Noname manuscript No.
(will be inserted by the editor)

Game Tree Search for Minimizing Detectability and Maximizing Visibility

Zhongshun Zhang · Jonathon M. Smereka · Joseph Lee · Lifeng Zhou · Yoonchang Sung · Pratap Tokekar

Received: date / Accepted: date

Abstract We introduce and study the problem of planning a trajectory for an agent to carry out a scouting mission while avoiding being detected by an adversarial opponent. This introduces a multi-objective version of classical visibility-based target search and pursuit-evasion problem. In our formulation, the agent receives a positive reward for increasing its visibility (by exploring new regions) and a negative penalty every time it is detected by the opponent. The objective is to find a finite-horizon path for the agent that balances the trade off between maximizing visibility and minimizing detectability.

We model this problem as a discrete, sequential, two-player, zero-sum game. We use two types of game tree search algorithms to solve this problem: minimax search tree and Monte-Carlo search tree. Both search trees can yield the optimal policy but may require possibly exponential computational time and space. We first propose three pruning techniques to reduce the computational time while preserving optimality guarantees. When the agent and the opponent are located far from each other initially, we present a variable resolution technique with longer planning horizon to further reduce computational time. Simulation results show the effectiveness of the proposed strategies in terms of computational time.

Distribution A: Approved for public release; distribution unlimited. OPSEC # 3929

Zhongshun Zhang
Department of Computer Science, University of Maryland, College Park. E-mail: zszhang@umd.edu

Jonathon M. Smereka
U.S. Army CCDC Ground Vehicle Systems Center. E-mail: jonathon.m.smereka.civ@mail.mil

Joseph Lee
Aptiv. E-mail: joseph.lee@aptiv.com

Lifeng Zhou
Department of Electrical & Computer Engineering, Virginia Tech, Blacksburg. E-mail: lfzhou@vt.edu

Yoonchang Sung
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge. E-mail: yooncs8@csail.mit.edu

Pratap Tokekar
Department of Computer Science, University of Maryland, College Park. E-mail: tokekar@umd.edu

1 Introduction

Planning for visually covering an environment is a widely studied problem in robots with many real-world applications, such as environmental monitoring [1], precision farming [2], ship hull inspection [3], and adversarial multi-agent tracking [4, 5]. The goal is typically to find a path for an agent to maximize the area covered within a certain time budget or to minimize the time required to visually cover the entire environment. The latter is known as the Watchman Route Problem (WRP) [6] and is closely related to the Art Gallery Problem (AGP) [7]. The goal in AGP is to find the minimum number of cameras required to see all points in a polygonal environment. In this paper, we extend this class of visibility-based coverage problems to adversarial settings.

We consider scenarios where the environment also contains an opponent that is actively (and adversarially) searching for the agent (Figure 1). The agent, on the other hand, is tasked with covering the environment while avoiding detection by the opponent. This loosely models stealth reconnaissance missions where the agent is required to exercise caution while collecting

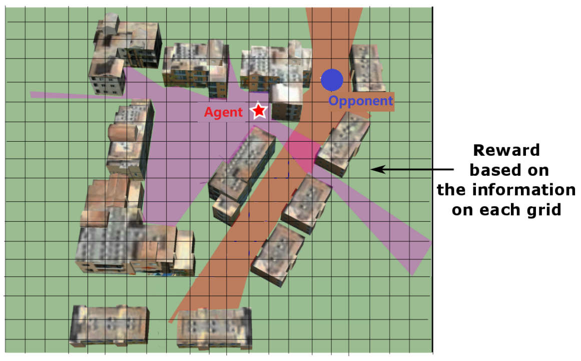


Fig. 1 The agent (the red star) aims to maximize the total area covered within the given time horizon while at the same time minimize the number of times the opponent (the blue dot) detects it.

information in the environment. We consider the version where there is a finite time horizon within which the agent must complete its mission. The objective of the agent is to maximize the total area covered within the given time horizon while at the same time minimizing the number of times the opponent detects it. In an exploration mission, the positive reward can be a function of the number of previously unseen cells visible from the current agent position (Figure 1). For ease of illustration, we assume that both sensing ranges are unlimited and obstacles in the environment can block lines-of-sight. The case of limited sensing range can be easily incorporated. The agent receives a negative reward when it is detected by the opponent (e.g., when it moves to a cell that lies within the opponent's visibility region).

We adopt a game-theoretic approach for this problem where the agent maximizes the total reward collected and the opponent minimizes that total reward. The total reward is a weighted combination of positive and negative rewards. The positive reward depends on the specific task at hand. For example, when the task is to scout an environment (Figure 1), the positive reward can be the total area that is scanned by the agent along its path. In this paper, we consider the case where the agent receives a fixed negative reward every time it is detected by the opponent. However, other models (e.g., time-varying negative rewards) can be easily incorporated. The total reward is a combination of the two reward functions.

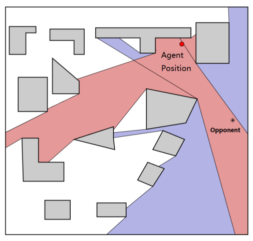
The proposed problem builds on classic pursuit evasion games [8–10] and visibility-based scouting problems [11,12]. In classic pursuit-evasion, the evader (*i.e.*, the agent in our setting) always tries to avoid the capture of the pursuer (*i.e.*, the opponent). In our setting, in addition to avoiding being detected by the opponent, the agent is tasked to explore the environment to maximize the total area covered. Thus, the definition of winning a game in our scenario is different. In classic pursuit-evasion games, the pursuer wins the game if the distance between the pursuer and the evader becomes less than a threshold [13] or if the evader is surrounded by the pursuer [14]. However, in our setting, winning the game depends on the information collected, such as the area explored, something that has not been considered in the conventional pursuit-evasion work. Also, by considering the opponent, this problem separates from the traditional exploration problems such as reconnaissance and surveillance where the goal is to maximize the information collected only.

There has been recent work on designing strategies for the visibility-based adversarial planning problem. Raboin et al. [10] introduced a heuristic search technique for solving pursuit-evasion games in partially observable Euclidean space. Another visibility-based pursuit-evasion problem formulated by Li et al. [15] is closely related to ours. Instead of relying on a regular discrete environment, the authors represented the game's state using visibility-based decomposition of the environment paired with a more classical grid-based decomposition. They also utilized minimax and MCTS to compute one player's optimal strategies. The main difference is that we consider an objective which is a combination of coverage and evasion, something that prior works have not addressed. We also exploit the properties of this problem to present ways to reduce the computational time in practice.

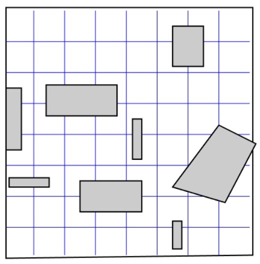
We abstract the underlying geometry and model the problem as a discrete, sequential, two-player, zero-sum game. Minimax tree search [16] and Monte-Carlo tree search (MCTS) [17] are well-known algorithms to solve discrete, two-player, zero-sum games. Both techniques build a search tree that contains all possible (or a subset of all possible) actions for both players over the planning horizon. The MCTS algorithm has been shown to converge to the optimal solution for turn-based [18] and simultaneous-move games [19].

To reduce the computation time of minimax tree search and MCTS, we first propose several pruning techniques (Theorems 1-2) based on the structural properties of the underlying problem. We show the resulting pruned tree still preserves optimality. To further reduce the computational time, we then introduce a *changing resolution* strategy that allows the agent to change the spatial horizon to build a search tree with fewer levels. We show that the resulting strategy outperforms the fixed resolution one, especially in larger environments.

