Shrinking POMCP: A Framework for Real-Time UAV Search and Rescue

Conference Paper · October 2024

DOI: 10.1109/ICAA64256.2024.00016

11 authors, including:

Yunuo Zhang
Vanderbilt University
3 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Baiting Luo
Vanderbilt University
12 PUBLICATIONS **59** CITATIONS
SEE PROFILE

Daniel Stojcsics
26 PUBLICATIONS **205** CITATIONS
SEE PROFILE

Anirban Roy
Lovely Professional University
38 PUBLICATIONS **239** CITATIONS
SEE PROFILE

Shrinking POMCP: A Framework for Real-Time UAV Search and Rescue

Yunuo Zhang*, Baiting Luo*, Ayan Mukhopadhyay*, Daniel Stojcsics*, Daniel Elenius†, Anirban Roy†, Susmit Jha†, Miklos Maroti‡, Xenofon Koutsoukos*, Gabor Karsai* and Abhishek Dubey*
*Vanderbilt University, †SRI, ‡University of Szeged

***Abstract*—Efficient path optimization for drones in search and rescue operations faces challenges, including limited visibility, time constraints, and complex information gathering in urban environments. We present a comprehensive approach to optimize UAV-based search and rescue operations in neighborhood areas, utilizing both a 3D AirSim-ROS2 simulator and a 2D simulator. The path planning problem is formulated as a partially observable Markov decision process (POMDP), and we propose a novel "Shrinking POMCP" approach to address time constraints. In the AirSim environment, we integrate our approach with a probabilistic world model for belief maintenance and a neurosymbolic navigator for obstacle avoidance. The 2D simulator employs surrogate ROS2 nodes with equivalent functionality. We compare trajectories generated by different approaches in the 2D simulator and evaluate performance across various belief types in the 3D AirSim-ROS simulator. Experimental results from both simulators demonstrate that our proposed shrinking POMCP solution achieves significant improvements in search times compared to alternative methods, showcasing its potential for enhancing the efficiency of UAV-assisted search and rescue operations.**

***Index Terms*—Search and Rescue, POMDP, MCTS**

I. Introduction

Search and rescue (SAR) operations are critical, time-sensitive missions conducted in challenging environments like neighborhoods, wilderness [1], or maritime settings [2]. These resource-intensive operations require efficient path planning and optimal routing [3]. In recent years, Unmanned Aerial Vehicles (UAVs) have become valuable SAR assets, offering advantages such as rapid deployment, extended flight times, and access to hard-to-reach areas. Equipped with sensors and cameras, UAVs can detect heat signatures, identify objects, and provide real-time aerial imagery to search teams [4].

However, the use of UAVs in SAR operations presents unique challenges, particularly in path planning and decision-making under uncertainty. Factors such as limited battery life, changing weather conditions, and incomplete information about the search area complicate the task of efficiently coordinating UAV movements to maximize the probability of locating targets [3]. To address these challenges, researchers have proposed formalizing UAV path planning for SAR missions as partially observable Markov decision processes (POMDPs) [5]–[7]. POMDPs provide a mathematical framework for modeling sequential decision-making problems in uncertain environments where the system's state is not fully observable [8].

POMDP-like planning is crucial for search operations due to inherent uncertainties [9]. In UAV-based SAR, POMDPs capture uncertainties in target locations, sensor observations, and environmental conditions while optimizing UAV paths [4]. They model unknown environmental states, imperfect sensor information [10], and the complex interdependence between decisions and future observations [11]. POMDPs naturally address partial observability and long-term action consequences [10]. However, solving large-scale POMDP problems remains computationally challenging, with complexity growing exponentially with state space, observation space, and planning horizon sizes, often making exact solutions intractable for real-world applications [12].

To address this challenge, recent research has focused on online POMDP solutions, aiming to find good policies quickly by interleaving planning and execution and using sampling-based techniques to explore the belief space efficiently [13], [14]. Online POMDP frameworks have been applied to UAV path planning for SAR operations, addressing uncertainties in target motion and sensor observations [15], partial observability of victim locations and environmental hazards [16], and challenges in multi-UAV search missions [17]. Despite these advancements, computational efficiency under strict time constraints remains a critical challenge for real-time applications.

This paper presents a novel online path planner for UAVs designed to enhance the efficiency of search and rescue operations in urban environments. Our approach combines advanced simulation techniques with an innovative POMDP formulation and solution approach. This method, called Shrinking POMCP (partially observable Monte Carlo planning), guides the agent toward the next best non-sparse region for planning (we define a sparse region as a region that has probability of target appearing in that region less than a given threshold). This innovation is particularly crucial for real-world applications with strict time constraints, as it allows for more effective decision-making within limited computational resources. We demonstrate the effectiveness of the approach using an Airsim-based simulator.

The outline of this paper is as follows. We first describe the necessary background concepts (§II), followed by the problem formulation (§III), solution framework (§III-A) and description of the POMDP planning algorithm (§IV), the primary contribution of this paper. We conclude the paper with a description of metrics and experimental results (§V).

II. Background and related research

POMDPs, or Partially Observable Markov Decision Processes, are a mathematical framework for modeling decision-making in situations where an agent must make decisions in an environment that is not fully observable. A POMDP is formally defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z})$, where:

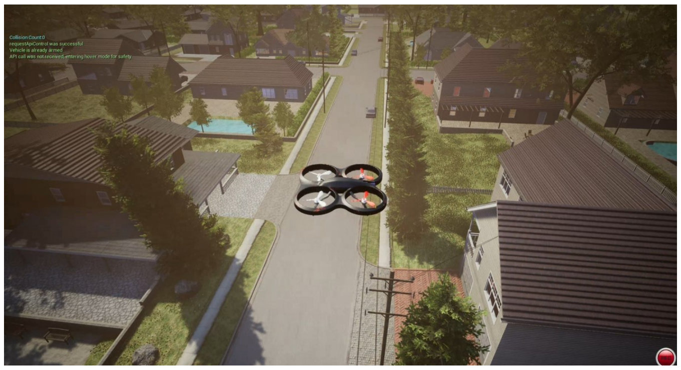
- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function, where $\mathcal{T}(s'|s, a)$ gives the probability of transitioning to state s' given action a is taken in state s.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function.
- $\mathcal{O}$ is a finite set of observations.
- $\mathcal{Z} : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \rightarrow [0, 1]$ is the observation function, where $\mathcal{Z}(o|s', a)$ is the probability of observing o given action a is taken and the system transitions to state s'.

