ABSTRACT

Title of dissertation: ROBOT PLANNING IN ADVERSARIAL ENVIRONMENTS USING TREE SEARCH TECHNIQUES

Zhongshun Zhang, Doctor of Philosophy, 2021

Dissertation directed by: Dr. Pratap Tokekar
Department of Computer Science

One of the main advantages of robots is that they can be used in environments that are dangerous for humans. Robots can not only be used for tasks in known and safe areas but also in environments that may have adversaries. When planning the robot's actions in such scenarios, we have to consider the outcomes of a robot's actions based on the actions taken by the adversary, as well as the information available to the robot and the adversary. The goal of this dissertation is to design planning strategies that improve the robot's performance in adversarial environments. Specifically, we study how the availability of information affects the planning process and the outcome. We also study how to improve the computational efficiency by exploiting the structural properties of the underlying setting.

We adopt a game-theoretic formulation and study two scenarios: adversarial active target tracking and reconnaissance in environments with adversaries. A conservative approach is to plan the robot's action assuming a worst-case adversary with complete knowledge of the robot's state and objective. We start with such a "symmetric" information game for the adversarial target tracking scenario with noisy sensing. By using the properties of the Kalman filter, we design a pruning strategy to improve the efficiency of a tree search algorithm. We investigate the performance limits of the asymmetric version where the adversary can inject false sensing data. We then study a reconnaissance scenario where the robot and the adversary have symmetric information. We design an algorithm that allows a robot to scan more area while avoiding being detected by the adversary. The symmetric adversarial model may yield too conservative plans when the adversary may not have the same information as the robot. Furthermore, the information available to the adversary may change during execution. We then investigate the dynamic version of this asymmetric information game and show how much the robot can exploit the asymmetry in information using tree search techniques. Specifically, we study scenarios where the information available to the adversary changes during execution. We devise a new algorithm for this asymmetric information game with theoretical performance guarantees and evaluate those approaches through experiments. We use qualitative examples to show how the new algorithm can outperform symmetric minimax and use quantitative experiments to show how much the improvement is.

ROBOT PLANNING IN ADVERSARIAL ENVIRONMENTS USING TREE SEARCH TECHNIQUES

by

Zhongshun Zhang

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2021

Dedication

To my dearest mother, Sufang Zhong.

献给亲爱的母亲，钟素芳

Acknowledgments

Six years ago, I was fortunate to get the opportunity to cross the pacific and start my Ph.D. journey on the other side of the earth. During these six years, I have experienced difficulties in academic research, cultural difference, and the frustration of being rejected. Luckily, with enormous help and support from my advisor, family, and friends, I overcame all the difficulties and achieved the goal I set six years ago.

First of all, I would like to thank my advisor, Dr. Pratap Tokekar. I still remember the afternoon when I first talk with him at Virginia Tech, and I was lucky to become the first Ph.D. student in our lab. During the next five years, I can not remember how many times he instructed me to revise a paper to midnight, advised me to analyze a research problem in detail, and help me to improve my presentation tirelessly for all my oral exams. Without his support and instruction, I could not accomplish what I have presented today. I will keep my gratitude and what I learned from him for the rest of my life.

I would also like to extend my thanks to my committee members: Prof. Edmund H. Durfee, Prof. Dinesh Manocha, Prof. Jeffrey Herrmann, and Prof. Huaishu Peng for all the comments and suggestions they shared with me. Without their insightful suggestions, this dissertation would not be in its current shape.

Through my research projects, I was lucky to meet my corroborators in different projects. It was my honor to work with Prof. Edmund H. Durfee, Dr. Jonathon Smereka, Dr. Joseph Lee, and Rob Brady on the project supported by the Automotive Research Center (ARC). They provide me valuable suggestions almost every week during our collaboration. I will miss the morning during the past two years that we discuss the problem in detail. I am also inspired and benefit from my corroboration with my lab mates from Dr. Lifeng Zhou, Dr. Yoonchang Sung, Jingxi Chen, and Amrish Baskaran. I appreciate the time and opportunity to work together with these young scientists.

In addition, I got valuable feedback to improve my oral exam during my preliminary exam and final defense. I would like to thank Guangyao Shi, Lifeng Zhou, and Jun Liu for their time and help.

I had wonderful four years at Virginia Tech in a cheerful group with Ashish, Aravind, Kevin, Lifeng, Yoonchang, Tianshu, Maymoonah, Deeksha, Varun, and Harnaik. Also, another two unforgettable years in Maryland to meet Guangyao, Jingxi, Peihong, Amrish.

At the end of my Ph.D. episode, I would like to express my deepest appreciation to my loved parents, Chongcai Zhang and Sufang Zhong. No matter where I am, I know you will support me with your unconditional love. This dissertation is dedicated to my mother, Sufang Zhong, and in memory of her.

Table of Contents

Chapter 1: Introduction

1.1 Motivation

Autonomous robots are being used in a wide range of applications. Many of these applications require the robot to operate in an adversarial environment. In these applications, robots must consider not only their own actions but also the effects of the adversary. Examples include a reconnaissance mission in contested environments (Figure 1.1(a)), where the robot balances the trade-off between exploring and avoiding detection, and an adversarial target tracking or a pursuit-evasion setting (Figure 1.1(b)), where the problem is to devise a strategy to control the position of a robot that is tracking a possibly escaping target.

Adversarial planning is closely related to game theory, where robots and adversaries both try to maximize their self-perceived reward functions. The adversarial planning problem can be abstracted as a discrete, sequential, two-player game. Minimax tree search [38] and Monte-Carlo Tree Search (MCTS) [86] are two well-known algorithms to solve dis-

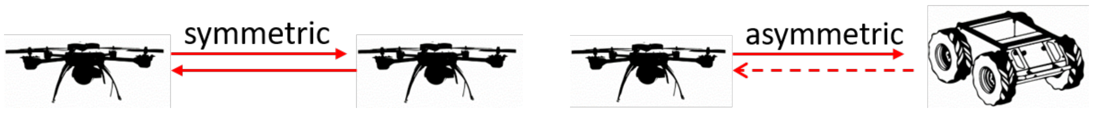
(a) Reconnaissance mission [1].

(b) Pursuit-Evasion [106].

Figure 1.1: Robot planning in adversarial environments

crete, two-player, zero-sum games. Both techniques build a search tree that contains all possible (or a subset of all possible) sequences of actions for both players over planning horizons. Minimax and MCTS have been applied in various adversarial planning problems such as non-cooperative control [22], active target tracking with a single robot [5], and board games [96].

This dissertation aims to design planning strategies that improve the robot's performance in adversarial environments. In particular, we study scenarios when the information available to the robot and adversary may not be the same and changes over time. We study how the availability of information affects the planning process and the outcome of the game. We also study how to improve computational efficiency by exploiting the structural properties of the underlying setting.



(a) Symmetric game: Both robot and its adversary have the same available information.

(b) Asymmetric game: The availability of information is different for robot and adversary.

Figure 1.2: The availability of information could be different for the robot and adversary. For instance, if the robot and the adversary have different sensors, they may not have the same information leading to asymmetry.

1.2 Background

In a minimax or Monte-Carlo search tree, we refer to the robot and the adversary as MAX and MIN players, respectively. At each time step[1], the robot moves first to maximize the total reward, and then the adversary moves to minimize the total reward. Given

[1]Even though the robot and the adversary move simultaneously, we model this problem as a turn-based game. We discuss this further in Chapter 6.

sufficient computational resources, we build the tree to search until the end of the game. In practice, we may have to cut-off the search after a finite planning horizon T, based on the computational time available for making each move. In the finite horizon case, we rebuild the tree after every move.

For example, consider an adversarial reconnaissance mission. We assume the robot can get a positive reward as it scans a new area but receives a negative penalty when the opponent detects it. So the robot tries to maximize its scanned area while avoiding being detected by the adversary. The minimax search tree starts with a root node that contains the initial position of the robot and the adversary. The MAX (i.e., robot) level expands the tree by creating a new branch for each control action for the robot. The MIN (i.e., adversary) level expands the tree by creating a new branch for each control action for the adversary. When the minimax tree is fully generated (i.e., the robot 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. The minimax policy chooses an action that maximizes and minimizes the backpropagated reward at the MAX and the MIN nodes, respectively. After executing one action, the robot will take a new measurement and rebuild the tree to plan for the next action.

Minimax is widely used in adversarial planning when the MAX player and MIN player both optimize the same objective function. In the following, we highlight scenarios where the MAX player and the MIN player may have asymmetric objectives. We also highlight the challenge of the computational complexity of tree search techniques.

1.2.1 Availability of information

A feature of the traditional search tree is that, by treating the adversary symmetrically, it only needs to compute one reward value for each leaf node in the tree. However, this limits

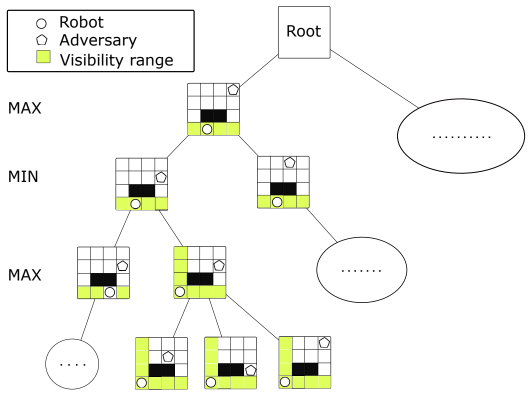


Figure 1.3: A (partial) minimax search tree. The root node contains the initial states of the agent and the opponent. Note that since we model what is actually a simultaneous move game as a turn-based game, two successive levels of the tree correspond to one time step. The agent moves to maximize the reward (MAX level) while the opponent moves to minimize the agent's reward (MIN level).

it to work properly when robot and the adversary maximize and minimize, respectively, the same objective function (Figure 1.2 (a)). The symmetric adversarial reward may yield plans that are too conservative when the adversary may not have the same information as the robot. For example, in an asymmetric setting, the robot may have better information than the adversary (Figure 1.2 (b)). It will be conservative to assume the adversary is as "smart" as the robot. Furthermore, the information available to the adversary may change during execution.

Another form of asymmetry is if the robot and the adversary may have different capabilities. For example, consider a target tracking problem where the adversary can inject false sensor data (i.e., corrupt the signal) of the robot. In such a case, the robot may not be able to detect the fact that the information it receives is being corrupted if the adversary is sophisticated. We propose a strategy, from the adversary's perspective, on how to design false measurement data that is injected to corrupt and mislead the output of a Kalman filter.

In this dissertation, I study how the availability of information affects the robot's planning process and the outcome of the game. I also study this from the adversary's perspective to see the limits of how the asymmetry can be exploited by the adversary.

1.2.2 Computational complexity

In a larger environment, the robot 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 robot and the adversary 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).

Monte-Carlo tree search, a randomized version of minimax search, can plan for a longer horizon compared to minimax, but is still expensive computationally when planning for longer horizons and when enough rollouts are required to build a good estimate of the returns.

One of the contributions in this dissertation is to improve the computational efficiency of the adversarial tree search algorithms. Depending on the problem, by exploiting the structural properties of the search tree, we design pruning strategies to improve the efficiency of the tree search algorithm.

1.3 Contributions

We adopt a game-theoretic formulation and study two scenarios: adversarial, active target tracking, and reconnaissance in environments with adversaries.

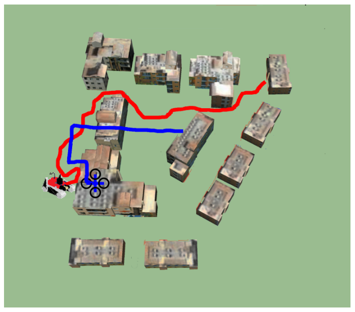
1.3.1 Target Tracking

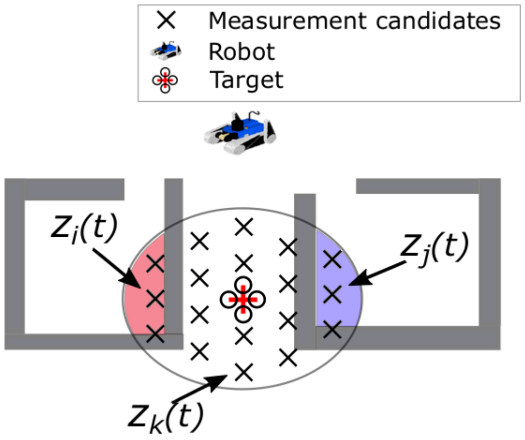
We start with the symmetric information game for the adversarial target tracking scenario (Figure 1.1(b)), where the robot has a tracking sensor with distance-dependent noise, and the adversarial target is actively escaping from the robot. The robot can use a Kalman filter to estimate the position of the target by fusing the noisy measurements. By using the properties of Kalman filters, we design a pruning strategy to improve the efficiency of a tree search algorithm. We then investigate the performance limits of the asymmetric version where the adversary can inject false data in the measurements obtained by the robot. Here the asymmetry exists because the robot is not aware of the false data injected by the adversary.