To summarize, the contributions of this effort are as follows: (1) We formulate a problem of minimizing detectability and maximizing visibility as a sequential,



(a) The case when the agent is detected by the opponent.



(b) The agent and the opponent move in a grid-based environment.

Fig. 2 A negative penalty will be added if the agent is inside the opponent's visibility polygon (*i.e.*, the blue region). In a reconnaissance mission, the area of the agent's visibility polygon (*i.e.*, the red region) is considered as a positive reward. Both the agent and the opponent move in the grid-based environment, as in (b).

two-player, zero-sum game between an agent and an opponent; (2) We propose three pruning strategies that preserve optimality; and (3) a *changing resolution* strategy, which can be applied for both the minimax tree search and MCTS to reduce computational costs.

A preliminary version of the paper was presented at ICRA 2019 [20] without the *changing resolution* strategy, without detailed descriptions of the minmax tree search and MCTS algorithms, and provide online planning results in richer environments including Gazebo simulations.

The rest of the paper is organized as follows. We begin by describing the problem setup in Section 2. We then describe two tree search techniques in Section 3 and present two approaches for improving the computational efficiency of these tree search techniques in Section 4. Next, we evaluate the effectiveness of the proposed approaches through extensive simulations in Section 5. In the end, we summarize the paper and outline some future work in Section 6.

2 Problem Formulation

We consider a grid-based environment where each cell in the environment is associated with a positive reward. Our problem is formulated by appropriately designing the reward function — the agent obtains positive rewards for maximizing visibility (depending on the type of mission) and receives negative rewards when detected by the opponent. The reward is used to measure both the visibility of an agent and the detectability by an opponent.

We make the following assumptions: (1) The agent and the opponent move in the same grid-based map and can move one grid in one time step. (2) Both the agent and the opponent know the full grid-based map *a priori*. (3) We assume that the agent and the opponent have known sensing ranges (not necessarily the same). In this paper, we assume that both sensing ranges are unlimited for ease of illustration. However, the case of limited sensing range can be easily incorporated. (4) The opponent has a sensor that can detect the agent when the agent is within its visibility region. (5) There is no motion uncertainty associated with the agent and opponent. (6) The agent is aware of the position of the opponent. These assumptions are applicable in scenarios where we expect the agent's actions to be conservative, taking into account an "intelligent" opponent that always chooses the best move.

Even though the last assumption may seem restrictive, there are some practical scenarios where it is justified. For example, Bhadauria and Isler [21] describe a visibility-based pursuit-evasion game where police helicopters can always provide the global positions of the evader to the pursuer that is moving on the ground and may not be able to directly see the pursuer. Thus, even if the opponent is not in the field-of-view of the agent, the agent may still know the position of the opponent by communicating with other (aerial) agents. Note that the agent still does not know where the opponent will move next, thereby, making the problem challenging.

In general, the environment could be any discrete environment, not just a grid-based environment, as long as it satisfies the above requirements. Continuous environments can be appropriately discretized such that they satisfy the above assumptions. Commonly used techniques for environment discretization include graph representation [22], occupancy maps [23], and randomized methods such as probabilistic roadmaps [24], and Rapidly-exploring Random Trees [25, 26].

The complexity of the tree search algorithm will depend on the number of vertices (or grid cells) in a given discretization. In Section 4, we present two ways to improve efficiency. First, we show how to prune away nodes and branches in the tree while preserving optimality. Second, we show how to change the spatial resolution of the tree (Section 4.3) at different levels for improving the search, especially in large environments. By losing some precision, the tree can predict further ahead, leading to better plans without incurring additional computation cost. However, this method will inevitably lose some accuracy. We show that reducing the resolution is beneficial in net, through experiments over a larger map.

We next describe the main problem to be solved in the paper. Consider that the agent receives the positive reward when exploring new area and penalties when detected by the opponent. The agent's objective can be written as:

$$\max_{\pi_a(t)} \min_{\pi_g(t)} \{R(\pi_a(t)) - \eta(\pi_a(t), \pi_g(t))P\} . \quad (1)$$

While the objective of the opponent is as follows:

$$\min_{\pi_g(t)} \max_{\pi_a(t)} \{R(\pi_a(t)) - \eta(\pi_a(t), \pi_g(t))P\} , \quad (2)$$

where $\pi_a(t)$ denotes an agent's path from time step 0 to t. $\pi_g(t)$ denotes an opponent's path from time step 0 to t. $R(\pi_a(t))$ denotes the positive reward collected by the agent along the path from time step 0 to t. P is a constant which gives the negative reward for the agent whenever it is detected by the opponent. $\eta(\pi_a(t), \pi_g(t))$ indicates the total number of times that the agent is detected from time step 0 to t. For the rest of the paper, we model $R(\pi_a(t))$ to be the total area that is visible from the agent's path $\pi_a(t)$.

We model this problem as a discrete, sequential, two-player zero-sum game between the opponent and the agent. In the next section, we demonstrate how to find the optimal strategy for this game and explain our proposed pruning methods.

3 Tree Search Techniques

We abstract the underlying geometry and model the problem as a discrete, sequential, two-player, zero-sum game. Minimax tree search [16] and MCTS [17] are two well-known algorithms to solve discrete, two-player, zero-sum games. Both techniques build a search tree that contains all possible (or a subset of all possible) actions for both players over planning horizons. In general, the size of search trees is exponential in planning horizon. Pruning techniques, such as alpha-beta pruning [27], can be employed to prune away branches that are guaranteed not to be part of the optimal policy.

We refer to the agent and the opponent as MAX and MIN players, respectively. Even though the agent and the opponent move simultaneously, we can model this problem as a turn-based game. At each time step, the agent moves first to maximize the total reward, and then the opponent moves to minimize the total reward. This repeats for a total of T planning steps. In this section, we first show how to build a minimax search tree to find the optimal policy. Then, we show how to construct a Monte-Carlo search tree to solve the same problem. The advantage of MCTS is that it finds the optimal policy in less computational time than minimax tree — a finding we corroborate in Section 5.

3.1 Minimax Tree Search

A minimax tree search is a commonly used technique for solving two-player zero-sum games [27]. Each node stores the position of the agent, the position of the opponent, the polygon that is visible to the agent along the path from the root node till the current node, and the number of times the opponent detects the agent along the path from the root node to the current node. The tree consists of the following types of nodes:

– *Root node*: The root node contains the initial positions of the agent and the opponent.
– *MAX level*: The MAX (i.e., agent) level expands the tree by creating a new branch for each neighbor of the agent's position in its parent node from the previous level (which can be either the root node or a MIN level node). The agent's position and its visibility region are updated at each level. The opponent's position and the number of times the agent is detected are not updated at this level.
– *MIN level*: The MIN (i.e., opponent) level expands the tree by creating a new branch for each neighbor of the opponent's position in its parent node (which is always a MAX level node). The opponent's position is updated at each level. The total reward is recalculated at this level based on the agent's and opponent's current visibility polygons and the total number of times the agent is detected up to the current level.
– *Terminal node*: The terminal node is always a MIN level node. When the minimax tree is fully generated (*i.e.*, the agent reaches a finite planning horizon), the reward value of the terminal node can be computed.

The reward values are backpropagated from the terminal node to the root node. For each node, the minimax policy chooses an action that maximizes (MAX level) or minimizes (MIN level) the backpropagated reward.

Figure 3 illustrates the steps to build a minimax tree that yields an optimal strategy by enumerating all possible actions for both the agent and the opponent. Algorithm 1 presents the algorithm of minimax tree search.