In a POMDP, the agent maintains a belief state $b(s)$, which is a probability distribution over all possible states. The belief state is updated after each action and observation using Bayes' rule. Solving a POMDP involves finding an optimal policy π^* that maximizes the expected cumulative reward. However, exact solutions to POMDPs are computationally intractable for all but the smallest problems. As a result, much research has focused on approximate methods, including Point-based Value Iteration (PBVI) [18], Heuristic Search Value Iteration (HSVI) [19], and Monte Carlo Tree Search (MCTS) methods, such as POMCP [13]. Note that Partially Observable Monte Carlo Planning (POMCP) is an online POMDP solver that extends Monte Carlo Tree Search (MCTS) to POMDPs [13]. (MCTS) is a heuristic search algorithm for decision processes, particularly effective in large state spaces.

Key components of MCTS include: 1) **Selection:** Starting from the root node, a child selection policy is recursively applied to descend through the tree until reaching a leaf node. 2) **Expansion:** If the leaf node is not terminal and is within the computational budget, one or more child nodes are added to expand the tree. 3) **Simulation:** A simulation is run from the new node(s) according to the default policy to produce an outcome. 4) **Backpropagation:** The simulation result is then 'backed up' through the selected nodes to update their statistics.

During selection phase, UCT (Upper Confidence Bounds for Trees) [20] is used as child selection policy: $UCT = \overline{X_j} + C\sqrt{\frac{2\ln n}{n_j}}$, Where $\overline{X_j}$ is the average reward from node j, n is the number of times the parent node has been visited, n_j is the number of times child j has been visited, and C is an exploration constant.

Saisubramanian et al. [21] introduced the Goal Uncertain Stochastic Shortest Path (GUSSP) problem, a specialized form of POMDP. GUSSPs extend the Stochastic Shortest Path framework to handle goal uncertainty, maintaining a belief state over possible goal configurations. While including an observation function for goals like POMDPs, GUSSPs simplify the problem by assuming full current state observability and myopic goal observations, resulting in a more tractable solution space compared to general POMDPs.



```
Collision Count:0
requestApiControl was successful
Vehicle is already armed
API call was not received, entering hover mode for safety
```

Fig. 1. Our problem features planning the mission of a drone in a neighborhood to search for some targets. The drone is not aware of the real-locations and only have access to the likelihood of targets.

Despite advancements, existing approaches to solving POMDPs and GUSSPs face challenges in real-time applications. The computational complexity often exceeds time constraints of real-world scenarios. Even with GUSSP simplifications, the problem remains computationally demanding for large state spaces. This highlights the need for efficient algorithms that can provide quality solutions within tight time bounds, especially for robotics and autonomous systems requiring rapid decisions.

A. AirSim and ROS2

Microsoft AirSim is an open-source simulator for autonomous vehicles, developed by Microsoft Research [22]. It was designed to bridge the gap between simulation and reality in the field of artificial intelligence, particularly for drones and self-driving cars. AirSim provides a platform for researchers and developers to test and train AI algorithms in a realistic, physics-based environment without the risks and costs associated with real-world testing [22]. The simulator leverages Unreal Engine [23] to create detailed environments and supports various autonomous system sensors like cameras, GPS, and IMUs [24]. To provide communication and integrate external processing, the system supports the Robot Operating System v2 (ROS2) [25]. AirSim has been upgraded for DARPA by Microsoft and extended with STR Algorithm Development Kit to provide mission generation and randomization.

III. Problem Formulation

Overall, the problem we are interested in is to perform autonomous target localization with multiple targets in an urban environment using an unmanned aerial vehicle (UAV) (fig. 1). A key aspect of this problem is the uncertainty in target locations. The quadrotor does not possess prior knowledge of exact target positions. Instead, it maintains a probabilistic map representing its belief state, which is continuously updated based on a perception system, which eventually leads to update of belief system. The perception system is also responsible for the tracking and identification of the target if they are in the

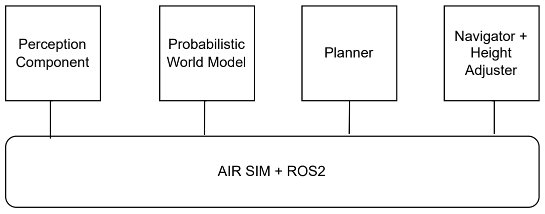


Fig. 2. Four key components of our approach. This paper focuses on the planner component.

range of the camera of the drone. Note that it is assumed that the quadrotor has only one camera that is pointed in the direction of travel and can only see in a limited area.

Solution to the problem requires perception, belief update, planning and navigation together to generate trajectories that minimize the overall cost function, increases the likelihood of finding targets, avoiding no-fly zones, and minimizing overall flight time.

A. Solution Approach

Figure 2 shows our software system for mission execution. The perception module detects objects with specific attributes and identifies relationships between them, handling environmental novelties due to varying camera views, occlusion, and weather perturbations. While state-of-the-art object detectors like YOLO world [26] perform well in closed-world setups, they struggle with novel attributes and relations in SAR operations. Our two-stage approach first detects generic objects, then uses a vision-language model (VLM) [27] to detect attributes and relations. VLMs, trained on diverse datasets, can handle novelties effectively.

The probabilistic world model maintains and updates the belief map using raw sensor information received from the perception components, and other sensors such as inertial measurement unit outputting the current drone position, obstacle map, and belief map. It efficiently represents probability distributions and their relations, generating updates from a formal specification derived from mission parameters. The world model also maintains the flight rules including location of no-fly areas. The planner component is responsible for generating flight plans – sequence of waypoints. The navigator component is responsible for finding the shortest and safest path (while avoiding collisions) in the 3-D environment. The height adjustment component is utilized to increase or decrease the height of the drone if no-safe path can be found. Overall, the system works as an interactive protocol between the planner and the navigator. When the planner publishes a waypoint (using ROS2), the determined waypoint is then passed to the navigator component, which guides the quadrotor while ensuring collision avoidance. This process continues until the drone reaches the waypoint or the navigator determines it's unreachable. In either case, a new decision epoch is initiated, allowing the drone to adapt dynamically to its environment based on the most current information.

Note while there are innovations in each of the above component, due to space limitations this paper is restricted to the description of the planning algorithm.

IV. Planner

In this section, we describe our POMDP framework and our approach for computing near-optimal actions for the POMDP. A major bottleneck in directly using online search algorithms (e.g., POMCP [13]) in our setting is scalability—the management of the UAV at each time step can only afford limited latency. To address this problem, we propose a "shrinking POMCP" algorithm. Intuitively, once a search tree is constructed, we hypothesize that the agent can traverse the most promising actions down the nodes of the search tree, provided it is in a sparse likelihood region of the map. For example, consider that the agent is in the lower left corner of a grid, with the target likely in the upper left corner. The agent can construct a search tree once and (likely) take multiple steps toward the target region without recomputing again. This technique is designed to dynamically reduce the decision space as planning progresses. The key innovation lies in its ability to guide the agent towards the next best non-sparse region for planning, effectively concentrating computational resources on the most promising areas of the state space. Note that to reduce complexity, we discretize the 3-D state space into a two dimensional slice at a given mission height, set to ensure that the perception component can operate efficiently. If required, the operating height is changed and the 2-D planner can be invoked again. The probabilistic world model can generate the belief distribution at any given height.