1.3.1.1 Symmetric Adversarial Target Tracking

In Chapter 2, we study the problem of devising a closed-loop strategy to control the position of a robot that is tracking a possibly moving target. The measurement noise depends on the relative states of the robot and the target. We consider scenarios where the measurement values are chosen by an adversary so as to maximize the estimation error. Furthermore, the target may be actively evading the robot. Our main contribution is to devise a closed-loop control policy for distance-dependent target tracking that plans for a sequence of control actions, instead of acting greedily. We consider a game-theoretic formulation of the problem and seek to minimize the maximum uncertainty (trace of the posterior covariance matrix) over all possible measurement values. We exploit the structural properties of a Kalman Filter to build a policy tree that is orders of magnitude smaller than naive enumeration while still preserving optimality guarantees. We show how to obtain even more computational savings by relaxing the optimality guarantees. The resulting algorithms are evaluated through simulations and experiments with real robots.

The preliminary version of this chapter was first presented in IEEE Conference on Decision and Control 2016 [116]. The complete version is published in IEEE Transactions on Control Systems Technology [117].


(a) A ground robot tracks a target aerial robot.



(b) The robot obtains a measurement with noise.

Figure 1.4: The robot obtains noisy measurements of the adversary's position to track the adversary, the adversary is actively escaping from the robot.

1.3.1.2 Asymmetric False Data Injection

In Chapter 3, we consider the problem of designing false measurement data that is injected to corrupt and mislead the output of a Kalman filter. Unlike prior work that focused on detection and filtering algorithms for the adversary, we study the problem from the adversary's point-of-view. In our model, the adversary can corrupt the measurements by injecting additive spoofing signals. The adversary seeks to create a separation between the estimate of the Kalman filter with and without spoofed signals. We present a number of results on how to inject spoofing signals while minimizing the magnitude of the injected signals. The resulting strategies are evaluated through simulations along with theoretical proofs. We also evaluate the spoofing strategy in the presence of a χ^2 spoof detector. We present a sufficient condition for this strategy to mislead the χ^2 failure detector.

This chapter was presented at the American Control Conference in 2018 [118].

1.3.2 Visibility-based Reconnaissance

The second scenario we consider in this dissertation is that of a reconnaissance mission. We introduce and study the problem of planning a trajectory for a robot to carry out a reconnaissance mission while avoiding being detected by an adversary (Figure 1.1(a)). We first consider the scenario that the robot and the adversary have symmetric information. We show traditional methods offer guarantees on the success, defined, for example, by exploring more area while avoiding being detected.

However, the symmetric adversarial model may yield plans that are too conservative when the adversary may not have the same information as the robot. Furthermore, the information available to the adversary may change during execution. We have investigated the static version of this “asymmetric” information game, and show how much the robot can exploit the asymmetry in information using tree search techniques.

1.3.2.1 Symmetric Adversarial Planning

In Chapter 4, we study the problem when the robot and the adversary have symmetric information. In our formulation, the robot receives a positive reward for increasing its visibility (by exploring new regions) and a negative penalty every time it is detected by the adversary. The objective is to find a finite-horizon path for the robot 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 robot and the adversary are located far from each other initially, we present a variable

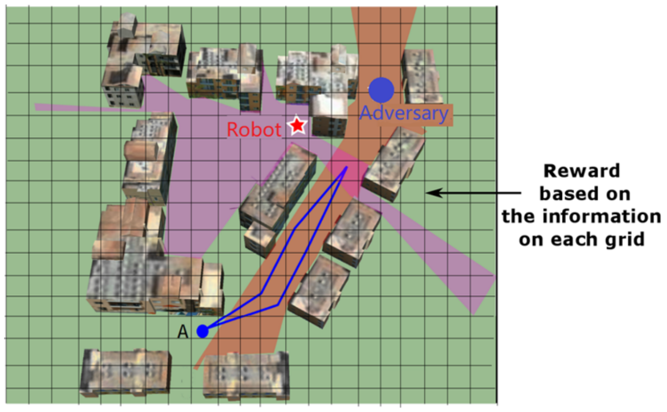


Figure 1.5: The robot is tasked with covering the environment while avoiding detection by the adversary.

resolution technique with a longer planning horizon to further reduce computational time. Simulation results show the effectiveness of the proposed strategies in terms of computational time.

A preliminary version of the chapter was presented at International Conference on Robotics and Automation (ICRA) 2019 [114]. The complete version of this chapter was published in Autonomous Robots [115].

1.3.2.2 Asymmetric Adversarial Planning

In Chapter 5, we study a more general problem where the robot has more information than the adversary. In asymmetric reconnaissance, there exist assets, and we assume only the robot knows the existence of the assets.

We consider the problem of planning a trajectory for a robot to carry out a reconnaissance mission while avoiding being observed by a mobile adversary, as well as avoiding the stationary assets being detected and collected by the adversary. Preventing the adversary from detecting the assets is more important since the penalty associated with collecting an asset is higher than that associated with the robot being detected. To collect an asset, the adversary must go close to it. We focus on scenarios where the adversary is not initially aware of the presence of the assets. We study how, and how much, the robot can exploit this asymmetry in information. Conventional game-theoretic planners such as minimax do not exploit this asymmetry. Instead, we introduce a new algorithm, DM1, which is specifically designed for such scenarios. The DM1 algorithm is built on the M1 algorithm in [25]. We extend the M1 algorithm by allowing the adversary's model to change dynamically. We evaluate this algorithm through simulations and present qualitative and quantitative results that show the DM1 search tree substantially improves the robot's performance when its asymmetric model of the adversary correctly characterizes the adversary's true knowledge.

The work in this chapter is being prepared to submit to a journal and is a collaboration with Rob Brady, Edmund H. Durfee, Jonathon M. Smereka, and Pratap Tokekar.

1.4 Organization

The rest of the dissertation is organized as follows. We introduce the symmetric version of the target tracking problem in Chapter 2. In Chapter 3, we present the asymmetric version of the target tracking problem by considering the adversary is able to inject false data into the robot's measurement. We then study another symmetric problem, this time of a reconnaissance mission in Chapter 4. We then consider the asymmetric version of the reconnaissance problem in Chapter 5. We conclude with a summary of our contributions and future research directions in Chapter 6.

Chapter 2: Symmetric Adversarial Target Tracking with Distance-Dependent Measurement Noise

In this chapter, we will start with the symmetric version of the target tracking problem. Our focus here is to improve the computational efficiency of the planner by exploiting the structural properties of the underlying problem.

2.1 Introduction

Tracking a moving, possibly adversarial target is a fundamental problem in robotics and has been studied has long been a subject of study [5, 10, 30, 32, 75, 109, 112]. Target tracking finds applications in many areas such as surveillance [83], telepresence [49], assisted living [72], and habitat monitoring [100]. Target tracking refers to broadly two classes of problems: (i) estimating the position of the target using noisy sensor measurements; and (ii) actively controlling the sensor position to improve the performance of the estimator. The second problem is distinguished as *active* target tracking and is the subject of study of this chapter.

One of the main challenges in tracking is that the target can be adversarial and actively avoid tracking by moving away from the robot. Furthermore, the value of future measurement locations can be a function of the unknown target state. Take as an example, a simple instance of estimating the unknown position of a stationary target where the measurement noise is a function of the distance between the robot and the target. If the true location of the

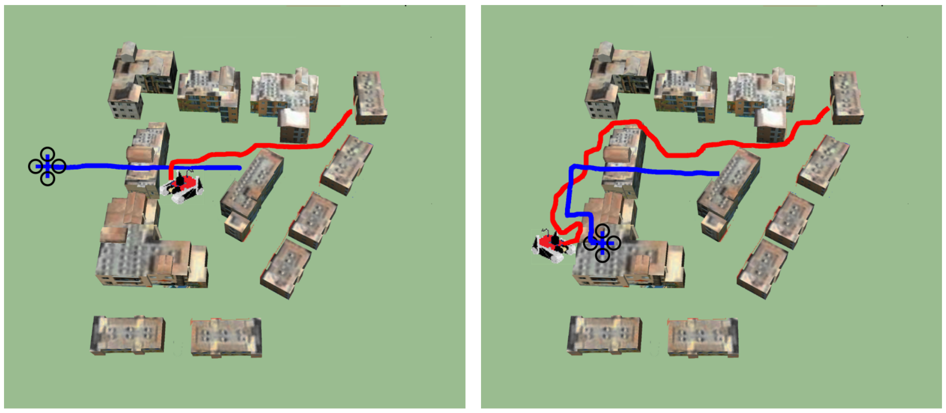

(a) Greedy. (b) Minimax.

Figure 2.1: A ground robot tracks a target aerial robot. The ground robot cannot move through the obstacles, whereas the aerial robot can fly over them. In (a), the ground robot plans greedily and gets stuck behind obstacles. In (b), the minimax plans non-myopically by predicting the target's adversarial moves and is able to plan around the obstacles.

target were known, the robot would always choose a control sequence that drives it closer to the target. Since, in practice, the true target location is unknown, we cannot determine such a control sequence exactly. A possible strategy, in this case, would be to plan with respect to the probability distribution of the target. However, the probability distribution itself will evolve as a function of the actual measurement values. This becomes even more challenging if the target is mobile.

In this chapter, we use an Kalman Filter (KF) to estimate the state of a moving target with a possibly distance-dependent measurement model where the standard deviation of the measurement noise is a function of the distance between the robot and the target. Although it is common to assume that the noise is independent of the state, many sensors such as infrared [13] and radio-based ranging [44] exhibit distance-dependent noise. Distance-dependent noise models have been used for planning, for example, for achieving distributed consensus [78]. We focus on the problem of tracking a single target by planning the motion of a mobile robot.

There has been recent work on resilient and robust game-theoretic algorithms in adversarial settings [92, 110, 114, 115]. We formulate the active adversarial target tracking problem as a minimax game. When planning non-myopically (for multiple steps in the future), one can enumerate all possible future measurements in the form of a tree. In particular, a minimax tree can be used to find the optimal (in the minimax sense) control policy for actively tracking a target [103]. The size of the minimax tree grows exponentially with the time horizon. The tree can be pruned using alpha-beta pruning [86]. Our main contribution is to show how the properties of an Kalman filter can be exploited to prune a more significant number of nodes without losing optimality. In doing so, we extend the pruning techniques first proposed by Vitus et al. [107] for linear systems with state independent noise. Using a minimax tree, we generalize these results to a system with distance-dependent noise. Our pruning techniques allow us to trade-off the size of the tree (equivalently, computation time) with the optimality guarantees of the algorithm. We demonstrate this effect in simulations and proof-of-concept indoor and outdoor tracking experiments. We also show how the tree can be updated online, when the measurements received and/or the target motion is not adversarial.

The rest of the chapter is organized as follows. We start with the related work in Section 2.2. The problem formulation is presented in Section 4.2. Our main algorithm is presented in Section 2.4. The pruning condition in different cases is discussed in Section 2.4. We validate the algorithm through simulations that are described in Section 2.5 and experiments on real robots reported in Section 2.6. Finally, we conclude with a brief discussion of future work in Section 3.6.

2.2 Related Work

The target tracking problem has been studied in various settings. Bar-Shalom et al. [10] have presented many of the commonly-used estimation techniques in target tracking. The five-part survey by Li and Jilkov [64] covers commonly-used control and maneuvering techniques for active target tracking.

Pursuit-evasion is a class of problems typically used to study adversarial target tracking [30]. Kolling and Carpin [57] study a pursuit-evasion problem on graphs that model the detection of intruders in complex indoor environments by robot teams. A typical approach is to model the problem as a non-cooperative game and use Pontryagin's minimum principle and the Bellman equation to find the optimal paths for the pursuer [11, 63]. These approaches typically assume noise-free sensing. Amongst pursuit-evasion works that explicitly address measurement noise are works by Vander Hook and Isler [105] using noisy bearing sensors.

Willman [109] studied the differential pursuit-evasion problem with state-independent Gaussian noise to the motion model as well as the measurement model. The author showed that the problem can be reduced to a deterministic one. Yaesh and Shaked [111] have shown the connection between the H_∞ optimal estimation theory and adversarial target tracking. Zengin and Dogan [112] have presented a real-time target tracking algorithm for autonomous UAVs in adversarial environments. More recently, Gu [41] proposed a minimax filter to estimate the state of an adversarial target using noisy measurements from a sensor network. Specifically, they modeled the estimation problem as a differential zero-sum game. Unlike these works, we use a minimax search tree to (non-myopically) plan for the control actions of the robot. Karaman et al. [47] showed how to solve a similar pursuit-evasion problem using RRT*. However, unlike the problem we study, their problem setup has no notion of measuring the target's (or the other agent's) position. As such, there is no need to run a state estimator and no way to handle distance-dependent measurement noise as we do in this chapter.

A search tree can provide optimal or near-optimal policies for target tracking. Li et al. [79] solve a visibility-based pursuit-evasion problem using tree search techniques.

However, building a search tree can be computationally expensive especially for large-scale instances. The key is to prune the tree to yield computational savings effectively. Vitus et al. [107] presented an algorithm that computes the optimal scheduling of measurements for a linear dynamical system. The goal is to track a linear dynamical system using a set of sensors such that one sensor can be activated at any time instance. The posterior covariance in estimating a linear system in a Kalman filter depends on the prior covariance and sensor variance but not on the future measurement values (unlike the case in non-linear systems). Thus, one can build a search tree enumerating all possible sensor selections and choosing the one that minimizes the final covariance. The main contribution of Vitus et al. is to present a pruning technique to reduce the size of the tree while still preserving optimality.