3.2 Monte-Carlo Tree Search

In the naive minimax tree search, the tree is expanded by considering all the neighbors of a leaf node, one-by-one. In MCTS, the tree is expanded by carefully selecting one of the nodes to expand. The node to select for expansion depends on the current estimate of the value of the node. The value is found by simulating many

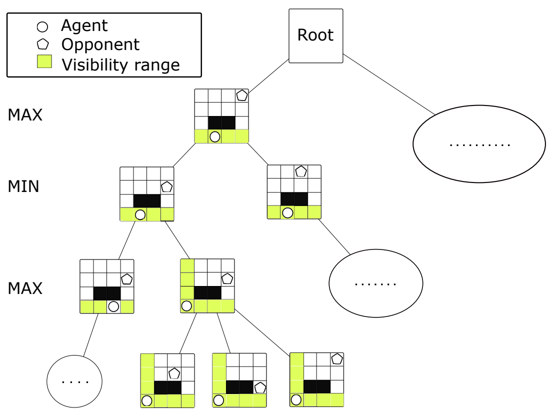


Fig. 3 A (partial) minimax search tree. The root node contains the initial states of the agent and the opponent. Two successive levels of the tree correspond to one time step. The agent moves first to an available position in order to maximize the reward (MAX level). The opponent moves subsequently to a neighboring cell to minimize the agent's reward (MIN level).

rollouts. In each rollout, we simulate one instance of the game, starting from the selected node, by applying some arbitrary policy for the agent and the opponent until the end of the planning horizon, T. The total reward collected is stored at the corresponding node. This reward is then used to determine how likely is the node to be chosen for expansion in future iterations. Algorithm 2 presents the algorithm of MCTS.

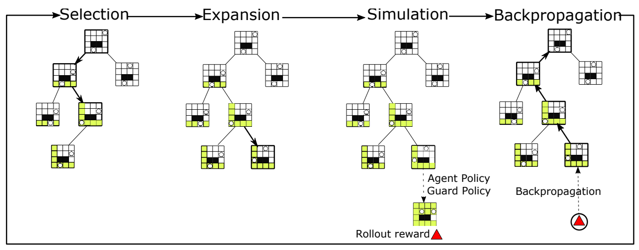


Fig. 4 Four iteration steps in Monte-Carlo search tree.

Each node in the Monte-Carlo search tree stores the total reward value, and the number of times the node is visited. Each iteration of MCTS consists of the following four steps [28] (Figure 4):

- **Selection** (Line 4 in Algorithm 2, pseudocode presented in Algorithm 3): Starting from the root node (in every iteration), the node selection algorithm uses the current reward value to recursively descend through the tree until we reach a node that is not at the terminal level (*i.e.*, corresponding to time T) and has children that have never been visited before. We use the Upper Confidence Bound for Trees (UCT) [18] to determine which node should be selected. The UCT value takes into account not only the average of the rollout reward obtained but also the number of times the node has been visited. If a node is not visited often, then the second term in the UCT value will be high, improving its likelihood of getting selected. At the agent level, we choose the node with the highest UCT value while at the opponent level with the lowest UCT value. Note that $n(v)$ stands for the number of simulations for the node v, and N stands for the total number of MCTS simulations.

Algorithm 1: The Minimax search with Pruning.

```
function Minimax($node, depth, \alpha, \beta, state$)
    if $node$ is a terminal node then
        return $value$
    else if $state$ is at the agent level then
        for each child $v$ of $node$ do
            $V \leftarrow$ Minimax($v, depth - 1, \alpha, \beta$, MIN)
            $bestvalue \leftarrow \max(bestvalue, V)$
            $\alpha \leftarrow \max(bestvalue, \alpha)$
            // Alpha-beta pruning
            if $\beta \leq \alpha$ then
                break
            end
            // Proposed condition
            if pruning condition is true then
                break
            end
            return $value$
        end
    else
        for each child $v$ of $node$ do
            $V \leftarrow$ Minimax($v, depth - 1, \alpha, \beta$, MAX)
            $bestvalue \leftarrow \min(bestvalue, V)$
            $\beta \leftarrow \min(bestvalue, \beta)$
            if $\beta \leq \alpha$ then
                break
            end
            if pruning condition is true then
                break
            end
            return $value$
        end
    end
    Initial $\leftarrow \{S_0\}$,Map
    $A_r(s), A_t(s) \leftarrow$ Minimax($S_0, 1, -\infty, \infty$, MAX)
end
```

- **Expansion** (Lines 6-9 in Algorithm 2): Child nodes (one or more) are added to the selected nodes to expand the tree. If the child node is at the agent level, the node denotes one of the available actions for the agent. If the child node is at the opponent level, the node denotes one of the available actions for the opponent. Expansion details are given in Algorithm 2.
- **Rollout** (Line 11 in Algorithm 2, pseudocode presented in Algorithm 4): A Monte-Carlo simulation is

carried out from the expanded node for the remaining planning horizon. The agent and the opponent follow a random policy uniformly. Based on this, the total reward for this simulation is calculated. Rollout details are given in Algorithm 4.
- **Backpropagation**(Line 13-17, Algorithm 2): The total reward found is then used to update the reward value stored at each of the predecessor nodes.

Algorithm 2: Monte-Carlo Tree Search

```
function MCTS(Tree, Initial agent and opponent state)
    Create root node $v_0$ with initial opponent and agent state $s_0$;
    while maximum number of iterations not reached do
        // Selection
        $v_i \leftarrow$ Monte_Carlo_Selection(Tree, $v_0$)
        // Expand or rollout
        if level($v_i$) $= T$ and $n(v_i) = 0$ then
            // Expand
            Tree $\leftarrow$ Expand(Tree,$v_i$)
            if Newly added node can be pruned then
                break
            end
        else
            // Rollout
            $R \leftarrow$ Rollout($v_i$);
        end
        // Backpropagation
        while $v_i \neq$ NULL do
            // Update total reward value
            $Q(v_i) \leftarrow Q(v_i) + R$
            $n(v_i) \leftarrow n(v_i) + 1$
            $v_i \leftarrow$ parent of $v_i$
        end
        $N \leftarrow N + 1$
    end
    return Tree
end
```

Algorithm 3: MCTS selection

```
function Monte_Carlo_Selection(Tree, $v_i$)
    while level($v_i$) $\neq$ TERMINAL do
        if level($v_i$) = AGENT then
            $v_i \leftarrow \arg\max_{v' \in \text{children}(v_i)} \frac{Q(v')}{n(v')} + c\sqrt{\frac{2 \ln N}{n(v')}}$
        else
            $v_i \leftarrow \arg\min_{v' \in \text{children}(v_i)} \frac{Q(v')}{n(v')} - c\sqrt{\frac{2 \ln N}{n(v')}}$
        end
    end
end
```

Given a sufficient number of iterations, the MCTS with UCT is guaranteed to converge to the optimal policy [19,29]. However, if the agent has n available actions, in the worst case, we need n^{k-1} in k-th level of the search tree to enumerate all the possible nodes. This may still require building an exponentially sized tree. In the next section, we present a number of pruning conditions to reduce the size of the tree, and strategies to expand the search tree with changing resolution to save the computation time.

Algorithm 4: MCTS rollout

```
function Rollout($v$)
    $R \leftarrow 0$
    while level($v$) $\neq 2T + 1$ do
        if level($v$) = AGENT then
            $v \leftarrow$ choose an agent action at random
        else
            $v \leftarrow$ choose an opponent action at random
            $R \leftarrow$ update reward
        end
        return $R$
    end
end
```

3.3 Online Planning with Search Tree

Once the tree is built, the agent can execute the policy for one step. If we are using minimax tree search, at the root node, the agent executes the first action along the optimal path found. In MCTS, the agent executes the first action along the path with the best average reward in the rollout simulations. After the agent executes one step and observes the new position, the agent will update the position of the opponent (based on new measurement or estimation) in the new root node and rebuild the search tree.