A. POMDP Formalization

Decision Epoch: In our framework, the decision epoch is defined as the moment when the solver is triggered to determine the next waypoint for the quadrotor. This dynamic decision-making process occurs at discrete time intervals, transitioning from time t to $t+1$, and is initiated by specific events rather than at fixed time intervals. In practice, these events can be monitored and controlled by a meta-controller. Specifically, the solver is activated to make a new decision when one of two conditions is met: either the quadrotor successfully reaches its previously issued waypoint, or it encounters a situation where the current waypoint is unreachable due to obstacles obstructing all valid paths. At each decision epoch, the solver receives a comprehensive update of the system's state, including the most recent belief states, the obstacles detected by the quadrotor's cameras in its immediate vicinity, the quadrotor's current position, and a request for the next waypoint. This event-driven approach to decision epochs ensures that the system remains responsive to the dynamic nature of the environment and the quadrotor's progress, allowing for adaptive and efficient navigation strategies.

States: In our POMDP framework, we define the state space $\mathcal{S}$ to encompass the position of the quadrotor and the locations of all targets. Let $s_t \in \mathcal{S}$ denote the pre-decision state at time t. Each state s_t is represented as a $(3+2M)$-dimensional vector:

$$s_t = [x_q, y_q, z_q, x_1, y_1, \ldots, x_M, y_M] \tag{1}$$

where

- $(x_q, y_q) \in \mathbb{R}^2$ represents the quadrotor's horizontal position
- $z_q \in \mathbb{R}^+$ denotes the quadrotor's altitude
- $(x_i, y_i) \in \mathbb{R}^2$ represents the location of the i-th target, for $i = \{1, \ldots, M\}$

Thus, the complete state space S is of dimensions equalling $\mathbb{R}^2 \times \mathbb{R}^+ \times (\mathbb{R}^2)^M$. This formulation captures the full spatial configuration of the system at any given time t, incorporating both the UAV's three-dimensional position and the two-dimensional locations of all M targets within the neighborhood area. To simplify the problem, we discretize the operational area into a grid. Let the original map be a square of side length L. We partition this map into an $N \times N$ grid, where each cell represents an area of $(L/N) \times (L/N)$ (in our implementation, $N = 20$, resulting in 20m ×20m cells). Formally, we can define the grid G as:

$$G = \{(i, j) \mid i, j \in \{0, 1, \ldots, N-1\}\} \tag{2}$$

At each decision epoch, the agent's position is mapped to one of these grid cells.

Actions: The action space $\mathcal{A}$ consists of four cardinal directions:

$$A = \{\text{West}, \text{South}, \text{East}, \text{North}\} \tag{3}$$

Each action $a \in \mathcal{A}$ corresponds to moving to an adjacent grid cell in the specified direction. For an agent in cell (i, j) at time t, an action $a \in \mathcal{A}$ results in a transition to a new cell (i', j') at time $t+1$, where the new coordinates depend on the chosen direction.

The actual waypoint w_{t+1} within the chosen grid cell is determined by finding a *valid* position closest to the quadrotor's current position x_t:

$$w_{t+1} = \underset{w \in V(i',j')}{\arg\min} \|w - x_t\| \tag{4}$$

where $V(i', j')$ is the set of valid positions within the grid cell (i', j'). We describe the translation of the grid-based decision into a specific waypoint for the quadrotor later.

.

Observation: In our POMDP framework, the observation space $\mathcal{O}$ provides partial information about the state. At each time step t, an observation $o_t \in \mathcal{O}$ is defined as a tuple (γ_t, B_t), where $\gamma_t = (x_q, y_q, z_q) \in \mathbb{R}^3$ represents the exact current position of the quadrotor, and B_t is the updated belief state. The belief state B_t is a probabilistic map over the grid G, where for each cell $(i, j) \in G$, $B_t(i, j) \in [0, 1]$ represents the probability of a target being present in that cell.

Reward: The reward function for our POMDP framework is designed to guide the quadrotor agent in efficiently locating targets within a specified neighborhood area. The reward structure comprises two primary components: target capture and token capture. This dual-component design balances the agent's focus between achieving the primary objective and maintaining comprehensive environmental awareness.

The reward function R is formally defined as:

$$R = R_{\text{target}} + \alpha \cdot R_{\text{token}} \tag{5}$$

where R_{target} denotes the reward for target capture, R_{token} represents the reward for token capture, and α is a hyperparameter controlling the relative importance of token capture.

Target Capture Reward (R_{target}): The target capture component directly addresses the primary objective of the simulation. It is defined as a binary function:

$$R_{\text{target}} = \begin{cases} 1, & \text{if the agent captures a target} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

This component provides a significant positive reinforcement upon successful target capture, incentivizing the agent to prioritize navigation towards known or suspected target locations.

Token Capture Reward (R_{token}): The token capture component serves as an exploration incentive, encouraging comprehensive coverage of the environment while prioritizing areas of higher probability. It is defined as:

$$R_{\text{token}} = \sum_i \mathbb{1}(i) \cdot P_i \tag{7}$$

Where $\mathbb{1}(i)$ is an indicator function for cell i, and P_i is the normalized probability token value for cell i. Note that

$$\mathbb{1}(i) = \begin{cases} 1, & \text{if cell } i \text{ is visited for the first time} \\ 0, & \text{if cell } i \text{ has been visited before} \end{cases} \tag{8}$$

and

$$P_i = \frac{p_i}{\sum_j p_j} \tag{9}$$

where p_i represents the raw probability value assigned to cell i in the probabilistic map, and P_i is its normalized form.

This cumulative reward structure incentivizes the exploration of new cells while weighting the reward based on the likelihood of finding targets in each cell. Cells with higher normalized probabilities yield greater rewards upon first visit, potentially facilitating the discovery of targets in areas deemed more promising by the probabilistic map.

The hyperparameter α in eq. (5) allows for fine-tuning of the agent's behavior, balancing the emphasis between target acquisition and environmental exploration. A higher α value encourages more thorough exploration, while a lower value prioritizes immediate target capture.