Monte-Carlo Tree Search (MCTS) [86], is an alternative algorithm to solve these discrete, two-player, zero-sum games. MCTS has been shown to be more effective in solving large two-player games, such as Go [37]. However, Li et al. [79] show that minimax has the advantage in finding deterministic solutions compare to MCTS in a pursuit-evasion game. Nevertheless, the pruning idea proposed in this chapter can also be applied in MCTS to remove redundant nodes.

Atanasov et al. [5] extended the idea of pruning search trees [107] to active target tracking with a single robot. A major contribution was to show that robot trajectories that are nearby in space can be pruned away (under certain conditions), leading to further computational savings. This was based on a linear system assumption. In this chapter, we build on these works and make progress towards generalizing the solution for distance-dependent observation systems.

A major bottleneck for planning for distance-dependent observation systems is that the future covariance is no longer independent of the actual measurement values. In Kalman filtering, the covariance update equation contains the noise variance, which in our case, depends on the position of the target. Since we do not know the actual position of the target, we use the estimated position which depends on the measurements. Thus, the search tree will have to include possible measurement values. Furthermore, finding an optimal path is no longer sufficient. Instead, one must find an optimal policy that prescribes the optimal set of actions for all possible measurements. We show how to use a minimax tree to find such an optimal policy while at the same time leveraging the computational savings that hold for the linear case.

2.3 Problem Formulation

We assume that the position of the robot is known accurately using onboard sensors (*e.g.*, GPS, laser, IMU, and cameras), and the accuracy of sensors is affected by distance. The motion model of the robot is given by:

$$X_r(t+1) = f(X_r(t), u_r(t)) \tag{2.1}$$

where, $f(\cdot)$ is the motion model of the robot, $X_r = [x_r(t), y_r(t)]^T \in \mathbb{R}^2$ is the position of robot and $u_r(t) \in \mathscr{U}_r(t)$ is the control input at time t.[1] $\mathscr{U}_r(t)$ is a finite space of control inputs. We assume there are n actions available as control inputs[2]:

$$\mathscr{U}_r(t) = \{u_{r1}(t), u_{r2}(t), \cdots, u_{rn}(t)\}.$$

The robot has a sensor that gets a measurement of the target's position. We assume that the target's motion model is given by:

$$X_o(t+1) = C_t X_o(t) + u_o(t) + v(t) \tag{2.2}$$

[1]This result can be extended to 3D and other state spaces

[2]Note that the subscript r denotes terms related to the robot and subscript o denotes the terms related to the target (not to be confused with the time). The time is always denoted inside paranthesis (t).

where, $X_o(t) = [x_o(t), y_o(t)]^T$ is the position of the target and $v(t)$ is the process noise drawn from a Gaussian distribution of known covariance. Here, $u_o(t) \in \mathscr{U}_o(t)$ is the control input for the target at time t with n' actions available inputs:

$$\mathscr{U}_o(t) = \{u_{o1}(t), u_{o2}(t), \cdots, u_{on'}(t)\}.$$

The task of the robot is to track the target using its noisy measurements. The measurements, $z(t)$, can be a function of the states of the target and the robot:

$$z(t) = HX_o(t) + w(X_r(t), X_o(t)) \tag{2.3}$$

The measurement noise, $w(t) \sim \mathscr{N}(0, \Sigma_w(t))$, is drawn from a 2D zero-mean Gaussian distribution. Specifically, we have

$$\Sigma_w(t) = \begin{bmatrix} \delta_1^2 + \delta_2^2 d(X_r(t), X_o(t)) & 0 \\ 0 & \delta_1^2 + \delta_2^2 d(X_r(t), X_o(t)) \end{bmatrix}.$$

Note that the noise variance $\Sigma_w(t)$ along each dimension is independent and given by $\delta_1^2 + \delta_2^2 d(X_r(t), X_o(t))$, where δ_1 and δ_2 are positive constant values. In general, the noise variance along the two dimensions can use different δ_1, δ_2 values; we use the same δ_1 and δ_2 values for both dimensions for ease of exposition. Here, $d(X_r(t), X_o(t)) =$

$$\begin{cases} \mathscr{C}, & ||X_r(t) - X_o(t)||_2 > \mathscr{B} \\ \dfrac{\mathscr{C}||X_r(t) - X_o(t)||_2}{\mathscr{B}}, & ||X_r(t) - X_o(t)||_2 \leq \mathscr{B}. \end{cases} \tag{2.4}$$

Here, $||\cdot||_2$ is the 2-norm which corresponds to the Euclidean distance between the robot and the target.

When the true distance between the robot and target is within $\mathscr{B}$, we assume that measurement noise variance is proportional to the true distance. When the true distance is greater than $\mathscr{B}$, the variance saturates at the maximum constant value of $\mathscr{C}$. Here, we consider a linear measurement model. A more general case is that the measurement model can be a non-linear function of the robot and target's state. For that case, we can use the Extended Kalman Filter and compute the observation matrix H by using the Jacobian of the measurement function [19].

The estimated position and the covariance matrix of the target at time t, $\hat{X}_o(t)$ and $\hat{\Sigma}_o(t)$, are given by the Kalman filter. The uncertainty in the estimate of the target's position is measured by the trace of the covariance matrix. The goal of the robot is to reduce the uncertainty in the target's estimate. The goal of the adversary is to increase the uncertainty. There are two types of adversaries:

1. **Game against an escaping target:** Here, the target actively chooses $u_o(t)$ to increase the estimation error. In our model, since the measurement noise is a function of the distance between the robot and the target, an adversarial target will choose to move away from the robot. When there are obstacles present in the environment, a myopic strategy that follows the estimated target position may get stuck (see Figure 2.1). Instead, we use minimax strategy that is able to track an escaping target better using a non-myopic approach.

2. **Game against nature:** Here, "nature" acts as an adversary and generates the worst-case measurements, $z(t)$, for the robot. In general, the measurements will not be the worst-case ones. However, planning against this adversary will lead to a policy that is robust to possible outlier measurements. Consider Figure 2.2 where the actual target is located in the corridor between the rooms. If at time t, the robot obtains an outlier measurement (say $z_i(t)$), the estimated target position will shift closer to the left room. A myopic planner will choose to move left and may get stuck in the

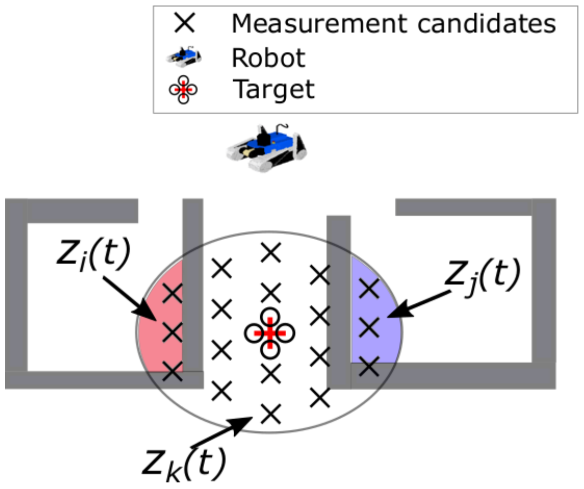


Figure 2.2: If the robot obtains a noisier measurement (say $z_i(t)$) at time t, a greedy strategy may follow the mean of the estimated target's position and get stuck in a room (left in the case of $z_i(t)$ even if the actual target is outside the room. A minimax strategy that plans against worst-case measurements for a finite horizon will instead hedge the bets, and stay in the middle until the target's estimate shrinks enough.

room. On the other hand, a minimax planner that considers adversarial measurements will choose to stay in the middle until such time that the estimated target covariance shrinks sufficiently. This way, even though the actual measurements are drawn from a random distribution, we can be robust to the noise by considering a game against nature.

The minimax tree strategy can be applied to either or both types of adversaries. Formally, the problem considered in this chapter can be formally stated as follows.

Problem 1. *Given an initial deterministic robot position, $X_r(0)$, and an initial target estimate, $[\hat{X}_o(0), \hat{\Sigma}_o(0)]$, find a sequence of control laws for the robot, $\sigma = u_r(0), u_r(1), \cdots, u_r(T)$ from time $t = 0$ to $t = T$ ($u_r(i) \in \mathscr{U}_r(i)$) to minimize trace of the covariance in the target's* *estimate at time $t = T$. That is,*

$$\min_{u_r(0),\ldots,u_r(T)} \max_{\substack{u_o(0),\ldots,u_o(T);\\ z(0)\ldots,z(T)}} tr(\Sigma_T). \tag{2.5}$$

such that,

$$\Sigma_{t+1} = \rho_t(\Sigma_t), \quad t = 0, 1, \cdots, T-1$$

where $\rho_t(\cdot)$ is the Kalman Riccati equation [58].

The Kalman Riccati equation [58], $\rho_t(\cdot)$, maps the current covariance matrix $\hat{\Sigma}_t$ to the covariance matrix at $t+1$ using the measurement $z(t)$:

$$\rho_t(\hat{\Sigma}_t) = C_t\hat{\Sigma}_tC_t^T - C_t\hat{\Sigma}_tH^T(H\hat{\Sigma}_tH^T + \Sigma_w)^{-1}H\hat{\Sigma}_tC_t^T + \Sigma_v. \tag{2.6}$$

Note that we solve Problem 1 at each time instance looking ahead T timesteps. Even though the solution for Problem 1 is a sequence of control inputs, $u_r(0),\ldots,u_r(T)$, we only apply the first one $u_r(0)$. Then, we use the actual measurement $z(0)$ to update the covariance matrix and then solve Problem 1 again for T timesteps. This is further explain in Section 2.4.4.

The true position of the target is unknown making it impossible to determine Σ_w exactly. Consequently, we use an estimate of Σ_w, denoted by $\hat{\Sigma}_w$, using the estimated target's position, $\hat{X}_o(k)$. Specifically, $\hat{\Sigma}_w$ is obtained by replacing $X_o(t)$ with $\hat{X}_o(k)$ in Equations (2.3)–(2.4). Since $\hat{X}_o(t)$ is computed using the values of $u_o(t)$ and $z(t)$, the Kalman Riccati map using the estimated noise covariance, $\hat{\Sigma}_w$, becomes a function of $u_o(t)$ and $z(t)$,

$$\begin{aligned}\rho_t(\hat{\Sigma}_t) = C_t\hat{\Sigma}_t C_t^T - \\ C_t\hat{\Sigma}_t H^T(H\hat{\Sigma}_t H^T + \hat{\Sigma}_w(u_o(t), z(t)))^{-1} H\hat{\Sigma}_t C_t^T + \Sigma_v.\end{aligned} \tag{2.7}$$

Remark 1. *The intuition behind the objective function is that it seeks to minimize the estimation uncertainty in the worst case. Also, note that the covariance will decrease as the robot and the target get closer since we assume the variance in the measurement noise is dependent on the distance between the robot and the target.*

Remark 2. *We can choose other measures for the objective function, such as the determinant [5]. We use trace since it is more robust in the following sense. For a 2D covariance matrix, the eigenvalues are the lengths of the major and minor axis of the uncertainty ellipse. If one of the two eigenvalues of the covariance matrix is close to* 0*, the determinant will also be close to* 0*, even if the other eigenvalue (i.e., the uncertainty along that direction) is very large. On the other hand, the trace will be large, if one of the two eigenvalues is large.*

Earlier works [5, 107] solve a similar problem but with a linear Gaussian system. The linearity assumption makes the Riccati equation independent of the position of the target (known as the separation principle). Consequently, they show an open loop policy can determine the optimal control sequence for the robot. In our case, the optimal control policy will be a closed-loop one since the measurement noise is a function of the position of the target. However, this generalization comes at the expense of discretization of the set of possible target measurements.

Equation (2.7) shows that trace at time T will be a function of $u_o(t)$ and $z(t)$. When we plan for a finite horizon, we do not know the exact sequence of actions the target takes,

$u_o(t)$, and true measurements, $z(t)$, that we obtain. Instead, we enumerate all possible actions the target can take and all possible measurements that we may obtain. Since the domain of the measurements is infinite, we discretize and assume that the measurement at any time step is chosen from one of m tuples of candidate measurements.[3] That is,

$$\hat{Z}(t) \in \{z_1(t), z_2(t), \cdots, z_m(t)\}. \tag{2.8}$$

For example, we can choose m candidate measurements from the data within 3 standard deviations of the mean value, which contain 99.7% of the possible measurements. Figure 2.3 shows a discretized Gaussian distribution.

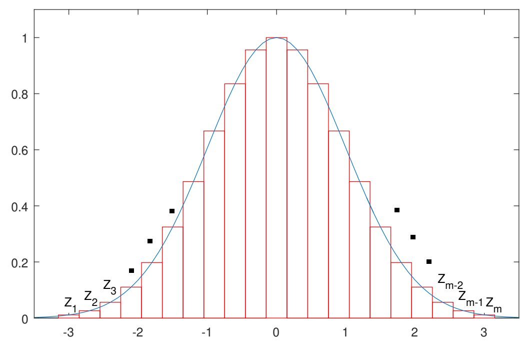


Figure 2.3: We discretize the set of candidate measurements using m samples drawn from a Gaussian distribution.

Note that the enumerated possible measurements are estimated while we build the search tree. To clarify the difference, we use the following notation:

- $\hat{Z}(t) = \{z_1(t), \ldots, z_i(t), \ldots, z_m(t)\}$: set of possible measurement that is obtained at time t.
- $z_i(t)$: possible measurement at time t.