4 Improved Computational Efficiency

In a larger environment, the agent may need to build a search tree that reaches far enough from its initial position to yield a good strategy. This is especially the case when the starting positions of the agent and the opponent are far from each other. However, when the size of the tree increases, the computational time required to generate the tree grows exponentially in the worst case (despite pruning). In this section, we present the following two strategies to reduce the computational cost: (1) Pruning strategies to reduce the size of the tree; and (2) Expanding the spatial reach of the search tree with changing resolution at different levels.

4.1 Pruning Techniques

In this section, we present several pruning techniques to reduce the size of the tree and the computational time required to build the minimax tree and the MCTS. Pruning a node implies that the node will never be expanded (in both types of trees). In MCTS, if a node is pruned we simply will break to the next iteration of the search. Pruning the tree results in considerable computational savings which we quantify in Section 5.

In the case of the minimax search tree, we can apply a classical pruning strategy called *alpha-beta pruning* [17]. Alpha-beta pruning maintains the minimax values at each node by exploring the tree in a depth-first fashion. It then prunes nodes if a node is clearly dominated by another, see [17] for more details. Alpha-beta pruning is preferable when the tree is built in a depth first fashion. However, we can exploit structural properties of this problem to further prune away nodes without needing to explore a subtree fully. We propose strategies that find and prune redundant nodes before the terminal level is reached.

Our proposed pruning techniques apply for both types of trees. Therefore, in the following we refer to a "search tree" instead of specifying whether it is minimax or MCTS.

Our first proposed class of pruning techniques (Theorem 1) are based on the properties of the given map. Consider the MIN level and the MAX level separately. The main idea of these pruning strategies is to compare two nodes A and B at the same level of the tree, say the MAX level. In the worst case, the node A would obtain no future positive reward while always being detected at each time step of the rest of the horizon (e.g., when the agent moves from behind an obstruction into an open area into the view of the opponent, and thus it is no longer able to collect a reward from proceeding on that path). Likewise, in the best case, the node B would collect all the remaining positive reward and never be detected in the future. If the worst-case outcome for node A is still better than the best-case outcome for node B, then node B will never be a part of the optimal path. It can thus be pruned away from the search tree. Consequently, we can save time that would be otherwise spent on computing all of its successors. Note that these conditions can be checked even before reaching the terminal node of the subtrees at A or B.

Given a node in the search tree, we denote the remaining positive reward (unscanned region) for this node by $F(\cdot)$. Note that we do not need to know $F(\cdot)$ exactly. Instead, we just need an upper bound on $F(\cdot)$. This can be easily computed since we know the entire map information *a priori*. The total reward collected by node A and by node B from time step 0 to t are denoted by $R^A(t)$ and $R^B(t)$, respectively.

Theorem 1 *Given a time horizon T, let A and B be two nodes in the same level of the search tree at time step t.*

In the MAX level, if $R^A(t) - (T-t)\eta \geq R^B(t) + F(B)$, then the node B can be pruned without loss of optimality.

Similarily, in the MIN level, if $R^A(t) + F(A) \leq R^B(t) - (T-t)\eta$, then the node B can be pruned without loss of optimality.

Proof We prove the case at the MAX level. The proof of the MIN level case is similar.

In the case of node A, the worst case occurs when in the following $T-t$ steps the agent is always detected at every remaining step and collects zero additional positive rewards. After reaching the terminal tree level, the reward backpropagated to node A will be $R^A(t) - (T-t)\eta$. For node B, the best case occurs in the following $T-t$ steps when the agent is never detected but obtains all remaining positive rewards. In the terminal tree level, the node B collects the reward of $R^B(t) + F(B)$.

Since $R^A(t) - (T-t)\eta \geq R^B(t) + F(B)$ and both nodes are at the MAX level, it implies that the reward returned to the node A is always greater than that returned to the node B. Therefore, the node B will not be a part of the optimal policy and can be pruned without affecting the optimality.

Now, we present the second pruning strategy. The main idea of the second type of pruning strategy (*i.e.*, Theorem 2) comes from the past path (or history). If two different nodes have the same agent and opponent position but one node has a better history than the other, then the other node can be pruned away.

Here, we denote by $S^A(\pi(t))$ and $S^B(\pi(t))$ the total scanned region in the node A and the node B from time step 0 to t, respectively.

Theorem 2 *Given a time horizon T and $0 < t_1 \leq t_2 \leq T$, let the node A be at the level t_1 and the node B be at the level t_2, such that both nodes are at a MAX level. If (1) the agent and the opponent's position stored in the nodes A and B are the same, (2) $S^A(\pi(t_1)) \supset S^B(\pi(t_2))$, and (3) $R^A(t) > R^B(t) + (t_2 - t_1)\eta$, then the node B can be pruned without loss of optimality.*

Proof With $0 < t_1 \leq t_2 \leq T$, we have the node B appear further down the tree as compared to the node A. $S^A(\pi(t_1)) \subseteq S^B(\pi(t_2))$ indicates that the node A's scanned area is a subset of the node B's scanned area.

Since the nodes A and B contain the same opponent and agent positions, one of the successors of node A contains the same opponent and agent positions as node B. Since $R^A(T) \geq R^B(T) + (t_2 - t_1)\eta$ and $S^A(\pi(t_1)) \supset S^B(\pi(t_2))$, the value backpropagated from the successor of node A will always be greater than the value backpropagated from the path of node B. Furthermore, more reward can possibly be collected by node A since $S^A(\pi(t_1)) \subseteq S^B(\pi(t_2))$. Thus, the node B will never be a part of the optimal path and can then be pruned away.

4.2 Bounding the Size of the Tree

We analyze the computational cost by bounding the number of nodes generated by the minimax search tree to find the optimal path. For the minimax search tree, we present the approximate computational cost by giving the size of the tree. Clearly, the tree's size is not the only factor determining the complexity. In most cases, the bottleneck is the tree's size, and therefore, the complexity will mainly come from the size of the tree. For MCTS, there is not clear way to determine the effect of pruning analytically. Instead, we present numerical results by comparing the time required to find the optimal solution with/without pruning, in the evaluation section. We present bounds on the size of the minimax search tree in the following.

Consider that the planning horizon is T steps, the height of the minimax search tree is $2T$, the agent has a available actions at each step, the opponent has b actions at each step, and there are $\mathcal{K}$ grid points/cells in the given environment. When a minimax search tree is generated using brute-force, the number of nodes in the full tree is $O((ab)^T)$. In the best case with alpha-beta pruning (which means the best moves are always searched first while we build the tree), the number of nodes in the tree is $\Theta((ab)^{\frac{T}{2}})$ [30].

With pruning techniques proposed in Theorem 1 and Theorem 2, we consider the best-case scenario similar to the alpha-beta pruning result above. The best-case indicates that for all nodes in the same level of the search tree, the more informative[1] nodes are always searched first. In Theorem 2, one requirement is that the agent and the opponent's positions stored in two nodes to compare are identical. If the best nodes in each position are all generated first, then other nodes at the same level containing the same agent and opponent's positions can all be punned away. In an environment with $\mathcal{K}$ grid points/cells, there are $\mathcal{K}^2$ possible combinations of the agent and the opponent's positions. Thus, at most $\mathcal{K}^2$ nodes are listed at each level of the search tree in the best case. The size of the tree is lower bounded by $\Omega(\mathcal{K}^2 \cdot 2T)$. In the trivial case where $a, b = O(\mathcal{K})$, we see that the best case is realized. Therefore, in the best case the size of the tree with pruning will be $\Theta(\mathcal{K}^2 \cdot 2T)$.

In the worst case, the less informative nodes are always selected first while building the search tree in a depth-first fashion. Both alpha-beta pruning and our punning techniques cannot prune any nodes, so the size of the tree is the same as the brute-force.

