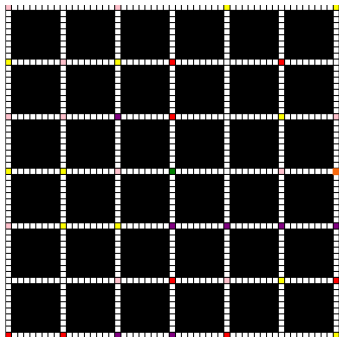
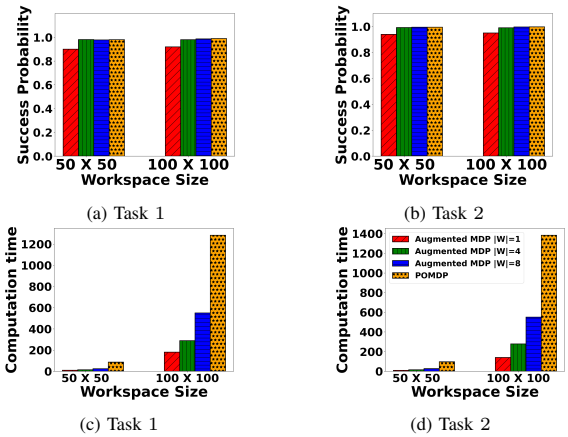


Fig. 7: Grid map having size 50×50. Different atomic propositions are marked with different color: Service Station 1 (S_1, Red), Service station 2 (S_2, Purple), Department Store (D, Green), Warehouse (W_h, Orange) Super Market (M, Yellow) and Checkpoint (C, Pink)



(a) Task 1 (b) Task 2

(c) Task 1 (d) Task 2

Fig. 8: Comparison between the AMDP and POMDP based method on the success probability and the computation time for environment having grid size 50×50 and 100×100 respectively

TABLE II: Success Probability of ROS-Gazebo setup (50 runs) and C++ setup (1000 runs) at sensor noise having standard deviation $\sigma_l = 0.6\text{m}$ and motion noise having standard deviation $\sigma_m = 0.1\text{m}$.

	Success proabability	
MDP or AMDP	Gazebo-ROS	C++ setup
$\mathcal{M}$	0.86	0.88
$\mathcal{A}_1$ ($\|W\| = 2$)	0.92	0.9
$\mathcal{A}_2$ ($\|W\| = 5$)	0.94	0.92
$\mathcal{A}_3$ ($\|W\| = 8$)	0.96	0.925

simulation for *Task* 2 and compare the success probability of MDP-based and Augmented MDP-based approaches. We also compare the ROS+Gazebo success probability with the success probability of the C++ setup, which is discussed in Section V-A.

The obtained results are shown in Table II. We construct the augmented MDP $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ having cardinality of W to be 2, 5, and 8 respectively. The simulation results corroborate the efficacy of our method in dealing with high sensor and motion noise in comparison with the MDP-based approach.

VI. Conclusion and Future Work

In this work, we present an approach to find the robot control strategy to maximize the probability of satisfying a task given as a PCTL formula in the presence of uncertainty in both action and sensing. Our algorithm is computationally as efficient as MDP based approach and provides performance almost similar to that of POMDP based approach. Thus, the proposed algorithm provides a scalable and precise solution to deal with the problem of temporal logic motion planning under localization uncertainty.

The performance of our algorithm depends on the set W (set of variance that discretizes the possible range of uncertainty). In future work, we plan to devise an algorithm that can automatically generate the optimal set of variance W, resulting in maximizing the probability satisfaction of a task. Furthermore, we plan to experiment with a real robot that uses the Vicon Motion capture system [34] for indoor localization by introducing noise in the localization measurement. We also plan to carry out outdoor experiments by using various kinds of GPSs, providing different degrees of accuracy.

References

[1] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[2] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[3] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Where's Waldo? Sensor-based temporal logic motion planning," in *ICRA*, 2007, pp. 3116–3121.

[4] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Motion planning with hybrid dynamics and temporal goals," in *CDC*, 2010, pp. 1108–1115.