[3]These m candidate measurements can be obtained by, for example, sampling from the continuous distribution of zero mean sensor noise around the current estimate of the target.

- $z(t)$: actual measurement obtained at time t. Here, $z(t)$ may or may not be the worst-case measurement.

In section 2.4, we present the minimax tree strategy and various pruning techniques that allow us to efficiently find the optimal closed-loop policy for this problem.

2.4 The minimax algorithm and pruning techniques

Table 2.1: Search Tree and Pruning Techniques for Various Models

Measurement Noise	**Target Motion**	**Strategy**	**Pruning Algorithm**
State-independent	Known/Stationary	Search tree	Algebraic redundancy (Theorem 1)
State-independent	Adversarial	Minimax tree	Alpha-beta pruning, Algebraic redundancy (Theorem 1)
Distance-dependent	Known/Stationary	Minimax tree	Alpha pruning
Distance-dependent	Adversarial	Minimax tree	Alpha-beta pruning, Algebraic redundancy (Theorem 2)

If the trajectory of the target is known, then we can find the optimal path for the robot using techniques such as dynamic programming. However, when tracking an adversarial target, we need to use game-theoretic planning to find the policy for the robot. We model Problem 1 as a sequential game[4] played between the robot and an adversary. The robot executes a control action, takes a measurement of the target, the target moves to the new position, and the sequence repeats. Based on this, we generate a search tree to find the optimal policy. The adversary (i,e. nature) chooses measurement noise and the target chooses actions at every step, whereas the robot chooses its own actions. By optimizing the minimax trace, the robot determines the best conservative policy.

[4]Note that, in practice, the robot and the target can move simultaneously.

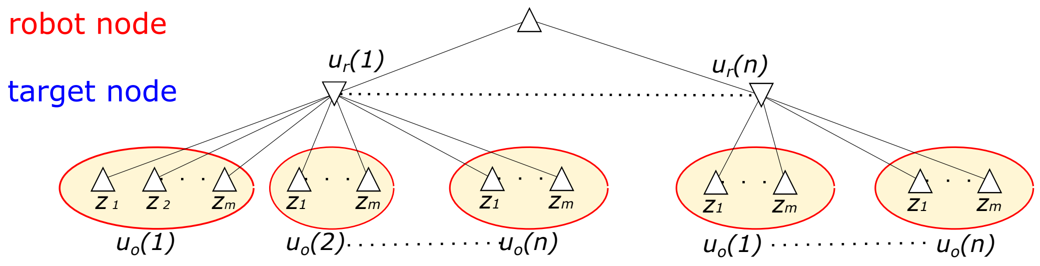


Figure 2.4: One step minimax tree enumerates all the possible actions the robot and target as well as all candidate measurements. Each node in the tree stores the robot position and estimated target position.

We find this optimal strategy by building a minimax tree. Figure 2.4 shows an example. This tree enumerates all possible control laws for the robot and the target and all possible measurements that the robot can obtain. A node on the kth level of the tree stores the position of the robot, $X_r(k)$, the estimated position of the target, $\hat{X}_o(k)$, and the covariance matrix $\hat{\Sigma}_k$. Each node at an odd level has one branch per control action of the robot. We term these nodes as "robot nodes." Each node at an even level has one branch per tuple of candidate measurement and candidate actions for the target. These nodes are termed as the "target nodes."

The robot's state and the target's estimate are updated appropriately along the control and measurement branches using the state transition equation (Equation 2.1) and the Kalman filter update equation, respectively. The minimax value is computed at the leaf nodes and is equal to the trace of the covariance matrix at that node. These values are propagated upwards to compute the optimal strategy.

The full enumeration tree has a size exponential in the number of control actions, candidate measurements, and the planning horizon. We present three pruning strategies to reduce the size of the tree. The first two are based on existing alpha-beta pruning while the third one is a new contribution of this chapter. Table 2.1 gives the summary of all pruning techniques.

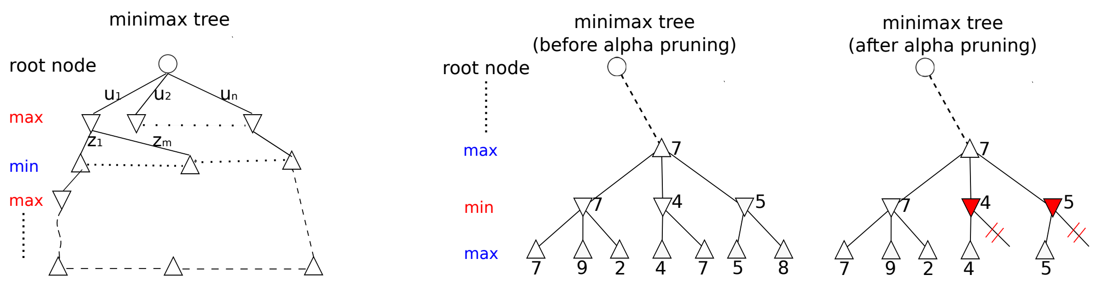


Figure 2.5: A minimax tree with alpha pruning. $\bigtriangledown$ and $\bigtriangleup$ are nodes in which we compute the minimum or maximum value of its children. The value at the leaf nodes equals the $\mathrm{tr}(\Sigma_k)$. $\bigtriangledown$ and $\bigtriangleup$ nodes represent robot and target nodes, respectively. The filled $\bigtriangledown$ are pruned by alpha pruning.

2.4.1 Alpha-Beta Pruning

As a first step in reducing the size of the tree, we use alpha-beta pruning [86]. The main idea in alpha-beta pruning is that if we have explored a part of the tree, we have an upper or lower bound on the optimal minimax value. For example, in alpha pruning where we consider the upper bound, when exploring a new node, n_i, if we find that the minimax value of the subtree rooted at n_i is greater than the upper bound found, that subtree does not need to be explored further. This is because an optimal strategy will never prefer a strategy that passes through n_i since there exists a better control policy in another part of the tree. Note that n_i must be a robot node. Figure 2.5 shows an example of alpha-beta pruning. Target nodes cannot be pruned since the robot has no control over the actual measurement values. That is, we only apply alpha-pruning. When the measurements and target's motion is known to be adversarial, then full alpha-beta pruning can be applied. In such a case, the target nodes can be pruned away as well.

2.4.2 Algebraic Redundancy Pruning

Vitus et al. [107] presented algebraic redundancy pruning of search trees (not minimax trees) for linear systems with state-independent noise. The key idea is that if a node, n_i, has higher uncertainty than another node, n_j at the same level, then any descendant of n_i will always have higher uncertainty than some descendant of n_j. Therefore, n_i is redundant and can be pruned away. They use the monotonicity and concavity of the Riccati mapping in linear systems with state-independent noise to prove the following result.

Theorem 1 (Algebraic Redundancy [107])**.** *Let $\mathscr{H} = \{(X_r^i(t), \Sigma_t^i)\}$ be a set of n nodes at the same level of the tree. If there exist non-negative constants $\alpha_1, \alpha_2, \ldots, \alpha_k$ such that,*

$$\Sigma_t^p \succeq \sum_{i=1}^{k} \alpha_j \Sigma_t^j \quad \textit{and} \quad \sum_{i=1}^{k} \alpha_i = 1$$

then the node $\left(X_r^p(t), \Sigma_t^p\right)$ is regarded as algebraically redundant[5] *with respect to $\mathscr{H} \setminus \{(X_r^p(t), \Sigma_t^p)\}$ and $(X_r^p(t), \Sigma_t^p)$ and all of its descendants can be pruned without eliminating the optimal solution from the tree.*

They prove that the trace of any successor of $(X_r^p(t), \Sigma^p(t))$ cannot be lower than one of the successors of $\mathscr{H} \setminus \{(X_r^p(t), \Sigma^p(t))\}$. Our main insight is that, a similar redundancy constrained can be defined for the non-linear case with suitable additional constraints as described below.

We extend these ideas for minimax trees with possibly distance-dependent noise. We first prove the monotonicity of distance-dependent Riccati equation.

Lemma 1. *Let A and B be two nodes in the same level of the minimax tree and Σ_t^A, S^A and Σ_t^B, S^B be the corresponding covariance matrices and measurement noise covariance matrices. If $\Sigma_t^A \succeq \Sigma_t^B$ and $S^A \succeq S^B$, then after applying one step of the Riccati equation, we have $\rho(\Sigma_t^A) \succeq \rho(\Sigma_t^B)$.*

Note that from the definition of the covariance matrix, the matrices S^A, S^B are positive definite, $S^A \succeq 0$, $S^B \succeq 0$.

[5] $M \succeq N$ represents that $M - N$ is positive semi-definite.

We will show conditions under which a node A is redundant with respect to a set of nodes, termed as candidate set, that is already present in the tree. Figure 2.6 shows an example.

Definition 2.1. *Let $\mathscr{H}$ be a set of nodes. The set $\mathscr{H}$ is called as a* candidate set *with respect to some node A if (i) the nodes in $\mathscr{H}$ are at the same level as that of A; and (ii) every node $B \in \mathscr{H}$ is on the optimal minimax path (the highlighted path in Figure 2.6) for the subtree rooted at the least common ancestor of A and B.*

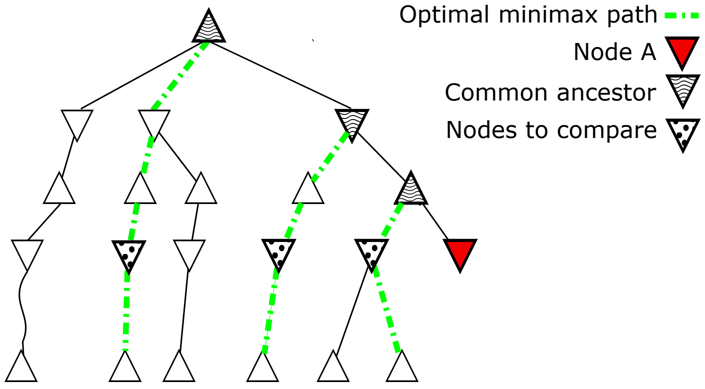


Figure 2.6: Nodes in the candidate set, $\mathscr{H} = \{(X_r^i(t), \hat{X}_o^i(t), \hat{\Sigma}_t^i)\}$, of node A are marked as red. Their least common ancestor is highlighted.

Before we present the full details, we list the conditions that will be used in Theorem 2. We have more conditions since pruning with distance-dependent noise is a general version of Theorem 1.

Let $\mathscr{H} = \{(X_r^i(t), \hat{X}_o^i(t), \hat{\Sigma}_t^i)\}$ be the candidate set of N nodes with respect to some node $A = (X_r^A(t), \hat{X}_o^A(t), \hat{\Sigma}_t^A)$. The conditions are as follows:

(C1) the robot and estimated target states are identical, i,e. $X_r^A(t) = X_r^i(t)$ and $\hat{X}_o^A(t) = \hat{X}_o^i(t)$ for all i in $\mathscr{H}$;[6]

[6]For time-invariant linear systems with constant H and C, condition (C1) is not required since all the covariance matrices are updated through the same Kalman filter Riccati equations.

(C2) the least common ancestor of A with any other node in $\mathscr{H}$ is a MIN (robot) node;

(C3) the least common ancestor of A with any other node in $\mathscr{H}$ is a MAX (target) node;

(C4) HH^T is invertible and there exist non-negative constants $\alpha_1, \alpha_2, \ldots, \alpha_N$ such that,

$$\Sigma_t^A \succeq \sum_{i=1}^{N} \alpha_i \Sigma_t^i \tag{2.9}$$

$$S_t^A \succeq \sum_{i=1}^{N} \alpha_j S_t^i + (T-t)\left(\delta_1^2 + \delta_2^2 \mathscr{C}\right) I_{2\times 2} \tag{2.10}$$

where $\sum_{i=1}^{N} \alpha_i = 1$.

(C5) HH^T is invertible and there exist non-negative constants $\alpha_1, \alpha_2, \ldots, \alpha_N$ such that,

$$\Sigma_t^A \preceq \sum_{i=1}^{N} \alpha_i \Sigma_t^i \tag{2.11}$$

$$S_t^A \preceq \sum_{i=1}^{N} \alpha_j S_t^i + (T-t)\left(\delta_1^2 + \delta_2^2 \mathscr{C}\right) I_{2\times 2} \tag{2.12}$$

Our main result is stated as follows.

Theorem 2. *[Distance-dependent Algebraic Redundancy] Let $\mathscr{H} = \{(X_r^i(t), \hat{X}_o^i(t), \hat{\Sigma}_t^i)\}$ be the candidate set of N nodes with respect to some node $A = (X_r^A(t), \hat{X}_o^A(t), \hat{\Sigma}_t^A)$. If the node A satisfies (C1), (C2) and (C4), then there exists a node in $\mathscr{H}$, say B, such that:*

$$tr(\Sigma_T^A) \geq tr(\Sigma_T^B).$$

Similarly, if node $A = (X_r^A(t), \hat{X}_o^A(t), \hat{\Sigma}_t^A)$ satisfies (C1), (C3) and (C5), then there exists *a node in $\mathscr{H}$, say B, such that:*

$$tr(\Sigma_T^A) \leq tr(\Sigma_T^B).$$

In both cases, the node A can be pruned from the minimax tree without affecting the optimal policy.

Proof. For nodes A and B at the same level k with $S^A \succeq S^B$, and $\Sigma_t^A \succeq \Sigma_t^B$. our goal is to prove that $\rho_t(\Sigma_t^A) \succeq \rho_t(\Sigma_t^B)$.