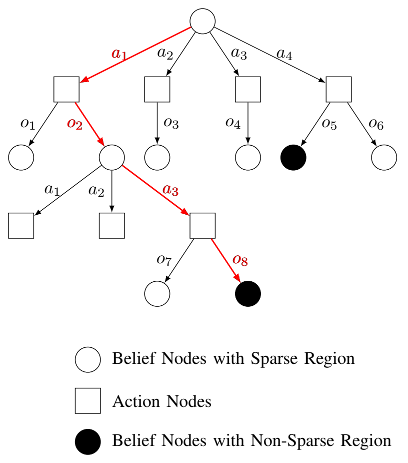


Fig. 3. A belief tree constructed by the Shrinking POMCP approach, illustrating its unique decision-making process. Circular nodes represent belief states, with their normalized probability $P(s)$ and quadrotor position $\gamma(s) = (x_q, y_q)$. Black-filled circles indicate non-sparse regions where $P(s) > P_\varepsilon$. Square nodes represent actions. The red arrows show the action sequence $\{a_1, a_2, \ldots, a_k\}$ decided by the agent, where each $a_i = \arg\max_a Q(b_i, a)$. This sequence terminates upon reaching either a non-sparse region (black node) or the maximum depth max_level. Unlike standard MCTS, this approach efficiently guides the agent towards high-probability areas, terminating when $P(b_{k+1}) > P_\varepsilon$ or $k =$ max_level, thus avoiding goal sampling oscillation.

Algorithm 1 Shrinking POMCP

```
Require: Initial belief $b_0$, max_iterations, max_time,
    max_level, $P_\varepsilon$
Ensure: Action sequence $A$
 1: Initialize belief tree $T$ with root node $b_0$
 2: while iterations $<$ max_iterations and time $<$ max_time
    do
 3:     $s_0 \sim b_0$
 4:     SimulateV($s_0$, $b_0$, 0)
 5: end while
 6: return GetActionSequence($b_0$, max_level, $P_\varepsilon$)
```

B. Shrinking MCTS

At each decision epoch, the algorithm constructs a belief tree T with alternating action (A) and belief (B) nodes. The root node $b_0 \in B$ represents the initial belief state (Line 1 in algorithm 1). Figure 3 illustrates this tree structure, where action nodes are represented by squares and belief nodes by circles. The red arrows represent the action sequence decided by the agent.

The algorithm begins by sampling a random state $s_0 \sim b_0$ to initialize the root node of the tree (Line 3 in algorithm 1). This initialization step ensures that the search starts from a plausible state within the current belief distribution. The tree expansion process involves a series of iterations, each comprising four main phases: selection, expansion, simulation, and backpropagation. In the selection phase, the algorithm traverses the tree from the root node using the Upper Confidence Bound for Trees (UCT) strategy. For a belief node b, the best action a^*

Algorithm 2 SimulateV(s, b, depth)

```
 1: if IsTerminal($s$) or depth $>$ max_depth then
 2:     return 0
 3: end if
 4: if $b$ is leaf node then
 5:     Expand($b$)
 6:     return Rollout($s$, $b$)
 7: end if
 8: $a = \arg\max_a \left[Q(b,a) + c\sqrt{\frac{\log(N(b))}{N(b,a)}}\right]$
 9: $(s', o, r) = \mathcal{G}(s, a)$
10: if $b$ has no child corresponding to $o$ then
11:     $b' =$ CreateNewBeliefNode$(b, a, o)$
12: else
13:     $b' = b$.child$(a, o)$
14: end if
15: $q = r + \gamma \cdot$ SimulateV$(s', b', \text{depth} + 1)$
16: UpdateStats($b$, $a$, $q$)
17: return $q$
```

is selected according to the equation:

$$a^* = \arg\max_a \left[Q(b,a) + c\sqrt{\frac{\log(N(b))}{N(b,a)}}\right] \tag{10}$$

where $Q(b,a)$ is the estimated value of action a in belief state b, $N(b)$ is the number of visits to node b, $N(b,a)$ is the number of times action a was selected from belief state b, and c is an exploration constant (Line 8 in algorithm 2). This selection strategy balances exploitation of known high-value actions with exploration of less-visited branches.

The simulation phase employs a simulator $\mathcal{G}$, which accepts a state and an action as inputs. This simulator produces three outputs: the probable subsequent state derived from the transition function, the associated observation, and the corresponding reward. This process can be formally represented as:

$$(s', o, r) \sim \mathcal{G}(s, a)$$

where s' is the next state, o is the observation, and r is the reward, all generated based on the current state s and action a (Line 9 in algorithm 2).

The expansion phase occurs when the selected action leads to an unexplored observation. In this case, a new belief node is added to the tree (Line 11 in algorithm 2). If the observation has been encountered before, the algorithm follows the existing path (Line 13 in algorithm 2). When a leaf node is reached during the simulation phase, the algorithm expands it by creating child nodes for all possible actions. Then, a rollout is performed till a terminal node. The result of this simulation is then propagated back up the tree in the backpropagation phase, updating the statistics (Q-values and visit counts) of all traversed nodes (Line 16 in algorithm 2).

The key innovation of the Shrinking MCTS algorithm lies in its decision-making process, which aims to move the agent to the next best non-sparse region while avoiding goal sampling

Algorithm 3 GetActionSequence(b, max_level, P_ε)

```
1: $A = []$, depth $= 0$
2: while depth $<$ max_level and $P(b) \le P_\varepsilon$ do
3:     $a = \arg\max_a Q(b, a)$
4:     $A$.append($a$)
5:     $b \sim \mathcal{G}(s, a)$
6:     depth$+= 1$
7: end while
8: return $A$
```

oscillation. Each belief node s in the tree stores two critical pieces of information: the quadrotor's position $\gamma(s) = (x_q, y_q)$ and the normalized probability $P(s)$ at that position. A belief node is classified as Non-Sparse if its normalized probability exceeds a predefined threshold, i.e., $P(b) > P_\varepsilon$. In fig. 3, a Belief Node with Non-Sparse Region is represented as a circle filled with black.

Unlike standard MCTS, which typically selects a single action at each step, our Shrinking approach determines an entire action sequence. As shown in fig. 3, this sequence (represented by red arrows) guides the agent towards high-probability areas. The decision sequence is determined by traversing the tree from the root, selecting the best action at each level until either a maximum depth max_level is reached or a Belief Node with Non-Sparse Region is encountered. In the figure, we can see this process leading to the action sequence $\{a_1, a_2, a_3, a_8\}$, terminating at a black node representing a non-sparse region.