In practice, the size of the tree will be in between the best and worse-case. We show the empirical results in Section 5.

[1] Here, more informative indicates that the value backpropagated from the current node's successor will be greater than the value backpropagated from the path of another node that contains the same agent and opponent's position.

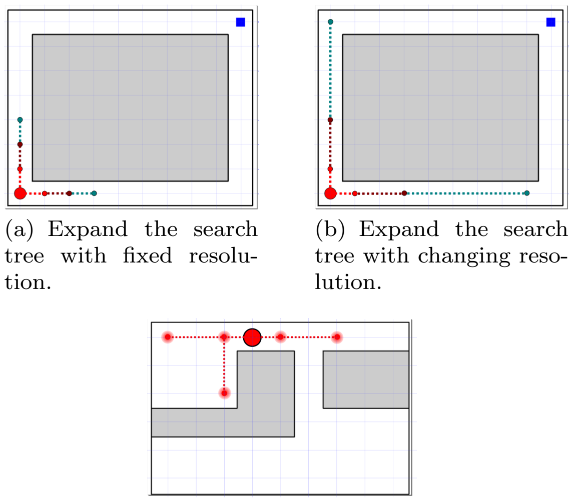
4.3 Expanding the Tree with Changing Resolution

Consider a scenario where the agent and opponent are located far from each other in a large environment. In such a case, even if the agent builds a search tree with many levels, the leaf nodes in the tree may still not go far enough to see the opponent (Figure 5). Instead, we propose a technique that changes the spatial resolution at different levels of the tree. We define the resolution as follows: Consider a search tree $\mathcal{T}$ and a node A at level k. The resolution $C(k)$ of node A is defined as the distance that will be traveled by the agent and opponent atomically when executing any action corresponding to A's child nodes.

Traditionally, we fix the resolution for all levels as one, e.g., $C(k) = 1$, as shown in Figure 5 (a). All the nodes expand with the same resolution. The agent (red square) looks ahead for only three steps in this 8×7 environment. The agent at least needs to plan for seven steps to discover the opponent (blue square) located in the top right corner.

In contrast, we apply the changing resolution approach, as shown in Figure 5 (b). In the k-th level of the search tree, the newly generate node in $(k+1)$-th level will expand by combining $C(k)$ grids into one "larger grid". $C(k)$ is defined as $C(k) = 2^{k-1}$. Thus, in the root node, $C(1) = 1$ will not reduce the accuracy and will return one of the nodes as the control action. As k grows, we sacrifice some accuracy by changing the resolution of the gird map but the agent can look ahead further.

Reducing the resolution of the map will inevitably leading to losing some accuracy in the plans (as well as in the representation of the map). However, the tree can look ahead a longer spatial horizon without additional



(a) Expand the search tree with fixed resolution.

(b) Expand the search tree with changing resolution.

(c) Disadvantage: Expand the search tree with changing resolution will lose some accuracy. The agent could miss the corridor and turn left by mistakenly thinking the environment is larger on the left.

Fig. 5 Two different ways to expand the search tree. The three different colors stand for the resolution of each step in different levels of the search tree.

computational cost. In Figure 5 (c), we show an example that increasing the resolution makes the agent miss the small corridor, which could have lead the agent to a larger, unscanned environment. However, our empirical results show that this does not happen often and the benefits of looking ahead outweigh this potential disadvantage.

In general, at the beginning of building the search tree, we do not need to reduce the resolution since the agent will execute one of the actions in the first level of the tree. After the search tree expands for a few generations, the accuracy of the map is not as important as the initial steps. The intuition behind the changing resolution strategy is when the precision becomes less critical, combining several grids into one can help the agent to plan in a longer horizon and decide which direction leads to better results in the distant future. Also, the computational cost does not increase since the depth of the search tree will remain the same. Finally, we also investigate the question of which $C(k)$ function to use to change the resolution.

Without changing the resolution, the agent can predict the effect of positions that are T steps away, which is the same as the search tree's depth if all control actions are unit length. With changing resolution, the search tree can reach farther away positions, with the same computational cost. For example, if the path is planned by a linearly changing resolution $C(k) = k$ in the search tree, we can reach agent positions that are $\frac{1}{2}T(T-1)$ away.

In the simulation, we show that although we cannot guarantee optimality, the empirical performance of the agent in most cases is better with this approach. This turns out to be the case especially when the environment is large, or when the agent and the opponent are located far from each other. By looking further ahead, the agent can make a better decision either to collect more rewards or to move away from the opponent.

5 Evaluation

In this section, we evaluate the proposed techniques in the context of a reconnaissance mission. We assume the visibility range of the agent and the opponent are both unlimited (only restricted by the obstacles in the environment). The experiments were conducted on a 2.90GHz i9-8950HK processor with 32 GB RAM. The software was written in MATLAB R2017a and used the VisiLibity library [31] to compute the visibility polygons.

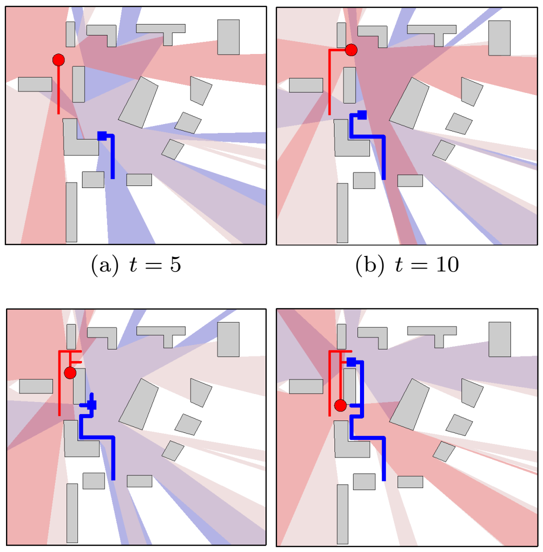
First, we present two qualitative examples that show the path found by the minimax algorithm. Second, we compare the computational cost of the two search tree algorithms with and without pruning. Third, we study the trade-off between solution quality and computational time by changing the resolution in the search process.

5.1 Varying Penalty

Both the minimax tree search and MCTS can find the same optimal solution for these instances. Figures 6 and 7 show two examples of the policy found by MCTS method, using high and low negative penalty values (P in Equation 1), respectively. We use a 25×25 grid environment. With higher negative reward $P = 30$, the agent tends to prefer avoiding detection by the opponent (Figure 6). With a lower negative reward $P = 3$, the agent prefers to explore more area (Figure 7).

Both tree search methods give the same optimal solution in both cases. (In general, there can be multiple optimal solutions. There could be multiple paths to collect the same reward in the same initial position, and the solution is not unique in most cases.) We can see the algorithm can help the agent to decide whether to detect more area or to avoid the detection of the opponent based on the penalty.

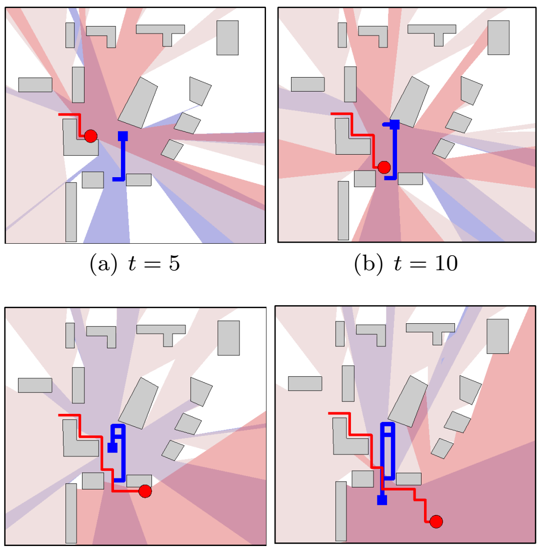
The MCTS finds the optimal solution (for $T = 10$) in 40,000 iterations taking a total of approximately 50 minutes. On the other hand, the minimax tree search



(a) $t = 5$ (b) $t = 10$

(c) $t = 15$ (d) $t = 20$

Fig. 6 Qualitative example (higher penalty $P = 30$): Path for the agent (red) and the opponent (blue) is given by MCTS for $T = 10$. The environment is a 25×25 grid. With a higher penalty, the agent prefers paths where it can hide from the opponent at the expense of the area explored (from (a) to (d), $t = 5, 10, 15, 20$.). Figure 7 shows the case with a lower penalty.