Applying the Riccati equation we have:

$$\begin{aligned}
&\rho_t(\Sigma_t^A) - \rho_t(\Sigma_t^B) \\
=&C_t\Sigma_t^A C_t^T - C_t\Sigma_t^A H^T \left(H\Sigma_t^A H^T + S^A\right)^{-1} H\Sigma_t^A C_t^T - C_t\Sigma_t^B C_t^T \\
&+ C_t\Sigma_t^B H^T \left(H\Sigma_t^B H^T + S^B\right)^{-1} H\Sigma_t^B C_t^T.
\end{aligned} \tag{2.13}$$

Define,

$$\begin{aligned}
K(\Sigma_t) &\triangleq -C_t\Sigma_t H^T (H\Sigma_t H^T + S)^{-1}, \\
F(\Sigma_t) &\triangleq C_t - (C_t\Sigma_t H^T)(H\Sigma_t H^T + S)^{-1} H.
\end{aligned}$$

Note that, $F(\Sigma_t) = C_t + K(\Sigma_t)H$, and $K(\Sigma_t)(C_t\Sigma_t H^T)^T = -K(\Sigma_t)(H\Sigma_t H^T + S)K^T(\Sigma_t)$.

Thus, $\rho_t(\Sigma_t^A) - \rho_t(\Sigma_t^B) = C_t\Sigma_t^A C_t^T + K(\Sigma_t^A)H\Sigma_t^A C_t^T - C_t\Sigma_t^B C_t^T - K(\Sigma_t^B)H\Sigma_t^B C_t^T$.

Then, we have,

$$
\begin{aligned}
&\rho_t(\Sigma_t^A) - \rho_t(\Sigma_t^B) - F(\Sigma_t^A)(\Sigma_t^A - \Sigma_t^B)F(\Sigma_t^A)^T \\
=&C_t\Sigma_t^A C_t^T + K(\Sigma_t^A)H\Sigma_t^A C_t^T - [C_t\Sigma_t^B C_t^T + K(\Sigma_t^B)H\Sigma_t^B C_t^T] \\
&- [C_t + K(\Sigma_t^A)H](\Sigma_t^A - \Sigma_t^B)[C_t + K(\Sigma_t^A)H]^T \\
=&K(\Sigma_t^A)(C_t\Sigma_t^A H)^T - K(\Sigma_t^B)C_t\Sigma_t^B H^T - K(\Sigma_t^A)H(\Sigma_t^A - \Sigma_t^B)C_t^T - C_t(\Sigma_t^A - \Sigma_t^B)H^T K^T(\Sigma_t^A) \\
&- K(\Sigma_t^A)H(\Sigma_t^A - \Sigma_t^B)H^T K^T(\Sigma_t^A) \\
=&K(\Sigma_t^A)(C_t\Sigma_t^A H)^T - K(\Sigma_t^B)C_t\Sigma_t^B H^T - K(\Sigma_t^A)[H\Sigma_t^A C_t^T - H\Sigma_t^B C_t^T] \\
&- [C_t\Sigma_t^A H^T - C_t\Sigma_t^B H^T]K^T(\Sigma_t^A) - K(\Sigma_t^A)H(\Sigma_t^A - \Sigma_t^B)H^T K^T(\Sigma_t^A) \\
=&K(\Sigma_t^A)(C_t\Sigma_t^A H)^T - K(\Sigma_t^B)C_t\Sigma_t^B H^T - K(\Sigma_t^A)H\Sigma_t^A C_t^T + K(\Sigma_t^A)H\Sigma_t^B C_t^T \\
&- C_t\Sigma_t^A H^T K^T(\Sigma_t^A) + C_t\Sigma_t^B H^T K^T(\Sigma_t^A) - K(\Sigma_t^A)H(\Sigma_t^A - \Sigma_t^B)H^T K^T(\Sigma_t^A),
\end{aligned}
\tag{2.14}
$$

note the following matrix is symmetric,

$$K(\Sigma_t)(C_t\Sigma_t H^T)^T = -K(\Sigma_t)(H\Sigma_t H^T + S)K^T(\Sigma_t),$$

we have,

$$K(\Sigma_t^A)(C_t\Sigma_t^A H)^T = C_t\Sigma_t^A H^T K^T(\Sigma_t^A).$$

The first and the fifth term in (2.14) can be canceled,

$$\begin{aligned}
=&-K(\Sigma_t^B)C_t\Sigma_t^B H^T - K(\Sigma_t^A)H\Sigma_t^A C_t^T + K(\Sigma_t^A)H\Sigma_t^B C_t^T + C_t\Sigma_t^B H^T K^T(\Sigma_t^A)\\
&-K(\Sigma_t^A)H(\Sigma_t^A - \Sigma_t^B)H^T K^T(\Sigma_t^A)\\
=&-K(\Sigma_t^B)C_t\Sigma_t^B H^T + K(\Sigma_t^A)(H\Sigma_t^A H^T + S^A)K^T(\Sigma_t)\\
&+K(\Sigma_t^A)H\Sigma_t^B C_t^T + C_t\Sigma_t^B H^T K^T(\Sigma_t^A) - K(\Sigma_t^A)H(\Sigma_t^A - \Sigma_t^B)H^T K^T(\Sigma_t^A),
\end{aligned} \tag{2.15}$$

combine the second and the last term,

$$\begin{aligned}
=&-K(\Sigma_t^B)(C_t\Sigma_t^B H^T) + K(\Sigma_t^A)(C_t\Sigma_t^B H^T)^T + (C_t\Sigma_t^B H^T)K^T(\Sigma_t^A)\\
&+K(\Sigma_t^A)(H\Sigma_t^B H^T + S^A)K^T(\Sigma_t^A),
\end{aligned}$$

note that,

$$\begin{aligned}
&(C_t\Sigma_t H^T)^T = (H\Sigma_t H^T + S)K^T(\Sigma_t)\\
=&\quad K(\Sigma_t^B)(H\Sigma_t^B H^T + S^B)K^T(\Sigma_t^B) - K(\Sigma_t^A)(H\Sigma_t^B H^T + S^B)K^T(\Sigma_t^B)\\
&-K(\Sigma_t^B)(H\Sigma_t^B H^T + S^B)K^T(\Sigma_t^A) + K(\Sigma_t^A)(H\Sigma_t^B H^T + S^A)K^T(\Sigma_t^A)\\
=&(K(\Sigma_t^B) - K(\Sigma_t^A))(H\Sigma_t^B H^T + S^B)(K(\Sigma_t^B) - K^T(\Sigma_t^A))^T\\
&+K(\Sigma_t^A)(S^A - S^B)K^T(\Sigma_t^A).
\end{aligned}$$

We have,

$$
\begin{aligned}
&\rho_t(\Sigma_t^A) - \rho_t(\Sigma_t^B) \\
=&F(\Sigma_t^A)(\Sigma_t^A - \Sigma_t^B)F^T(\Sigma_t^A) + K(\Sigma_t^A)(S^A - S^B)K^T(\Sigma_t^A) \\
&+ (K(\Sigma_t^B) - K(\Sigma_t^A))(H\Sigma_t^B H^T + S^B) \quad (K(\Sigma_t^B) - K(\Sigma_t^A))^T \\
=&C_t H(\Sigma_t^A - \Sigma_t^B)H^T C_t^T + K(\Sigma_t^A)\Big((H\Sigma_t^A H^T + S^A) - (H\Sigma_t^B H^T + S^B)\Big)K^T(\Sigma_t^A) \\
&+ (K(\Sigma_t^B) - K(\Sigma_t^A))(H\Sigma_t^B H^T + S^B) \cdot (K(\Sigma_t^B) - K(\Sigma_t^A))^T
\end{aligned}
\tag{2.16}
$$

since $S^A \succeq S^B$, we have $H\Sigma_t^A H^T + S^A \succeq H\Sigma_t^B H^T + S^B$, and $\Sigma_t^A \succeq \Sigma_t^B$. Thus,

$$\rho(\Sigma_t^A) - \rho(\Sigma_t^B) \succeq 0 \tag{2.17}$$

$\square$

These conditions are based on the algebraic redundancy conditions given in [107] (Theorem 1) for the sensor scheduling problem. The search tree in [107] is a non-adversarial search tree whereas we generalize these conditions to the adversarial case using a minimax search tree with two types of nodes (MIN and MAX).

The conditions in Theorem 2 state when a node A can be made redundant by another node B. The two nodes must have the same robot and estimated target states as given in (C1). In informal terms, Theorem 2 states that if node B is better than node A, some descendant of node B will be better than that of node A. As a result, node A can be pruned from the tree. The notion of "better" depends on whether their common ancestor is a MIN (robot controlled) or a MAX (target controlled) node, given in (C2) and (C3). If their common ancestor is a MIN node, then we say B is better if it has lower uncertainty than A. In this case, the robot will never choose the path that leads to A as opposed to B. If the common ancestor is a max node, then we say B is better if it has higher uncertainty than A.

As a result, the target will always prefer the path that leads to B than A. In both cases, the optimal path will not include A which can, therefore, be pruned away.

We use the above result to prune away nodes while building the tree. Note that the algebraic redundancy pruning is more effective when the tree is being built in a breadth-first fashion (since we compare nodes at the same level). On the other hand, the alpha-beta pruning is useful only when the tree is built in a depth-first fashion. In order to apply both pruning strategies, the tree must be built depth-first. While adding a new node to the tree, we check whether the conditions in Theorem 2 are satisfied with respect to all other existing nodes at the same level. In order to check for the optimal path, we require at least one path to a leaf node from the current node. Therefore, the conditions can be checked for all predecessors of the current node under consideration. If the conditions are met for any predecessor, then the predecessor node (and all its descendants) are pruned from the tree.

Since $\mathscr{H}$ can be of any size, checking for conditions (C4) and (C5) can be computationally expensive and requires solving a Linear Matrix Inequality (LMI). If we restrict $\mathscr{H}$ to contain only one node, then conditions (C4) and (C5) amount to checking positive semidefiniteness of a matrix which can be done much faster. However, this would mean fewer nodes get pruned away. As the level of the tree grows, the number of nodes increases exponentially. We can trade-off the two factors, by solving LMI at lower depths in the tree (when there are fewer nodes), and then only making pairwise comparisons for higher depths. We provide a more detailed discussion of the steps we implement to check (C1)–(C5) in the simulation section (Section 2.5). In the next subsection (Section 2.4.3), we describe more ways of saving computational time at the expense of optimality.

In an environment with $\mathscr{K}$ gridpoints, there are $\mathscr{K}^2$ possible combinations of the robot's and the target's position. Thus, at most $\mathscr{K}^2$ nodes are listed at each MIN level of the search tree in the best case. And by considering the m measurements in the MAX level for each grid, at most $m \cdot \mathscr{K}^2$ nodes are listed at MAX level. The size of the full tree will therefore be $\Omega(m\mathscr{K}^2T + \mathscr{K}^2T)$. The $\Omega(\cdot)$ indicates a lower bound. Therefore, in the best-case with our pruning conditions the size of the tree is $\Theta((m+1)\mathscr{K}^2 \cdot 2T)$. 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 actual size will be between the best and worst-case.

2.4.3 Sub-optimal Pruning algorithm

We can further reduce the number of branches at the expense of losing optimality by relaxing the alpha-pruning and algebraic redundancy constraints. We use two parameters $\varepsilon_1 > 0$ and $\varepsilon_2 > 0$ as relaxation parameters for alpha pruning and algebraic redundancy pruning, respectively. In each case, we bound the loss in optimality as a function of the parameters.

Specifically, while building the tree, we prune away a node if it satisfies either of the following two conditions. When checking for alpha pruning, we prune a node if its alpha value is greater than or equal to the best minimax value found so far minus ε_1. Similarly, we replace the first constraint of (C4) (Equation (2.9)) in Theorem 2 with the following:

$$\Sigma_t^A + \varepsilon_2 \succeq \sum_{i=1}^{N} \alpha_i \Sigma_t^i. \tag{2.18}$$

Similar condition can be applied to the first constraint of (C5) given in Equation (2.10) of Theorem 2.

By varying ε_1 and ε_2, we can vary the number of nodes in the search tree. Next we bound the resulting loss in the optimality of the algorithm.

Theorem 3 (ε_1-alpha pruning)**.** *Let $J_{2k}^* = tr(\hat{\Sigma}_{2k}^*)$ be the optimal minimax value returned by the full enumeration tree. If $J_{2k}^{\varepsilon_1} = tr(\hat{\Sigma}_{2k}^{\varepsilon_1})$ is the value returned by the ε_1–alpha pruning* *algorithm, then* $0 \leq J_{2k}^{\varepsilon_1} - J_{2k}^{*} \leq \varepsilon_1$.

The proof follows directly from the fact that if a node on the optimal policy, say n_i is pruned away, then the alpha value at n_i is at most the alpha value of some other node, say n_j, that is present in the tree minus ε_1. The alpha value of n_j cannot be less than the value returned by the ε_1 algorithm. The bound for ε_2-algebraic redundancy pruning is more complicated.