Mathematically, this process can be described as taking k actions $\{a_1, a_2, \ldots, a_k\}$ such that $a_i = \arg\max_a Q(b_i, a)$ and $b_{i+1} = \tau(b_i, a_i, o_i)$; the constraints we impose for termination are $P(b_{i+1}) \le P_\varepsilon$ or $k \le$ max_level, where $\tau(b, a, o)$ represents the belief update function given action a and observation o. The sequence terminates when either $P(b_{k+1}) > P_\varepsilon$ or $k =$ max_level, as illustrated in fig. 3 where the sequence ends at a black node (non-sparse region). This approach efficiently guides the agent towards promising areas while avoiding the oscillation often seen in standard MCTS implementations.

This approach allows the algorithm to dynamically shrink the decision space by focusing on actions that lead to non-sparse regions, effectively guiding the agent towards areas of the state space with higher certainty. By doing so, the Shrinking POMCP algorithm can potentially overcome the limitations of traditional POMDP solvers in environments with vast or sparse state spaces, leading to more efficient and effective planning in partially observable domains.

Rollout In Monte Carlo planning, rollouts present a significant computational challenge. These rollouts are essential for approximating the value of leaf nodes within the search tree using a computationally inexpensive strategy. Our approach employs the A* algorithm [28] as the rollout policy, balancing efficiency and effectiveness in pathfinding. While each individual rollout is computationally inexpensive, the cumulative cost of performing thousands of rollouts for each decision becomes a significant bottleneck. To address this challenge and improve overall performance, we implement the A* algorithm using C programming language [29]. We found that this approach led to better results than standard random rollouts.

Each belief node s in our tree structure encapsulates information about the quadrotor's position $p(s)$. This positional information is represented at two distinct levels of granularity:

1) Fine-grain level: The exact position on a high-resolution map of dimensions $L \times L$.
2) Discretized level: A coarser $N \times N$ grid, where each cell corresponds to a region of the fine-grain map.

While the Monte Carlo Tree Search (MCTS) simulation operates on the discretized $N \times N$ grid for computational efficiency, the rollout process necessitates a more precise position determination. Given a current state and action, we first identify the target cell in the discretized grid. The challenge then becomes determining the exact position within this target cell.

To address this, we employ a sampling-based approach that balances accuracy and computational efficiency. Let γ_c represent the set of all possible positions in the current cell, and γ_t represent the set of all possible positions in the target cell. We define a sampling function $S(\gamma_t)$ that returns a subset of positions from γ_t.

From this sampled subset, we identify the set of valid positions $V(S(\gamma_t))$:

$$V(S(\gamma_t)) = \{\gamma \in S(\gamma_t) \mid \gamma \text{ is a valid position}\} \tag{11}$$

The next exact position γ_{next} is then determined by finding the position in $V(S(\gamma_t))$ that minimizes the Euclidean distance from the current position $\gamma_{current}$:

$$\gamma_{next} = \underset{\gamma \in V(S(\gamma_t))}{\arg\min} ||\gamma - \gamma_{current}|| \tag{12}$$

where $|| \cdot ||$ denotes the Euclidean distance.

Once γ_{next} is determined, we calculate the rollout value by running the A* algorithm from $\gamma_{current}$ to γ_{next}, using the given obstacle map. Let $L(\gamma_{current}, \gamma_{next})$ denote the path length returned by A*. The rollout value R is then computed as a function of this path length:

$$R = f(L(\gamma_{current}, \gamma_{next})) \tag{13}$$

where f is a monotonically decreasing function. This formulation ensures that shorter paths, indicating easier navigation, result in higher rollout values, while longer paths, suggesting more complex navigation, yield lower values.

C. Height Adjustment

Our planning approach initially assumes constant altitude but incorporates an adaptive height adjustment strategy to balance obstacle avoidance with smooth flight patterns, mitigating undesirable "up and down" motions due to system noise.

The quadrotor starts at altitude h_{init}. For each target cell C_t in the planned path, we evaluate the number of valid positions $N_v(C_t, h)$ at current height h:

$$N_v(C_t, h) = |\{\gamma \in C_t \mid \gamma \text{ is valid at height } h\}| \tag{14}$$

We define an “obstacle_tolerance_threshold” τ. If $N_v(C_t, h) \geq \tau$, the waypoint’s altitude remains unchanged. Otherwise, we incrementally adjust the height:

$$h_{new} = \min(h + \Delta h, h_{max}) \tag{15}$$

where Δh is the height increment and h_{max} is the maximum allowed height. This process repeats until $N_v(C_t, h_{new}) \geq \tau$ or $h_{new} = h_{max}$.

If $N_v(C_t, h_{max}) < \tau$, we designate the cell as a no-fly zone and re-execute the Monte Carlo Tree Search (MCTS) algorithm for an alternative path. This strategy ensures safe obstacle avoidance while minimizing unnecessary altitude changes, resulting in smoother and more efficient flight trajectories.

V. Analysis and Evaluation

We evaluate our proposed framework using both the AirSim-ROS2 simulator and the two-dimensional simulator described in section III-A and report the performance of Shrinking POMCP against baseline.

A. Environment Simulators

We test our approach in two simulation environments:

1) **3D AirSim-ROS2 Simulator:** This advanced environment incorporates the full functionality of our framework, including the Probabilistic World Model and Navigator components.
2) **Two-dimensional Simulator:** In this simplified environment, we test our approach without considering altitude effects. The Probabilistic World Model and the Navigator are replaced with surrogate ROS2 nodes that provide equivalent functionality, allowing for efficient testing and validation of our algorithms.

B. Hyperparameters

The testing environment for this mission scenario is set within a 400m $\times$ 400m map, with a strict time constraint of 5 minutes to locate all targets. The quadrotor’s initial altitude is set at 10 meters, providing a balance between coverage area and detail resolution.

Several hyperparameters are adjusted to optimize the search strategy. The discount factor in the POMDP framework is tested with values of 0.8, 0.9, and 0.995, influencing the balance between immediate and future rewards. The α value in the reward function discussed in section IV, which affects the weighting of different reward components, is varied among 0, 1, and 10. Shrinking MCTS runs 3000 iterations for every decision epoch, with the exploration parameter in UCT set to $\sqrt{2}$. For vertical navigation, the height adjustment parameter δh (discussed in section IV-C) is set to 3m, with a maximum allowable altitude of 30m. These parameters collectively define the operational constraints and decision-making framework for the quadrotor’s search mission.