[5] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[6] F. Patrizi, N. Lipovetzky, G. De Giacomo, and H. Geffner, "Computing infinite plans for LTL goals using a classical planner," in *IJCAI*, 2011, pp. 2003–2008.

[7] B. Lacerda, D. Parker, and N. Hawes, "Optimal and dynamic planning for markov decision processes with co-safe LTL specifications," in *IROS*, 2014, pp. 1511–1516.

[8] E. M. Wolff, U. Topku, and R. M. Murray, "Optimization-based trajectory generation with linear temporal logic specification," in *ICRA*, 2014, pp. 5319–5325.

[9] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Automated composition of motion primitives for multi-robot systems from safe LTL specifications," in *IROS*, 2014, pp. 1525–1532.

[10] T. Kundu and I. Saha, "Energy-aware temporal logic motion planning for mobile robots," in *ICRA*, 2019, pp. 8599–8605.

[11] M. Kloetzer and C. Mahulea, "Path planning for robotic teams based on LTL specifications and petri net models," *Discrete Event Dynamic Systems*, vol. 30, no. 1, pp. 55–79, 2020.

[12] D. Khalidi, D. Gujarathi, and I. Saha, "T* : A heuristic search based path planning algorithm for temporal logic specifications," in *ICRA*, 2020, pp. 8476–8482.

[13] P. Purohit and I. Saha, "DT*: Temporal logic path planning in a dynamic environment," in *IROS*, 2021, pp. 3627–3634.

[14] D. Gujarathi and I. Saha, "MT* : Multi-robot path planning for temporal logic specifications," in *IROS*, 2022.

[15] J. Cortes and T. Simeon, "Probabilistic motion planning for parallel mechanisms," in *ICRA*, vol. 3, 2003, pp. 4354–4359.

[16] M. Lahijanian, S. B. Andersson, and C. Belta, "Temporal logic motion planning and control with probabilistic satisfaction guarantees," *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 396–409, 2011.

[17] M. Lahijanian, J. Wasniewski, S. B. Andersson, and C. Belta, "Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees," in *ICRA*, 2010, pp. 3227–3232.

[18] M. Lahijanian, S. B. Andersson, and C. Belta, "Control of markov decision processes from PCTL specifications," in *ACC*, 2011, pp. 311–316.

[19] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press Cambridge, 2000, vol. 1.

[20] F. Ciesinski and M. Größer, *On Probabilistic Computation Tree Logic*. Springer Berlin Heidelberg, 2004, pp. 147–188.

[21] G. Norman, D. Parker, and X. Zou, "Verification and control of partially observable probabilistic systems," *Real Time Syst.*, vol. 53, no. 3, pp. 354–402, 2017.

[22] M. Bouton, J. Tumova, and M. J. Kochenderfer, "Point-based methods for model checking in partially observable markov decision processes," in *AAAI*, vol. 34, no. 06, 2020, pp. 10 061–10 068.

[23] N. Roy and S. Thrun, "Coastal navigation with mobile robots," in *NeurIPS*, 2000, pp. 1043–1049.

[24] N. Roy, "Finding approximate POMDP solutions through belief compression," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, September 2003.

[25] O. Vysotska and C. Stachniss, "Improving slam by exploiting building information from publicly available maps and localization priors," *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, vol. 85, no. 1, pp. 53–65, 2017.

[26] M. T. J. Spaan, *Partially Observable Markov Decision Processes*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 387–414.

[27] A. R. Cassandra, "Exact and approximate algorithms for partially observable markov decision processes," Ph.D. dissertation, Brown University, USA, 1998.

[28] L. Nardi and C. Stachniss, "Uncertainty-aware path planning for navigation on road networks using augmented MDPs," in *ICRA*, 2019, pp. 5780–5786.

[29] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bull. Calcutta Math. Soc.*, vol. 35, pp. 99–109, 1943.

[30] H. Kurniawati, D. Hsu, and W. S. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces." in *RSS*, 2008.

[31] "TurtleBot3," http://www.turtlebot.com, 2017.

[32] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IROS*, 2004, pp. 2149–2154.

[33] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[34] http://www.vicon.com, 2019.