Theorem 4 (ε_2-Distance dependent algebraic redundancy pruning)**.** *Let* $J_{2k}^{*} = tr(\hat{\Sigma}_{2k}^{*})$ *be the optimal minimax value returned by the full enumeration tree of* $2k$ *levels. If* $J_{2k}^{\varepsilon_2} = tr(\hat{\Sigma}_{2k}^{\varepsilon_2})$ *is the value returned by the* ε_2*–algebraic redundancy pruning algorithm, then*

$$0 \leq J_{2k}^{\varepsilon_2} - J_{2k}^{*} \leq B^{\varepsilon_2}$$

where,

$$B^{\varepsilon_2} = tr\left\{ \sum_{j=1}^{k} \left[\prod_{i=j-1}^{0} (F_i(\Sigma)\Phi_{2i}(\Sigma)) \prod_{i=0}^{j-1} (F_i(\Sigma)\Phi_{2i}(\Sigma))^T \right] \varepsilon_2 \right\}$$

where, $F_i(\Sigma) = C - CK_i(\Sigma)H$ *and* $K_i(\Sigma)$ *is the Kalman gain given by* $K_i(\Sigma) = \Sigma H^T (H\Sigma H^T + \Sigma_w)^{-1}$*, and* $\Phi_{2k}(\cdot)$ *is the application of the Riccati equation* $\rho(\cdot)$*, over k measurement steps:*

$$\Phi_{2k}(\cdot) = \underbrace{\rho_{2(k-1)}(\rho_{2(k-2)}(\dots\rho_0(\cdot)))}_{k \text{ steps } \rho(\cdot)}. \tag{2.19}$$

Note that in Theorem 4, the conditions (C1)–(C3) are still required. (C1) and (C2) are always required for the MAX level, (C1) and (C3) are required for the MIN level. Theorem 4 relaxes the condition for (C4) for the MAX level, and (C5) for the MIN level.

Proof. For some level i, suppose that we prune a node on the optimal policy. We have,

$$\mathrm{tr}(H\left(\Sigma_{2i}^{\varepsilon_2}\right)H^T) \leq \mathrm{tr}(H\left(\Sigma_{2i}^{*}+\varepsilon_2 I\right)H^T),$$

we apply the following two proprieties from [107] Second(Lemma 1 and Theorem 3 in [107]), $\forall \Sigma, Q \in \mathbb{R}^{n\times n}$ and $\varepsilon \geq 0$:

$$\rho_{2i}(\Sigma+\varepsilon Q) \preceq \rho_{2i}(\Sigma)+F_i(\Sigma)QF_i^T(\Sigma)\varepsilon \tag{2.20}$$

$$\Phi_{2k}(\Sigma+\varepsilon_2 Q) \preceq \Phi_{2k}(\Sigma)+\left[\prod_{i=k-1}^{0}\left(F_i(\Sigma)\Phi_{2i}(\Sigma)\right)Q\prod_{i=0}^{k-1}\left(F_i(\Sigma)\Phi_{2i}(\Sigma)\right)^T\right]\varepsilon_2 \tag{2.21}$$

Let $\left\{\hat{\Sigma}_i^*\right\}_{i=1}^k$ be the series of covariance matrices along the optimal minimax trajectory. Suppose that the sequence of covariace matrices along the optimal trajectory returned by ε_2–algebraic redundancy pruning algorithm is $\left\{\hat{\Sigma}_i^{\varepsilon_2}\right\}_{i=1}^k$. We get,

$$\hat{\Sigma}_i^{\varepsilon_2} \preceq \hat{\Sigma}_i^* + \varepsilon_2 I, \quad \forall i = 1,2,\ldots,k$$

Consider the worst case, the ε_2 pruning condition was used at all the remaining $2k$ levels of the minimax tree. It will be at most be pruned by k times. Thus, added all the bounded value from 1 to k. we obtain the desired bound:

$$\begin{aligned}
0 &\leq J_{2k}^{\varepsilon_2} - J_{2k}^{*} = (\hat{\Sigma}_k^{\varepsilon_2}) - (\hat{\Sigma}_k^{*}) \\
&\leq \left\{\sum_{j=1}^{k}\left[\prod_{i=j-1}^{0}\left(F_i(\Sigma)\Phi_{2i}(\Sigma)\right)\prod_{i=0}^{j-1}\left(F_i(\Sigma)\Phi_{2i}(\Sigma)\right)^T\right]\varepsilon_2\right\} \\
&= B^{\varepsilon_2}
\end{aligned}$$

$\square$

By combining the two results, let $J_{2k}^{\varepsilon_1,\varepsilon_2}$ be the upper bound when we apply ε_1-alpha pruning and ε_2-Distance dependent algebraic redundancy pruning together, we get

$$0 \leq J_{2k}^{\varepsilon_1,\varepsilon_2} - J_{2k}^{*} \leq \max\{\varepsilon_1, B^{\varepsilon_2}\}.$$

The parameter ε_1 has a direct relationship with the suboptimality. On the other hand, ε_2 has a more indirect relationship with the suboptimality. In Section 2.5, we plot the upper bound of $J_{2k}^{\varepsilon_1}$ and $J_{2k}^{\varepsilon_2}$, given by Theorems 3 and 4, in order to visualize this relationship. The selection of ε_1 and ε_2 can balance the trade-off between the size of the tree (equivalently, computation time) with the optimality guarantees of the algorithm. The bounds help us determine the extent of the trade-off.

2.4.4 Online Execution of the Search Tree

The techniques presented in the previous subsection allow us to find the optimal policy when looking ahead for T steps. While the pruning strategies reduce the size of the search tree, it may still be impractical to look ahead all the way till the end of longer episodes. In such cases, we can build a tree up to a shorter horizon T depending on the amount of computational time available for making each move.

Once the tree is built, the robot can execute the optimal policy. Note that this will, in general, not be the optimal policy since we are looking ahead only T steps and not until the end of the episode. At the root node, the robot executes the first control action along the optimal minimax path found. Then, the robot obtains a measurement. This measurement may not correspond to the worst-case measurement. Furthermore, the actual

Algorithm 1: The Minimax Search With Pruning.

```
function Minimax($node, depth, \alpha, \beta, state$)
    if node i is a leaf node then
        return $tr(\Sigma_T^i)$
    else if state is at the MAX level then
        $bestvalue \leftarrow -\infty$
        for each control input $u_{r1}(t), \cdots, u_{rn}(t)$ do
            $v \leftarrow$ New robot states
            $V \leftarrow$ 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
            // Check the condition in Theorem 2
            if $(C1)\&(C2)\&(C4)$ are true then
                break
            end
            return bestvalue
        end
    else
        $bestvalue \leftarrow +\infty$
        for each candidate measurements $z_1(t), \cdots, z_m(t)$ do
            $v \leftarrow$ Update estimated target states
            $V \leftarrow$ 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 (C1)&(C3)&(C5) are true then
                break
            end
            return bestvalue
        end
    end
end
$\{S_0\} \leftarrow$ Initial
Minimax($S_0, 1, -\infty, \infty$, MAX)
```

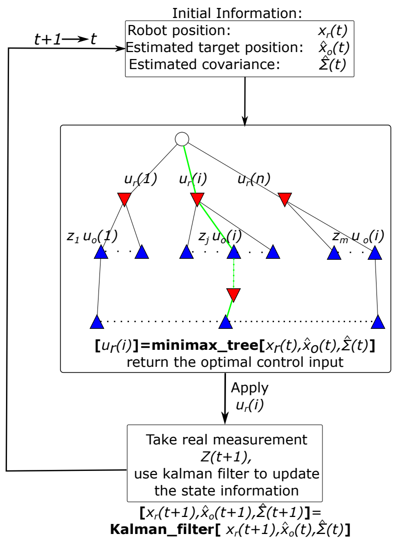

Initial Information:
Robot position: $x_r(t)$
Estimated target position: $\hat{x}_o(t)$
Estimated covariance: $\hat{\Sigma}(t)$
$t+1 \longrightarrow t$
$u_r(1)$
$u_r(i)$
$u_r(n)$
z_1 $u_o(1)$
z_j $u_o(i)$
z_m $u_o(i)$
$[u_r(i)]$=minimax_tree$[x_r(t),\hat{x}_o(t),\hat{\Sigma}(t)]$
return the optimal control input
Apply $u_r(i)$
Take real measurement $Z(t+1)$, use kalman filter to update the state information
$[x_r(t+1),\hat{x}_o(t+1),\hat{\Sigma}(t+1)]$=
Kalman_filter$[x_r(t+1),\hat{x}_o(t),\hat{\Sigma}(t)]$

Figure 2.7: Online execution of the minimax tree. At each time step, the robot executes the first control action given by the tree and obtains a measurement of the target. If the measurement $z(t+1)$ is close to one of the existing nodes, then that node is termed as the new root node. The tree can then extended to have a depth of $(k+1)$. If the new measurement $z(t+1)$ is not close to any nodes, then the tree can be rebuilt with the current estimate as the root node.

value of the measurement may not even be in the k candidate measurements in the tree. Therefore, the updated target estimate may not correspond to a node in the tree. Instead, we compute the node in the tree whose target estimate is closest to the actual one. We can use Bhattacharyya distance [16] to find the closest target estimate in the tree. The corresponding node then becomes the new root node of the tree. The optimal policy starting at that node is executed, iteratively. If the Bhattacharyya distance of the closest node is too large, we still can rebuild the whole minimax tree with the current target estimate as the root node. Figure 2.7 illustrates the process of building the search tree online.

When rebuilding the tree, solving the LMI given in Conditions (C4) and (C5) may be prohibitively slow. Instead, we can restrict the comparison to just a pair of nodes which takes significantly less time.

2.4.5 Trajectory Optimization

One of the assumptions we make is that the robot and the target have a set of finite, discrete control inputs. The size of the tree (without pruning) is exponential in the number of control actions. Therefore, the minimax tree approach is reasonable when the number of control inputs is limited. However, in practice, the set of control inputs available to the robot could be large, potentially infinite. In such cases, the proposed tree search algorithm will not scale.

In such cases, we can use the minimax search approach presented along with a trajectory optimization algorithm (e.g., [104]). Trajectory optimization methods take as input an initial trajectory and then refine it so as to improve the quality of the trajectory. We can use the path produced by the minimax search as an initial path that is further refined by trajectory optimization. For scalability, we can choose a small set of control inputs for the robot and the target to build the minimax search tree. Then, we can use a trajectory optimization method, e.g., Iterative Linear Quadratic Gaussian (iLQG) [104], that is not restricted to the smaller set of control inputs to refine the trajectory. In an online setting, we will execute only the first control input in the refined trajectory obtained. The process is then repeated after every time step as described in the previous subsection.

Different initial trajectories given to trajectory optimization lead to different output trajectories especially in environments with obstacles. The minimax search generates a better initial trajectory. Figure 2.8 provides an example that different initial trajectories will lead to different results. In Figure 2.8, two different initial collision-free trajectories are computed considering limited control inputs (up, down, left, right). In this example, to simplify the comparison, we consider the target's path is fixed. We see the effect of the initial trajectory given to the optimization routine.

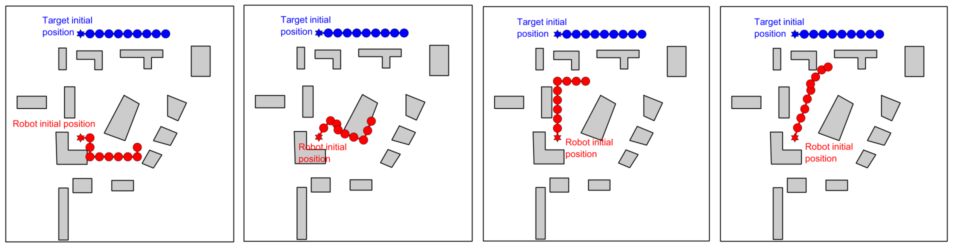
The trajectory optimization algorithm is allowed to pick control inputs for the robot and the target within a unit ball at every time step. That is, $||u_r(t)||_2 \leq 1$ and $||u_o(t)||_2 \leq 1$. However, any other suitable control input space can be used. Given the initial trajectory, we iteratively sample control inputs within a unit ball of the position of the robot and the target given in the initial trajectory, for each time step. The objective function for trajectory optimization is still Equation 2.5 (trace of the covariance in the final step). We alternate between trajectory optimization for the robot and the target. That is, we first optimize the trajectory for the robot assuming that the target's trajectory is fixed. We then fix the optimized robot's trajectory, and find the trajectory for the target. This process is then repeated for a fixed number of iterations or until a time deadline is met.

We follow similar steps in the previous subsection when we run the minimax with trajectory optimization algorithms online. The robot executes the first control action along the optimal trajectory. Then, the robot obtains a measurement and repeats the process if the measurement does not correspond to the worst-case measurement.

With an initial path generated by minimax, the trajectory optimization approach can improve the overall performance and compensate for the disadvantage of minimax that we only consider limited control inputs. We can use coarser discretization to get a faster trajectory first and then refine it with trajectory optimization. In this way, we can balance the trade-off between the accuracy and speed of the algorithm. We present more results in the simulation section.

2.5 Simulations

We carry out four types of evaluations via simulations. First, to show the efficacy of the minimax tree search in adversarial target tracking, we present qualitative and quantitative results. These results show the advantage of applying minimax over three baseline approaches. Second, we investigate the computational savings due to our algorithm by comparing the number of nodes in the pruned minimax tree and the full enumeration tree.



(a) Initial trajectory A. (b) Optimized trajectory A. (c) Initial trajectory B. (d) Optimized trajectory B.