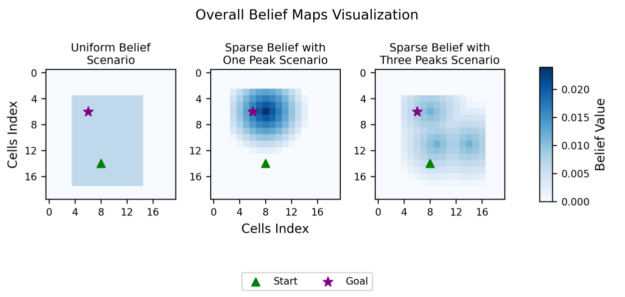


Fig. 4. Environments for different belief scenarios. Left: Uniform belief distribution across the environment. Center: Sparse belief with a single peak, indicating high certainty in one area. Right: Sparse belief with three peaks, representing multiple areas of high certainty. The green triangle (▲) represents the start position, and the purple star ($\star$) indicates the goal position. Color intensity corresponds to belief value, with darker blue indicating higher belief.

C. Baselines

In our experiments, we evaluate the performance of our Shrinking POMCP approach against three baseline methods: Lawnmower algorithm, Greedy algorithm, and standard MCTS (POMCP without shrinking). We test these methods on two types of belief maps: Uniform belief map and Sparse belief map with one peak. All methods are compared on the same map, with identical no-fly zones and starting agent positions for each scenario to ensure a fair comparison.

Lawnmower Algorithm: This baseline performs a systematic search in non-zero belief areas. At the start of each episode, the agent moves to the nearest non-zero probability position on the probability map. It then executes a lawnmower pattern search, systematically covering the non-zero belief region until all targets are found. This method ensures complete coverage of the search area but may not be optimal in terms of efficiency.

Greedy Algorithm: This approach makes local decisions based on immediate information. At each step, the agent selects its next position by choosing the adjacent cell with the highest value in the resized array. While this method can be effective in quickly identifying high-probability areas, it may suffer from getting trapped in local maxima.

Monte Carlo Tree Search (MCTS): We implement the standard POMCP algorithm without our proposed shrinking approach as a baseline. This method makes decisions only for cardinal directions (WEST, SOUTH, EAST, NORTH), moving to the next adjacent cell in the discretized grid at each step. This baseline allows us to directly compare the performance gains achieved by our Shrinking POMCP approach.

D. Results (Two-Dimensional)

Belief Maps: Three types of initial belief distributions are tested in this scenario. The first is a Uniform Belief (fig. 4), where probabilities are evenly distributed across the entire map. The second is a Sparse Belief with One Peak (fig. 4), characterized by a high probability concentration at a single location that gradually diffuses outward. The third is a Sparse Belief with Three Peaks (fig. 4), featuring multiple

TABLE I
HYPERPARAMETER SWEEP RESULTS: DF REPRESENTS DISCOUNT FACTOR, RA IS REWARD ALPHA. SE IS STANDARD ERROR OF MEAN. BOLD NUMBERS REPRESENT THE OPTIMAL COMBINATION OF DISCOUNT FACTOR AND REWARD ALPHA CHOSEN FOR EACH BELIEF TYPE, BASED ON THE BEST AVERAGE PERFORMANCE.

DF	**RA**	**Uniform Belief**		**One Peak**		**Three Peaks**	
		Mean	SE	Mean	SE	Mean	SE
0.8	0	17.3	3.4	17.2	5.1	6.2	1.0
0.8	1	24.3	3.4	15.2	4.8	8.4	3.5
0.8	10	37.4	14.5	12.5	4.8	12.6	6.0
0.9	0	18.1	9.9	15.9	5.1	6.1	0.9
0.9	1	16.1	7.7	12.2	5.0	15.4	6.3
0.9	10	15.3	8.4	12.4	4.9	6.7	3.3
0.995	0	**5.7**	**1.6**	12.0	4.9	11.3	4.7
0.995	1	7.3	1.6	12.2	5.0	6.7	2.0
0.995	10	22.7	9.7	**11.3**	**4.8**	**3.0**	**0.9**

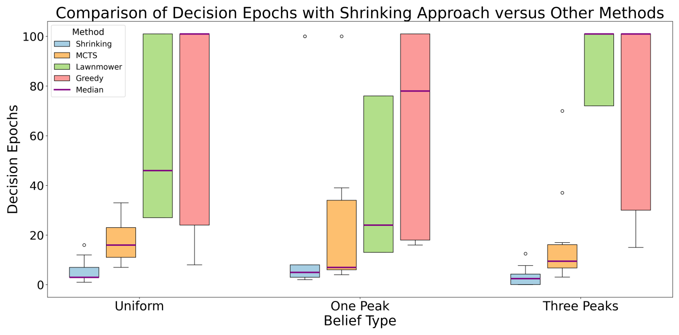


Fig. 5. Comparison between our Shrinking approach and other methods (MCTS, Lawnmower, and Greedy algorithms). Shrinking POMCP requires significantly fewer decision epochs to locate all targets across all belief types. Colors represent different approaches: Blue - Shrinking, Orange - MCTS, Green - Lawnmower, Red - Greedy.

areas of high probability concentration. In all scenarios, the belief distribution changes to reflect different levels of prior knowledge about the environment.

Hyperparameter Sweep: We conducted a comprehensive hyperparameter sweep to evaluate the performance of our framework under different conditions and compare it with baseline algorithms. The experiments were performed on the three types of belief maps:

Uniform Belief Scenario: For the Uniform Belief scenario as shown in table I, we explored different combinations of discount factors (0.8, 0.9, 0.995) and reward alphas (0, 1, 10). The performance metric is the number of decision epochs required to locate all targets, with a maximum limit of 100 epochs. The results indicate that higher discount factors generally lead to better performance. This suggests that in uniformly distributed belief scenarios, our framework benefits from considering long-term rewards more heavily.

Sparse Belief with One Peak Scenario: In the Sparse Belief with One Peak scenario as shown in table I, we used a similar experimental setup. Similar as Uniform Belief, higher discount factors generally lead to better performance.

Sparse Belief with Three Peaks Scenario: For the Sparse Belief with Three Peaks scenario, higher discount factors, particularly when combined with higher reward alpha values, indicate better performance for our approach. This suggests that in more complex belief distributions with multiple high-probability areas, our framework benefits from both long-term planning (higher discount factors) and a stronger emphasis on immediate rewards (higher reward alpha).

Based on the results of our hyperparameter sweep, we selected the combination of discount factor and reward alpha that achieved the best average performance for each belief type. These optimal hyperparameter values were then used consistently in both our 2D environment and AirSim environment experiments, ensuring a standardized approach across different simulation platforms.

Comparison between Shrinking POMCP and Other Methods: The performance comparison between our Shrinking approach and other methods (MCTS, Lawnmower, and Greedy algorithms) demonstrates the superior efficiency of our proposed method. As shown in Figure 5, the Shrinking approach consistently requires significantly fewer decision epochs to locate all targets across both Uniform and One Peak belief types.