(a) $t = 5$ (b) $t = 10$

(c) $t = 15$ (d) $t = 20$

Fig. 7 Qualitative example (lower penalty $P = 3$): With a lower penalty, path for the agent (red) and the opponent (blue) is given by MCTS. The agent prefers paths where it increases the area explored at the expense of being detected often. From (a) to (d), $t = 5, 10, 15, 20$.

required approximately 10 hours to find the optimal solution. More thorough comparison is in the next subsection.

5.2 Pruning Techniques

MCTS: We evaluate the computational time required to find the optimal solution by varying the time horizon T. Figure 8 shows the computational time for the two search algorithms. The time horizon T ranges from 1 to 5; the tree consists of 3 to 11 levels. When the time horizon T is less than 3, the minimax search tree performs better than MCTS. This can be attributed to the fact that Monte-Carlo search requires a certain minimum number of iterations for the estimated total reward value to converge to the actual one. When the horizon T is increased, the MCTS finds the solution faster since it does not typically require generating a full search tree. We only compare up to $T = 5$ since beyond this value, we expect MCTS to be much faster than minimax. Furthermore, the computational time required for finding the optimal solution for the minimax tree beyond $T = 5$ is prohibitively large.

Figure 8, as expected, shows that the computational time with pruning is lower than that without pruning for both techniques. Next, we study this effect in more detail.

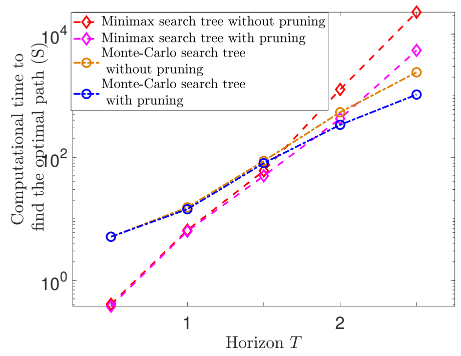


Fig. 8 Comparison of the time required to find the optimal solution with the minimax tree and the MCTS, with and without pruning. Note that the y axis is in log scale.

Minimax Tree Search: We show the effectiveness of the pruning algorithm by comparing the number of nodes generated by the brute force technique (no pruning) with the minimax tree with pruning. We generate the initial position of the agent and the opponent randomly. We find the optimal path for various horizons ranging from $T = 2$ to $T = 7$. Therefore, the minimax tree depth ranges from 5 to 15 (if the planning horizon is T, then we need a game search tree with $2T + 1$ level).

The efficiency of the proposed pruning algorithm is presented in Table 1, which shows the individual effect of alpha-beta pruning and the combined effect of all pruning techniques.

Since the efficiency of pruning is highly dependent on the order in which the neighboring nodes are added

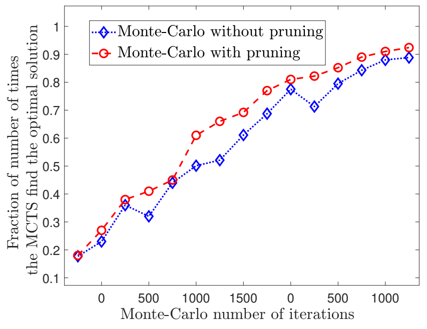


Fig. 9 Effect of increasing the number of iterations in MCTS, with and without pruning, on the the likelihood of finding the optimal solution. The y–axis shows the fraction of the number of trials (out of 50 trials) MCTS was able to find the optimal solution given by the minimax tree for $T = 3$.

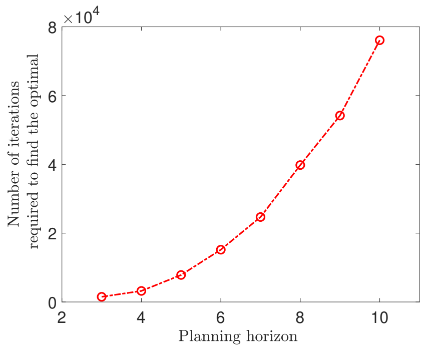


Fig. 10 Effect of the planning horizon on the number of iterations required to find the optimal solution for MCTS with pruning.

to the tree first, different results can be achieved by changing the order in which the children nodes are added to the minimax tree. Table 1 compares the number of nodes generated. The table shows the effect of individual pruning techniques. By applying the pruning algorithm, the best case only generates 2.94×10^4 nodes to find the optimal solution, while brute force takes 9.76×10^6 nodes to find the same solution.

Figure 9 shows the fraction of the times we find the optimal solution as a function of the number of iterations when $T = 3$ in a 10×10 grid map. We first find the optimal solution using a minimax tree. Then, we run the MCTS for a fixed number of iterations and verify if the best solution found has the same value as the optimal. The x-axis in this figure is the number of iterations in MCTS. Note that since there is more than one optimal solution, we check the accumulated collected reward instead of how the agent moves in each step.

We make the following observations from Figure 9: (1) The proposed pruning strategy increases the (empirical) likelihood of finding the optimal solution in the same number of iterations; and (2) The probability of finding the optimal solution grows as the number of iterations grows.

The number of iterations required to find the optimal solution also depends on the planning horizon. Figure 10 shows the effect of the planning horizon over the number of iterations required to find the optimal solution. Note that even though the likelihood of finding an optimal solution increases with more iteration times in general, it is always possible that only a suboptimal is found due to "overfitting" caused by the UCT selection rule. Therefore, we run the MCTS multiple times and find out how often we find the same total reward within a given number of iterations. If we find the optimal solution 80% or more times, we consider it as success. We find that the number of iterations required to find success 80% or more times increases exponentially as we vary the planning horizon.

5.3 Changing Resolution Approach

In this section, we evaluate the effectiveness of the changing resolution strategy in the minimax search and MCTS. First, we present a qualitative example to show some limitations of the baseline fixed resolution approach and how they are overcome with the changing resolution strategy (using $C(k) = k$).

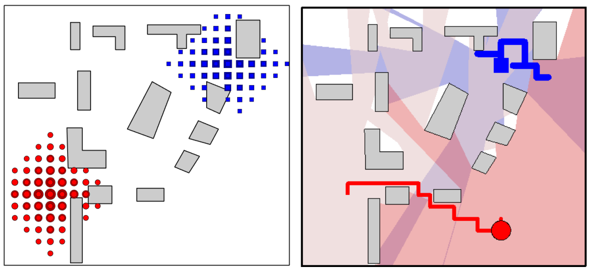
In Figure 11, we present a qualitative example of using the Monte-Carlo search tree with/without changing the resolution (planning horizon is five steps) for the agent and the opponent. It would be more direct if we look at the results from the opponent's perspective, as shown in Figure 11. Figure 11(a) shows the limitation of the traditional approach that without changing resolution, within five steps, the opponent (blue square) is not able to discover the agent. From the results of the search tree, the opponent's move cannot affect the agent since the agent cannot be detected in the Monte-Carlo simulations within five steps. As a result, the opponent ends up moving back-and-forth locally (because the opponent cannot discover the agent during the roll-out, it cannot find an optimal solution). In Figure 11(b), we linearly decrease the resolution in the search tree ($C(k) = k$). As a result, the new search tree with changing resolution can look ahead a length of 15 units away without any additional computation cost.