Figure 2.8: A robot moves in a 2-D environment with obstacles. (a) and (c) are the initial collision-free trajectory computed by considering limited control input (up, down, left, right). (b) and (d) are the output of the trajectory optimization for ten times of iteration steps. The initial positions are marked in the hexagram. We set the initial trace of the covariance $tr(\Sigma_0) = 20$. With initial trajectory A, the trace of the covariance is 2.09 at the final step after applying trajectory optimization. With initial trajectory B, the trace of the covariance is 1.59 at the final step.

Then, we study the effect of varying the ε_1 and ε_2 parameters on the number of nodes. Finally, we use the control policy given by our algorithm and execute it by drawing actual measurements from a random distribution. This represents a realistic scenario where the measurements are not necessarily adversarial. We demonstrate how our strategy can be used in such a case, and compare the average-case with the worst-case performance.

2.5.1 Comparisons with Baseline Approaches

We compare the performance of the minimax tree search with three baseline approaches, as shown in Figure 2.9 and Figure 2.10, based on two environments in Figure 2.11. We assume that the robot can move faster than the target. The robot has the following control inputs, $\mathscr{U}_r$, to choose from,

$$\begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}-1\\0\end{bmatrix}, \begin{bmatrix}0\\-1\end{bmatrix}, \begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{bmatrix}, \begin{bmatrix}\frac{1}{\sqrt{2}}\\-\frac{1}{\sqrt{2}}\end{bmatrix}, \begin{bmatrix}-\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{bmatrix}, \begin{bmatrix}-\frac{1}{\sqrt{2}}\\-\frac{1}{\sqrt{2}}\end{bmatrix}.$$

The target moves slower and only has the following four control inputs,

$$\mathscr{U}_o = \left\{ \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}, \begin{bmatrix} -0.5 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -0.5 \end{bmatrix} \right\}$$

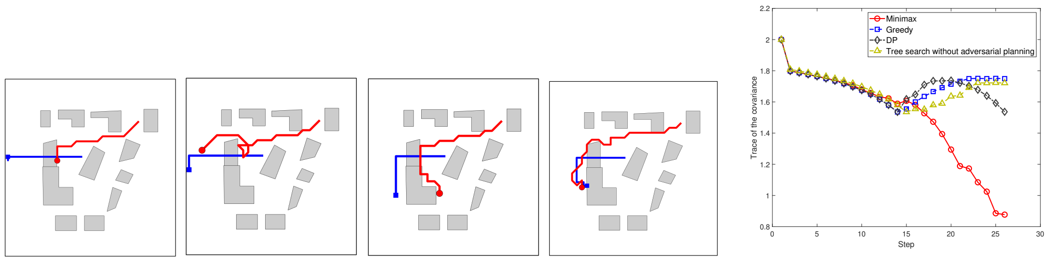
Even though the target moves slower than the robot, it can fly across the obstacles since we model it as an aerial robot. The robot, on the other hand, moves on the ground plane and therefore cannot pass through obstacles. Note that we assume the robot (ground robot) moves faster than the target (aerial robot) in the following experiments to demonstrate interesting behaviors. The tree search techniques also apply in other cases where the targets and the robots have the same mobility or when the target is faster than the robot.

We use three baselines: a greedy strategy [9], Dynamic Programming (DP) [4, 80], and tree search without adversarial nodes [5, 103]. Note that prior work on active target tracking has not considered the distance-dependent noise case. As such, there is no existing algorithm that we can compare against directly. We compare against three types of planners used for target tracking that are qualitatively different from the adversarial tree search planner that we employ.

The greedy strategy is to choose a control input that takes the robot closest to the mean of the target's estimate. The DP strategy maximizes $J(T)$ which is given by the following recurrence:

$$J(t) = \min_{u_r(t)} \left(\text{tr}(\hat{\Sigma}_t) + J(t-1) \right). \tag{2.22}$$

This objective function is the cumulative sum of the traces whereas in minimax we only optimize the final trace at the end of the planning horizon. The third baseline strategy is the same as minimax but without the MAX levels. That is, we will no longer consider the adversarial motion of the target (instead plan considering that the target is stationary).



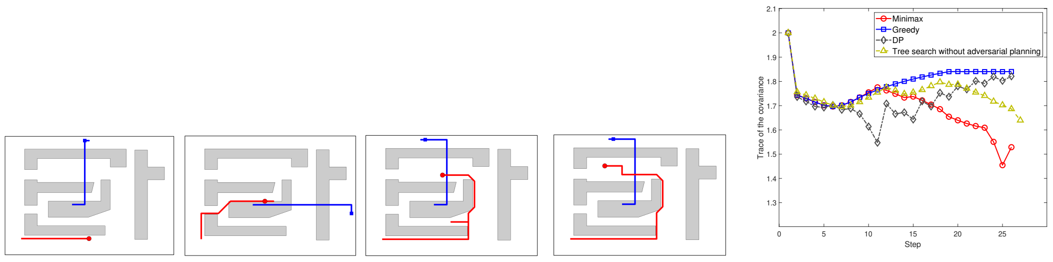
(a) Greedy (b) DP (c) Tree search without adversarial planning (d) Minimax (e) Trace of covariance matrix at each step.

Figure 2.9: Qualitative example 1: (a)–(d) provide the path for the robot (red) and the target (blue), given by greedy, DP, tree search without adversarial planning, and minimax. (e) is the comparison of the updated trace of covariance matrix $\text{tr}(\Sigma_t)$ at each step. The size of the environment is a 25×25. For DP and minimax, the planning horizon for each step is 5 (The height of the minimax tree is 11).

Atanasov et al. [5] used a similar tree search approach for tracking a non-adversarial target, assuming that the motion of the target is known. Here, since we do not know the motion of the target, we assume it is stationary, as a baseline comparison. The baseline strategies consider expected measurements when planning over the horizon. The evaluation is always done online for each of the four strategies: at each time step, we execute the action given by the planner, take an actual measurement, update the target's estimate, and replan.

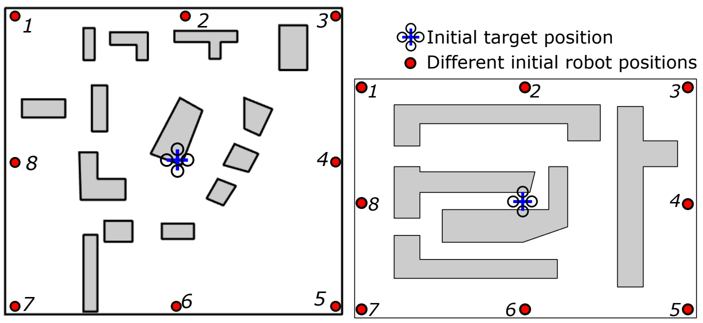
In general, minimax is a non-myopic planning algorithm and can plan on a longer horizon. If the target is actively escaping from the robot, minimax can predict the adversarial moves of the target. Thus minimax can better track the target than DP and tree search without adversarial planning. This is reflected in the two qualitative examples shown in Figures 2.9 and 2.10.

We also show how the trace of the covariance matrix $\text{tr}(\Sigma_t)$ evolves over time. The initial covariance matrices are $\Sigma_0 = I$. From Figure 2.9 and Figure 2.10, the greedy algorithm performs poorly since it is myopic and can be easily blocked by the obstacles. The DP has better tracking results than the greedy. However, in these two qualitative examples, the



(a) Greedy (b) DP (c) Tree search without adversarial planning (d) Minimax tree search (e) Trace of covariance matrix at each step in these four cases.

Figure 2.10: Qualitative example 2: (a)-(d) provide the path for the robot (red) and the target (blue), given by greedy, DP, tree search without adversarial planning, and minimax. (e) is the comparison of the updated trace of covariance matrix $\mathrm{tr}(\Sigma_t)$ at each step. The size of the environment is a 21×15.



(a) Environment A (25×25). (b) Environment B (21×15).

Figure 2.11: Environments used for the online simulations. Blue dots are the different initial positions for the robot.

target chooses the optimal path to escape. The minimax algorithm can better predict the target's movement than a tree search without adversarial planning and DP. In Figure 2.10(b), even using the non-myopic DP, the robot was stuck, with a planning horizon of five steps.

We also performed quantitative comparisons for the scenarios. We compare the tracking performance of the four approaches. For both environments, we vary the starting position of the robot (the target always start in the center). Table 2.2 shows the average of the trace of the covariance matrix after 50 time steps. Not surprisingly, all strategies perform better in environment A since it is more open than B. The greedy strategy performs the worst.

The minimax tree search outperforms the other baselines.

Table 2.2: Quantitative examples in different environments, each trial starts with different initial positions.

	Average final trace of the covariance matrix after 50 steps			
Environment	**Greedy**	**DP**	**Tree search without adversarial**	**Minimax tree search**
A	1.473	0.931	0.986	0.684
B	1.856	1.571	1.615	1.346

2.5.2 Comparing the Number of Nodes

In this section, the models used in the simulation are as follows. The robot follows a linear motion model and can choose from four actions at each time step, $X_r(t+1) = X_r(t) + u_r(t)$,

$$\mathscr{U}_r = \left\{ \begin{bmatrix} 1.0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1.0 \end{bmatrix}, \begin{bmatrix} -1.0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -1.0 \end{bmatrix} \right\} \tag{2.23}$$

We build the tree using five candidate measurements at each step: $z(t) = \{z_1(t), z_2(t), \cdots, z_5(t)\}$. The five values are randomly generated by drawing from a Gaussian distribution. The rest of the parameters are $C_t = I, H = I, \Sigma_v = I, \delta_1 = 0.5, \delta_2 = 0.1, \mathscr{B} = 1$. We assume the target is stationary in this section.

Just like Algebraic Redundancy in [107], (C4) and (C5) can be checked using an LMI solver. However, solving for an LMI is computationally expensive. A simpler method is to only check pairs of nodes. That is, when a new node A is generated, check node A and only one of the candidate nodes one by one. This results in lesser pruning but faster checks.

In our experiment, we check the candidate nodes one by one as follows. To check

(C1)–(C5) in Theorem 2 or Theorem 4, for each node in the search tree, we store its current level (depth in the search tree), current position X_r, the estimated target position $\hat{X}_o$, the covariance matrix, and a vector that stores the ancestors. We also need a list to store the nodes that are along the optimal minimax path (green path in Figure 2.6). For a newly generated node A, we do the following steps:

- For the nodes at the same level, find the nodes where the robot and target's estimated position are the same as node A (condition (C1)).
- For all the nodes we found, only keep the nodes along the optimal minimax path and then find their common ancestors with node A (conditions (C2) and (C3)).
- Check condition (C4) or (C5) based on the type of the common ancestor (MIN or MAX).
- If (C4) or (C5) is true, mark node A as being pruned and not explore its children nodes.

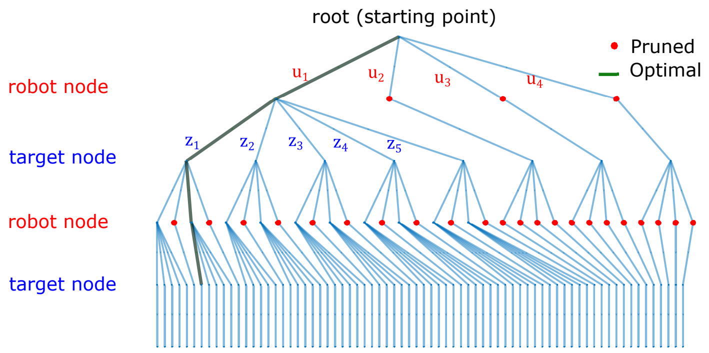


Figure 2.12: A five-level minimax tree with pruning (189 nodes). Full enumeration has 505 nodes.

Figure 2.12 shows an example of a five-level minimax tree with pruning. Figure 2.13 shows the number of nodes in the minimax tree after pruning and the number of nodes in a full enumeration tree, respectively. We prune a node by comparing it to the nodes already explored. More nodes will be pruned if initial nodes encountered are “close” to the optimal policy. For instance, if the first set of nodes explored happen to be the optimal control law that drives the robot close to the target, then we expect the nodes encountered later will be pruned earlier in the process. To provide a fair assessment, we generate the search trees for various true positions of the target. Figure 2.13 shows the average and standard deviation of the number of nodes.

Figure 2.13 shows that our algorithm prunes orders of magnitudes of nodes from the full enumeration tree. For a tree with depth 13, there are 8.08×10^7 nodes in the full tree but the same optimal solution can be computed using 4.36×10^5 nodes with our pruning strategy.

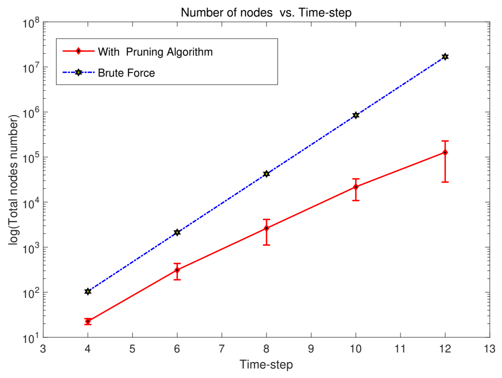


Figure 2.13: Comparison of the number of total nodes generated for minimax tree. Note that the y axis is log scale.