The key advantage of our Shrinking POMCP lies in its ability to output an action sequence at each decision epoch, rather than a single action. This enables the agent to efficiently navigate to the next best non-sparse region for planning in every epoch. By doing so, our approach effectively mitigates a common challenge faced by traditional POMCP, where sampling goals can cause the agent to oscillate between different actions. The result is a more decisive and efficient search strategy, as evidenced by the consistently lower number of decision epochs required across different belief scenarios.

E. Results (AirSim-ROS2)

No-Fly-Zones: No-fly zones provide spatial and temporal constraints for quadrotor operations in simulated environments. These zones are defined areas that the quadrotor should not enter. Each no-fly zone is represented by geometric boundaries, which could be 3D shapes like cubes, cylinders, or complex polygons. A key feature of AirSim's no-fly zones is their temporal aspect. Each zone includes `no_earlier_than` and `no_later_than` parameters, specifying the time window during which the restriction is active.

Entities of Interest: The configuration defines specific vehicles as targets, each with unique attributes (e.g., a red SEDAN). As the quadrotor explores, its cameras identify vehicles matching these criteria. The system processes captured images to update a belief map, reflecting the likelihood of finding target entities in different locations.

Evaluation Scenarios: We evaluate the components by running them through simulations in the mission scenario (example of one scenario shown in figure 6). A **scenario** is defined as a single mission and environmental configuration, consisting of mission-related aspects (e.g., target types and arrangements, areas of interest, keep-out zones, and belief

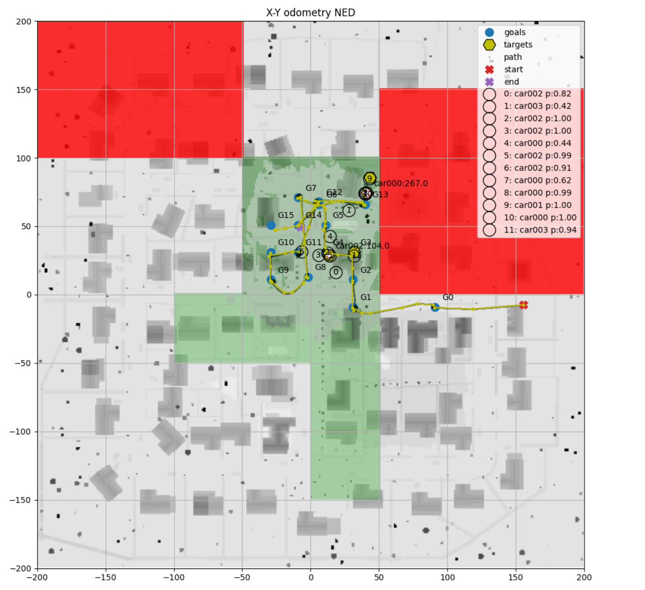


Fig. 6. X-Y odometry plot (in North East Down coordinates) of a quadrotor drone's area search mission, showing the drone's path (yellow), ground truth targets (hexagons), detected targets (numbered circles), no-fly zones (red), buildings (grey), and search areas (green). The legend lists detected targets with their probabilities.

maps) and environmental factors (e.g., weather, time of day, and camera noise).

Metrics: Overall, the success of the mission is evaluated using a set of metrics: **COP Completeness**[1]**:** Percentage of correctly reported target information elements out of total target information elements. It's calculated by maintaining a running status of each information element (e.g., type, location) throughout the trial, determining if it's in scope and correctly reflected in the COP. The metric is presented as a distribution with mean and 95 % CI for each evaluation condition. **COP Accuracy:** Percentage of correctly reported targets out of total reports. This metric is captured per trial and presented as a distribution with mean and 95 % CI for each evaluation condition, reflecting the accuracy of the system's target identification and reporting; and **COP Reporting Latency:** Average time between a target change and its correct reporting. It's calculated using time points for every relevant change during each trial at a 1-second resolution. The metric is presented as a distribution with mean and 95% CI for each evaluation condition, indicating the system's responsiveness to target changes.

The figure (fig. 6) describes a single run of the system for one of the scenarios. The map is divided into different zones: green areas represent regions of interest with non-zero probability of containing targets, grey areas indicate obstacles, like buildings and trees, and red areas denote no-fly zones. The drone's path, shown in yellow, navigates efficiently through the green areas, avoiding obstacles and restricted zones. In this run, our planner algorithm located all 4 cars of interest, marked

[1]COP stands for common operating picture

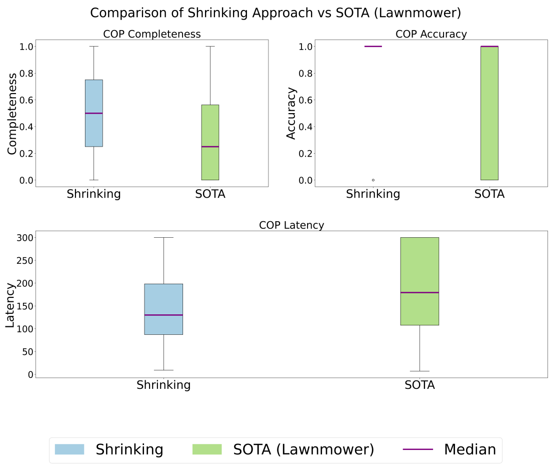


Fig. 7. Performance comparison between the Shrinking POMCP approach and SOTA (Lawnmower) method. The box plots show the distribution of COP Completeness, COP Accuracy, and COP Latency across multiple runs. The Shrinking approach (light blue) consistently outperforms the SOTA method (light green) across all metrics, with higher completeness and accuracy, and lower latency. Median values are indicated by purple lines.

as numbered targets on the map using the novel perception component developed by SRI in the same DARPA project. The drone's path demonstrates an efficient search strategy, entering high-probability areas directly and minimizing time spent in low-probability regions. This efficiency is due to our shrinking approach in the algorithm, allowing the quadrotor to make faster decisions and focus on promising areas.

Figure 7 illustrates the performance of our Shrinking POMCP framework compared to the SOTA Baseline (lawnmover search pattern with YOLO v8 World perception) across multiple scenarios and runs. We show variance to remove experimental bias. Our method consistently outperforms the Baseline in all three key metrics: COP Completeness, COP Accuracy, and COP Latency. For COP Completeness, our approach shows a higher median and a more concentrated distribution, indicating more consistent and thorough coverage. In terms of COP Accuracy, our Shrinking approach demonstrates superior performance, with a median accuracy of 1.0 compared to the SOTA's lower and more varied distribution. The average COP Accuracy for our approach is **0.81**, while the SOTA achieves only **0.57**, highlighting our method's significantly higher precision in correctly reporting targets. Regarding COP Latency, our framework exhibits lower latency with a tighter distribution, suggesting faster and more consistent response times. These enhancements are due to our framework's efficient search strategy, enabling the quadrotor to make faster decisions and prioritize high-probability areas.