The right figure shows the online path for 20 steps. With a changing resolution in the search process, Figure 11(b) gives a more reliable predicted path for the opponent. Also, the agent first explores part of the environment then goes back to hide the opponent.

In Figure 13, we compare the collected reward between fixed resolution with changing resolution. We test the results in different simulation environments that are

Table 1 Comparision of the number of nodes generated by different pruning techniques, from $T = 3$ to $T = 6$.

		Number of nodes generated			
Planning horizon		$T = 3$	$T = 4$	$T = 5$	$T = 6$
Brute force		625	1.56×10^4	3.90×10^5	9.76×10^6
With only alpha-beta	Maximum	403	3844	7.08×10^4	1.70×10^6
	Median	206	2822	1.80×10^4	2.46×10^5
	Minimum	104	1444	7860	1.86×10^5
With all pruning techniques	Maximum	388	1389	3.3×10^4	4.81×10^5
	Median	105	639	4064	3.74×10^4
	Minimum	78	563	3016	2.94×10^4


(a) Search tree with fixed resolution. The search depth in the tree is not enough for the opponent to locate the agent. The Monte-Carlo search tree returns a result that the opponent only moves locally, and the agent explores the environment by ignoring some potential "danger".

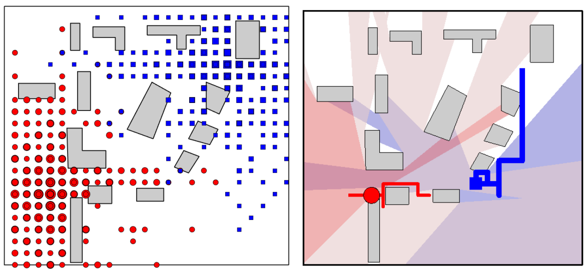
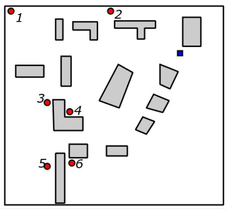
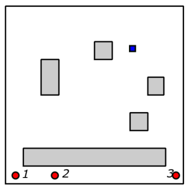
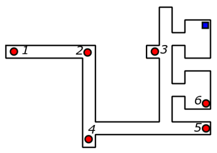

(b) Search tree with increasing resolution. The search depth in the tree remains the same. The Monte-Carlo search tree returns a simulation result that the opponent can move closer to the agent, and the agent avoids the opponent, even when they are far away initially.

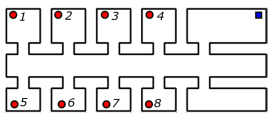
Fig. 11 Qualitative examples. The effect of using changing resolution in the Monte-Carlo search tree. The left figure shows the structure in the search tree in the initial position, lists all the locations included in the five depth of the search tree. The right figure shows the online planning path for 20 steps. Opponent will move back and forth if it cannot detect the agent in planning horizon.

shown in Figure 12. We compute the difference between the average reward collected various initial positions for the agent (marked as red dots in Figure 12). The path of the opponent is planned by a linearly changing resolution $C(k) = k$ in the search tree. In all the experiments, the opponent is planning with the changing resolution approach to ensure the opponent can locate the agent even if they are far apart.


(a) Environment A (25 × 25).


(b) Environment B (10 × 10).


(c) Environment C (13 × 16).


(d) Environment D (25 × 10).

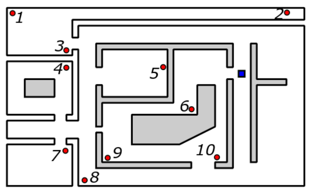

(e) Environment E (50 × 30).

Fig. 12 Environments used for the online simulations. Red dots are the different initial positions for the agent. The starting position for the opponent is fixed at the blue dot.

From Figure 13, we can see on average, applying the changing resolution approach will produce a better path for the agent. This is especially the case when the environment is large, or when the agent and opponent are located far from each other, such as in the 50 × 30 environment E. Also, as expected, there are cases where fixed resolution collects more or the same reward, such as position 3 in environment C, positions 1, 2, and 3 in environment B. This is due to the fact that the initial positions are too close to the opponent and the environment is not large enough. Intuitively, the observation shows we should increase the resolution when the agent and the opponent are not able to locate each other in the given planning horizon.

5.4 Gazebo Experiments

The previous simulation results show the proposed algorithm can be applied in the visibility-based scouting problem. However, some of the assumptions made for

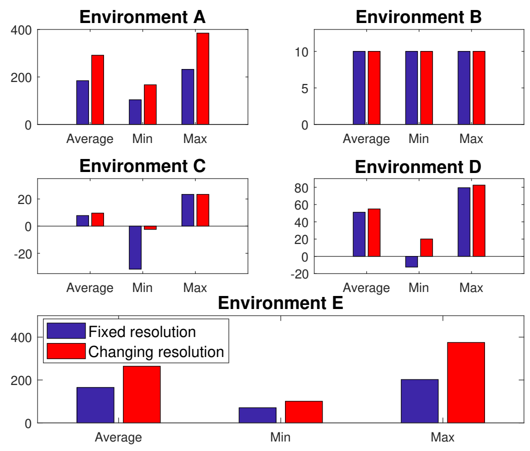


Fig. 13 Fix resolution vs. changing resolution. From environment A to E, the planning horizon for each steps are 20, 5, 15, 20, 30.

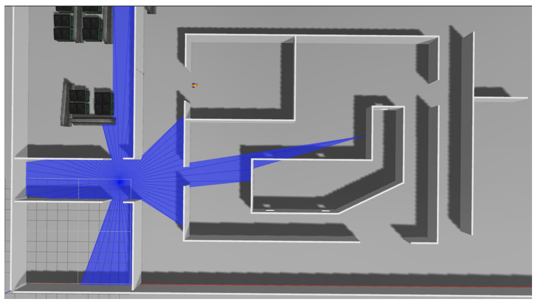
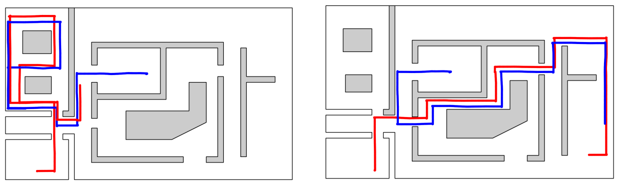


Fig. 14 Gazebo simulation environment. The agent and the opponent are simulated as differential drive robots equipped with a 360-degree lidar scanner to generate the visibility polygon (we only plot one robot's lidar scanner in blue).



(a) With a higher penalty $P = 50$.

(b) With a lower penalty $P = 3$.

Fig. 15 Gazebo qualitative experiments. Actual paths of the agent (red) and the opponent (blue) given by MCTS for 30 time steps. By varying the negative penalty values of being detected by the opponent.

the previous simulations may not hold in the real world. In this section, we discuss how to extend our algorithm to incorporate more realistic settings. We demonstrate this through ROS Gazebo simulations [32].

One of the assumptions is that the agent and the opponent operate in a grid-based environment. This is easily addressed in our algorithm. We do not actually need a grid-based environment since we can rebuild the tree after every step. Here, the tree is rooted at the current position of the agent and the opponent. Subsequent states in the tree are relative to the respective starting positions of the agent and the opponent.