VI. Conclusion

In this paper, we presented an optimized approach for UAV-based search and rescue operations in neighborhood areas. We developed a realistic simulator using AirSim and ROS2, and formulated the path planning problem as a POMDP. Our Shrinking POMCP approach addresses time constraints in SAR missions. Experimental results demonstrate that this method significantly outperforms alternatives, locating all targets in fewer decision epochs. This suggests our solution can enhance the efficiency of UAV-assisted SAR missions, saving critical time in emergencies.

VII. Discussion and Future Work

While Shrinking POMCP shows promise in adapting to non-stationary environments, future research could explore integrating function approximation techniques into the planning process. Neural network approximators could be used to learn the inherent uncertainty in the environment [30], potentially improving the algorithm's ability to adapt to changes. Additionally, leveraging learned approximators to accelerate MCTS convergence [31] could enhance computational efficiency. Combining these approaches with our Shrinking POMCP could lead to a more robust and efficient algorithm capable of rapid adaptation in non-stationary environments while maintaining computational tractability, particularly in domains with large state spaces and complex dynamics.

Acknowledgments

This material is based upon work sponsored in part by the by the Defense Advanced Research Projects Agency (DARPA) under its Assured Neuro Symbolic Learning and Reasoning (ANSR) project and the US Air Force Research Laboratory (AFRL) and in part by the National Science Foundation Grant 2238815. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA. The authors would like to thank Microsoft for creating the AirSim simulator and STR for developing the challenge scenarios for the ANSR program.

References

[1] J.-H. Ewers, D. Anderson, and D. Thomson, "Deep reinforcement learning for time-critical wilderness search and rescue using drones," 2024.

[2] S. Aronica, F. Benvegna, M. Cossentino, S. Gaglio, A. Langiu, C. Lodato, S. Lopes, U. Maniscalco, and P. Sangiorgi, "An agent-based system for maritime search and rescue operations," in *Workshop From Objects to Agents*, 2010.

[3] L. Lin and M. A. Goodrich, "Uav intelligent path planning for wilderness search and rescue," in *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS'09, p. 709–714, IEEE Press, 2009.

[4] S. Waharte and A. Trigoni, "Supporting search and rescue operations with uavs," *2010 International Conference on Emerging Security Technologies*, pp. 142–147, 2010.

[5] F. Trotti, A. Farinelli, and R. Muradore, "An online path planner based on pomdp for uavs," in *2023 European Control Conference (ECC)*, pp. 1–6, 2023.

[6] M. Ahmadi, N. Jansen, B. Wu, and U. Topcu, "Control theory meets pomdps: A hybrid systems approach," *IEEE Transactions on Automatic Control*, vol. 66, no. 11, pp. 5191–5204, 2021.

[7] B. R. O. Floriano, G. A. Borges, H. C. Ferreira, and J. a. Y. Ishihara, "Hybrid dec-pomdp/pid guidance system for formation flight of multiple uavs," *J. Intell. Robotics Syst.*, vol. 101, mar 2021.

[8] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[9] N. Roy, G. Gordon, and S. Thrun, "Finding approximate POMDP solutions through belief compression," *Journal of Artificial Intelligence Research*, vol. 23, pp. 1–40, 2005.

[10] M. T. J. Spaan, "Cooperative active perception using POMDPs," in *Proceedings of the 2008 AAAI Workshop on Advances in Preference Handling*, AAAI Press, July 2008.

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

[12] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987.

[13] D. Silver and J. Veness, "Monte-carlo planning in large pomdps," in *Advances in Neural Information Processing Systems* (J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds.), vol. 23, Curran Associates, Inc., 2010.

[14] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "Despot: Online pomdp planning with regularization," in *Advances in Neural Information Processing Systems* (C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, eds.), vol. 26, Curran Associates, Inc., 2013.

[15] S. Ragi and E. K. Chong, "Uav path planning in a dynamic environment via partially observable markov decision process," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 4, pp. 2397–2412, 2013.

[16] S. Carpin, D. Burch, N. Basilico, T. Chung, and M. Kölsch, "Variable resolution search with quadrotors: Theory and practice," *Journal of Field Robotics*, vol. 30, 09 2013.

[17] S. Perez-Carabaza, J. Bermudez-Ortega, E. Besada-Portas, J. A. Lopez-Orozco, and J. M. de la Cruz, "A multi-uav minimum time search planner based on acor," in *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, (New York, NY, USA), p. 35–42, Association for Computing Machinery, 2017.

[18] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: an anytime algorithm for pomdps," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, (San Francisco, CA, USA), p. 1025–1030, Morgan Kaufmann Publishers Inc., 2003.

[19] T. Smith and R. Simmons, "Heuristic search value iteration for pomdps," 2012.

[20] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006* (J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, eds.), (Berlin, Heidelberg), pp. 282–293, Springer Berlin Heidelberg, 2006.

[21] S. Saisubramanian, K. H. Wray, L. Pineda, and S. Zilberstein, "Planning in stochastic environments with goal uncertainty," 2020.

[22] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," 2017.

[23] Epic Games, "Unreal engine." https://www.unrealengine.com/, 2023. Version 5.2.

[24] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," 2017.

[25] Open Robotics, "Ros 2 documentation." https://docs.ros.org/en/rolling/, 2024. Accessed: 2024/07/25.

[26] T. Cheng, L. Song, Y. Ge, W. Liu, X. Wang, and Y. Shan, "Yolo-world: Real-time open-vocabulary object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16901–16911, 2024.

[27] H. You, H. Zhang, Z. Gan, X. Du, B. Zhang, Z. Wang, L. Cao, S.-F. Chang, and Y. Yang, "Ferret: Refer and ground anything anywhere at any granularity," *arXiv preprint arXiv:2310.07704*, 2023.

[28] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[29] B. W. Kernighan and D. M. Ritchie, *The C programming language*. Englewood Cliffs, NJ: Prentice Hall, 2nd ed., 1988.

[30] B. Luo, Y. Zhang, A. Dubey, and A. Mukhopadhyay, "Act as you learn: Adaptive decision-making in non-stationary markov decision processes," in *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, (Richland, SC), p. 1301–1309, International Foundation for Autonomous Agents and Multiagent Systems, 2024.

[31] A. Pettet, Y. Zhang, B. Luo, K. Wray, H. Baier, A. Laszka, A. Dubey, and A. Mukhopadhyay, "Decision making in non-stationary environments with policy-augmented search," in *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, (Richland, SC), p. 2417–2419, International Foundation for Autonomous Agents and Multiagent Systems, 2024.