In addition, the agent and the opponent move simultaneously and do not move in turns as the model assumes. While we assume the agent and the opponent move at the same speed in each turn, in practice, the two robots will not move with the exact pace for the same speed at all times. It is possible that one of the robots reaches its goal position before the other. This is where the anytime nature of MCTS comes in handy. We let MCTS run until one of the two robots reaches the goal positions. As soon as one robot reaches the goal positions, we use the solution that's returned by MCTS and use that to plan the actions for the next step. Here, the assumption we make is that once the two robots commit to an action at the start of the timestep; they do not change the action midway. Therefore, the agent can observe the action chosen by the opponent at the start of the timestep and use that to invoke MCTS, which runs until either the agent or the opponent reaches their goal position for the current timestep.

Figure 14 shows the setup where the agent and the opponent are simulated by using the model of a differential-drive robot and the two robots are equipped with a 360-degree lidar scanner (scan range marked with blue). We use the MCTS with changing resolution techniques ($C(k) = k$) to generate the paths for the agent path and the opponent (planning horizon $T = 5$). In these Gazebo experiments, we set the speeds of both the agent and the opponent as 0.2 and set the unit length between each grid cell as 3. When the agent moves, we take the agent and the opponent's goal positions as the input. After the agent reaches its current goal position, MCTS is terminated, and the agent will execute the best action generated by the MCTS.

Similar to the previous qualitative results, we show two examples of the policy found by the MCTS, using high and low negative penalty values in Figure 15 and the attached video.[2] In the video, we show an example that with a higher penalty $P = 50$, the agent tends to avoid all possible detection by the opponent, e.g., the agent only collects 449.88 positive reward but only being detected for only once. In contrast, with a lower negative reward, the agent prefers to explore more areas. With a lower penalty $P = 3$, the agent explores a much larger area and collects 1027.72 positive reward despite the opponent detecting it 22 time in 30 time steps.

[2] https://youtu.be/_UuawB8CZ-E

6 Conclusion

We introduce a new problem of maximizing visibility and minimizing detectability in an environment with an adversarial opponent. The problem can be solved using minimax and the MCTS to obtain an optimal strategy for the agent. Our main contribution is a set of pruning techniques that reduce the size of the search tree while still guaranteeing optimality. We also investigate how changing the resolution of the tree can lead to better performance in large environments. An immediate avenue for future work is to incorporate additional constraints, such as kinematic/dynamic constraints, as part of the planning process. Further, one may want to relax the assumption that the opponent's position is known at all times. This can be handled in MCTS by maintaining a belief over the opponents position. During the rollouts, one can randomly draw a sample from this belief. The resulting strategy can then take into account uncertain positions of the opponent.

LEGAL

Distribution A: Approved for public release; distribution unlimited. OPSEC # 3929. This research was supported in part by the Automotive Research Center (ARC) at the University of Michigan, with funding and support by the Department of Defense under Contract No. W56HZV-14-2-0001.

References

1. P. Tokekar and V. Kumar, "Visibility-based persistent monitoring with robot teams," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on.* IEEE, 2015, pp. 3387–3394.
2. C. Peng and V. Isler, "View selection with geometric uncertainty modeling," *arXiv preprint arXiv:1704.00085*, 2017.
3. A. Kim and R. M. Eustice, "Active visual slam for robotic area coverage: Theory and experiment," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 457–475, 2015.
4. G. Hollinger, S. Singh, J. Djugash, and A. Kehagias, "Efficient multi-robot search for a moving target," *The International Journal of Robotics Research*, vol. 28, no. 2, pp. 201–219, 2009.
5. L. Zhou, V. Tzoumas, G. J. Pappas, and P. Tokekar, "Resilient active target tracking with multiple robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 1, pp. 129–136, 2018.
6. S. Carlsson and B. J. Nilsson, "Computing vision points in polygons," *Algorithmica*, vol. 24, no. 1, pp. 50–75, 1999.
7. J. O'rourke, *Art gallery theorems and algorithms.* Oxford University Press Oxford, 1987.
8. Z. Zhang and P. Tokekar, "Non-myopic target tracking strategies for non-linear systems," in *Decision and Control (CDC), 2016 IEEE 55th Conference on.* IEEE, 2016, pp. 5591–5596.
9. E. Raboin, D. S. Nau, U. Kuter, S. K. Gupta, and P. Svec, "Strategy generation in multi-agent imperfect-information pursuit games." in *AAMAS*, 2010, pp. 947–954.
10. E. Raboin, U. Kuter, and D. Nau, "Generating strategies for multi-agent pursuit-evasion games in partially observable euclidean space," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, 2012, pp. 1201–1202.
11. N. M. Stiffler and J. M. O'Kane, "Complete and optimal visibility-based pursuit-evasion," *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 923–946, 2017.
12. V. Macias, I. Becerra, R. Murrieta-Cid, H. Becerra, and S. Hutchinson, "Image feedback based optimal control and the value of information in a differential game," *Automatica*, vol. 90, pp. 271–285, April 2018.
13. S. D. Bopardikar, F. Bullo, and J. P. Hespanha, "Sensing limitations in the lion and man problem," in *American Control Conference, 2007. ACC'07.* IEEE, 2007, pp. 5958–5963.
14. S. Jin and Z. Qu, "A heuristic task scheduling for multi-pursuer multi-evader games," in *Information and Automation (ICIA), 2011 IEEE International Conference on.* IEEE, 2011, pp. 528–533.
15. A. Quattrini Li, R. Fioratto, F. Amigoni, and V. Isler, "A search-based approach to solve pursuit-evasion games with limited visibility in polygonal environments," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018, pp. 1693–1701.
16. S. Gelly and Y. Wang, "Exploration exploitation in go: Uct for monte-carlo go," in *NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop*, 2006.
17. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Prentice Hall Press, 2009.
18. L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning.* Springer, 2006, pp. 282–293.
19. V. Lisy, V. Kovarik, M. Lanctot, and B. Bosansky, "Convergence of monte carlo tree search in simultaneous move games," in *Advances in Neural Information Processing Systems*, 2013, pp. 2112–2120.
20. Z. Zhang, J. Lee, J. M. Smereka, Y. Sung, L. Zhou, and P. Tokekar, "Tree search techniques for minimizing detectability and maximizing visibility," in *2019 International Conference on Robotics and Automation (ICRA).* IEEE, 2019, pp. 8791–8797.
21. D. Bhadauria and V. Isler, "Capturing an evader in a polygonal environment with obstacles." in *IJCAI*, 2011, pp. 2054–2059.
22. P. Surynek, "A novel approach to path planning for multiple robots in bi-connected graphs," in *2009 IEEE International Conference on Robotics and Automation.* IEEE, 2009, pp. 3613–3619.
23. S. Hrabar, "3d path planning and stereo-based obstacle avoidance for rotorcraft uavs," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2008, pp. 807–814.
24. L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," in *Robotics and Automation, 1996. Proceedings., 1996*

IEEE International Conference on, vol. 4. IEEE, 1996, pp. 3020–3025.
25. S. M. LaValle, *Planning algorithms.* Cambridge university press, 2006.
26. S. Karaman and E. Frazzoli, “Incremental sampling-based algorithms for a class of pursuit-evasion games,” in *Algorithmic foundations of robotics IX.* Springer, 2010, pp. 71–87.
27. S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,, 2016.
28. G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, “Monte-carlo tree search: A new framework for game ai.” in *AIIDE*, 2008.
29. H. Baier and M. H. Winands, “Monte-carlo tree search and minimax hybrids,” in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, 2013, pp. 1–8.
30. D. E. Knuth and R. W. Moore, “An analysis of alpha-beta pruning,” *Artificial intelligence*, vol. 6, no. 4, pp. 293–326, 1975.
31. K. J. Obermeyer and Contributors, “The visilibity library,” https://karlobermeyer.github.io/VisiLibity1/.
32. N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2149–2154.