2.5.3 Comparing the Sub-optimal Pruning algorithm

By sacrificing optimality, we can prune even more nodes. We evaluate this by varying ε_1 and ε_2 individually first, and then jointly. As shown in Figure 2.14, ε_1-alpha pruning is relatively better at reducing the size of the minimax tree. This is intuitive because ε_1-alpha pruning condition compares nearly every pair of nodes at the same depth. ε_2-algebraic redundancy pruning, on the other hand, requires more conditions to be satisfied.

Nevertheless, Figure 2.14 shows that by sacrificing optimality, the number of nodes can be substantially reduced.

We also study the effect that varying ε_1 and ε_2 has on the optimality. Figure 2.15 plots the upper and lower bounds for the trace of the covariance in the optimal case ($J^*_{2k} = tr(\hat{\Sigma}^*_{2k})$) as well as in the suboptimal cases ($J^{\varepsilon_1}_{2k}$ and $J^{\varepsilon_2}_{2k}$). The lower bound of the optimal value J^*_{2k} is obtained by applying the Kalman Riccati equation k times to the initial covariance matrix Σ_0 assuming that the distance $d(X_r(t), X_o(t)) = 0$ for all t. This corresponds to the case when the variance of the noise is minimum. The upper bound is obtained by considering the worst case noise which occurs when $||X_r(t) - X_o(t)||_2 = \mathscr{B}$. The solution of a minimax search tree, J^*_{2k}, lies between the upper and lower bounds (i.e., between the two blue curves).

From the upper bound of J^*_{2k}, we can plot the upper bounds for $J^{\varepsilon_1}_{2k}$ and $J^{\varepsilon_2}_{2k}$. The upper bound for $J^{\varepsilon_1}_{2k}$ is given by Theorem 3 whereas that for $J^{\varepsilon_2}_{2k}$ is given by Theorem 4. For the latter, we use the worst-case measurements to compute the recursive term. We use $\varepsilon_1 = 0.1$, $\varepsilon_2 = 0.2$ and $\mathscr{B} = 1$ to plot the figure. When we have $\varepsilon_1 = 0.1$ and $\varepsilon_2 = 0.2$ together, the bound will be the maximum of the upper bounds for $J^{\varepsilon_1}_{2k}$ and $J^{\varepsilon_2}_{2k}$.

Table 2.3 shows the error in tracking the target target caused by sacrificing optimality (i.e., non-zero ε_1 and ε_2). In this experiment, the robot starts at $(0,0)$ and the target's initial position is $(1,0)$. We run the policy for $k = 7$ time steps. We assume the robot and target can move with the same speed, $\mathscr{U}_r = \mathscr{U}_o = 1$. The initial distance between the robot and the target is 1. Table 2.3 shows the average distance (of 50 trials) after k time steps using a suboptimal policy with various values of ε_1 and ε_2. The table shows that the average separation increases as ε_1 and ε_2 increase, as expected. We also observe that ε_1 has a larger effect on the tracking error as compared to ε_2.

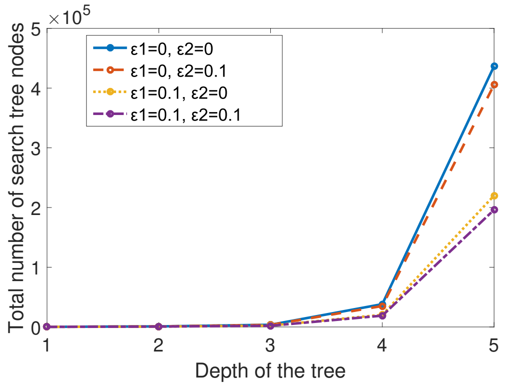


Figure 2.14: Effect of the ε_1 and ε_2 relaxation parameters on the number of nodes in the search tree. The baseline case is the optimal solution with alpha pruning and algebraic redundancy with both parameters set to zero.

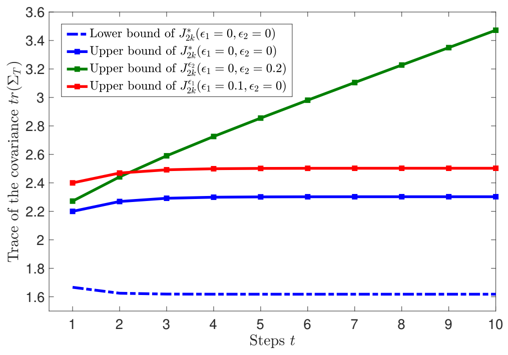


Figure 2.15: Applying Thereom 3 and Thereom 4. The range of $J^*_{2k} = tr(\hat{\Sigma}^*_{2k})$, $J^{\varepsilon_1}_{2k}, J^{\varepsilon_2}_{2k}$ with planning horizon $k = 1, 2, \cdots, 10$. Initial value $\Sigma_0 = I$, $\delta_1 = 0.5$, $\delta_2 = 0.1$, $\mathscr{B} = 1$.

Table 2.3: Effect of the ε_1 and ε_2 relaxation parameters on the tracking error.

	$\varepsilon_1 = 0$, $\varepsilon_2 = 0$	$\varepsilon_1 = 0$, $\varepsilon_2 = 0.2$	$\varepsilon_1 = 0.2$, $\varepsilon_2 = 0$	$\varepsilon_1 = 0.2$, $\varepsilon_2 = 0.2$	$\varepsilon_1 = 0.3$, $\varepsilon_2 = 0.3$
Average distance in 50 times	1.00	1.36	1.54	1.70	2.26

2.5.4 Trajectory Optimization

In this subsection, we compare the performance of executing the controls found by minimax only and minimax with trajectory optimization. We use the environment in Figure 2.11. We use minimax to generate the initial trajectory by using only four control inputs for both the robot and the target (given in Equation 16). Then we use the same objective function in Equation 2.5 (trace of the covariance in the final step). The trajectory optimization is allowed to refine the trajectory subject to $||u_r(t)||_2 \leq 1$ and $||u_o(t)||_2 \leq 1$. Unlike the previous simulations, we assume the target cannot go over the obstacles to show the effect of trajectory refinement.

We use this in an online setting as described in Section 2.4.5. We compare two cases: minimax without trajectory optimization and minimax with trajectory optimization.. In both cases, the target uses minimax with trajectory optimization (even if the robot is not). This is because in practice the target is not restricted to use a minimax tree or subject to only follow the four control inputs we use to build the tree. The goal of this experiment is to show how much improvement we can get with trajectory optimization in terms of the tracking performance.

A qualitative example is shown in Figure 2.16. In Figure 2.16-(a), we see the two trajectories followed by the robot using minimax only (left) and minimax with trajectory optimization (right). We observe that the robot is able to get closer to the target, follow a smoother trajectory, as well as improve the performance in tracking. This can be seen in Figure 2.16-(b) which shows the improvement of the trace of the covariance.

We also performed quantitative comparisons. For the two environments in Figure 2.11, we vary the starting position of the robot (the target always start in the center). Table 2.4 shows the mean and the standard deviation of the trace of the covariance matrix after 25 time steps. The trajectory optimization approach can always improve the path found by the minimax tree. We observe that the improvement is larger in environment B. We suspect that this is because environment B has longer, narrower corridors where if a robot goes down an incorrect path it does not get easy opportunities to correct it. On the other hand, environment A has smaller obstacles that the robot can go around. Therefore, the trajectory optimization results in larger improvement in the more challenging environment (B).

Table 2.4: Quantitative examples for minimax only vs minimax with trajectory optimization.

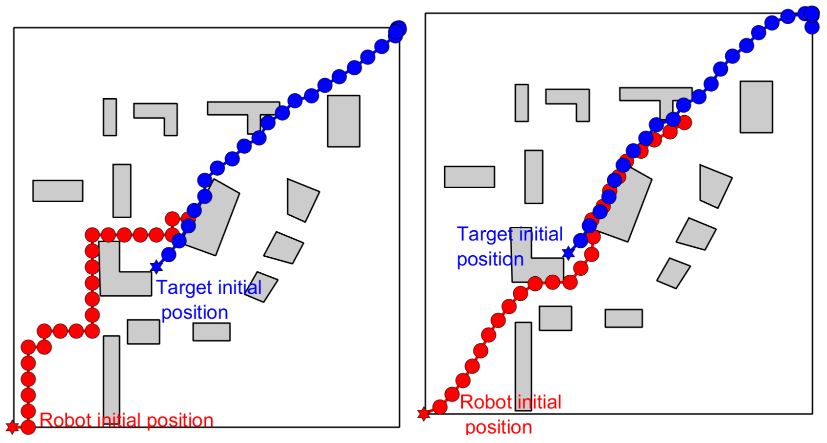
	Final trace of the covariance matrix after 25 steps mean (standard deviation)	
Environment	**Minimax only**	**Minimax with optimization**
A	0.97 (0.16)	0.73 (0.13)
B	1.87 (0.29)	1.12 (0.25)

2.6 Experiments

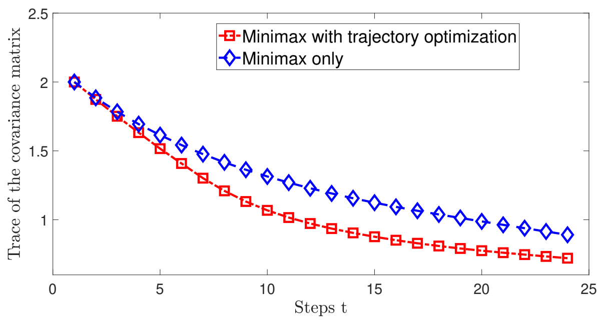
We implemented the worst-case minimax tracking algorithm using indoor and outdoor robots. We use a five-level minimax tree with a look-ahead of two steps. Our experiments show that the algorithm can be successfully implemented and executed on real hardware.[7]

For the indoor experiment, we used two Pioneer 3DX robots (Figure 2.17) equipped with a 2.6GHz i7-6700HQ processor and 16 GB RAM to find the optimal minimax strategy. We use MATLAB 2015b to execute the proposed algorithm and send the control inputs to the Pioneer 3DX robot through MATLAB ROS toolbox. One Pioneer acts as the target and the other acts as the tracking robot. The motion and measurement models and parameters are similar to the Gazebo simulation reported in the previous section. The measurement noise is generated using parameters $\delta_1 = 0.5$ and $\delta_2 = 0.1$ from the true position of the target robot. The robot's speed is $0.4m/s$ and the target's speed is $0.25m/s$.

[7]The video can be found at `https://youtu.be/ATh_Vv3pgS4`.



(a) Online adversarial planing: Minimax only vs minimax with trajectory optimization.



(b) The trace of the covariance at each step.

Figure 2.16: Qualitative online path planning examples for minimax with trajectory optimization: (a) provides the path for the robot (red) and the target (blue), given with/without trajectory optimization. The initial positions are marked by a hexagram. (b) show the comparison of the trace of covariance matrix tr(Σ_t) at each step. Note that this is an online execution where the robot replans at every timestep. This is why the optimized trajectory deviates from the minimax only trajectory in terms of going around the obstacle.

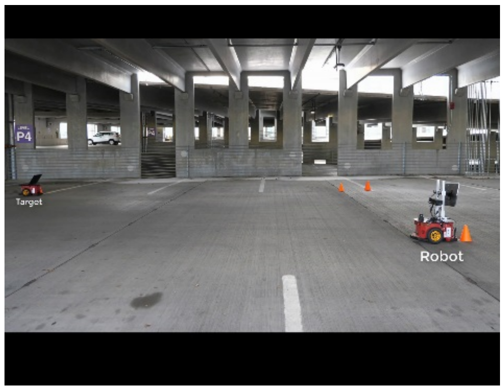
We use minimax with optimal pruning conditions in real-world experiments. The results presented in this section are found by minimax only, without trajectory optimization since the environment does not contain any obstacles and our focus was on studying the online performance of minimax.

The robot only takes a new measurement and computes the next control input after it finishes the previous movement. In this experiment, we use a 5 level minimax tree (look ahead for two steps) to compute the strategy. It takes on an average (of 10 trials) $1.36s$ to compute the policy with a 5 level minimax tree. The computation time may vary based on the specific instance and the parameters.

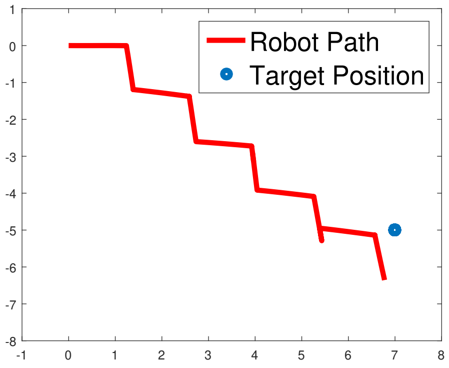
We carried out three sets of experiments: (1) tracking a stationary target; (2) tracking a target that moves in a straight line; and (3) tracking a target that actively chooses adversarial control inputs to evade the tracker. Figure 2.17 shows the robot and the target's trajectories for the three experiments. In all cases, we see that the minimax algorithm with pruning drives the robot towards the target. Furthermore, the hardware experiments demonstrate that minimax algorithm can be applied in real-time on actual hardware[8].

The outdoor experiment consisted of an Unmanned Aerial Vehicle (UAV) tracking a stationary target using the minimax tree. The UAV uses a DJI Flame Wheel F450 frame and ArduPilot Mega (APM) firmware running on a Pixhawk autopilot. The on-board computer interfaces with the autopilot using the mavros package of the Robotic Operating System (ROS). The same computer used for the indoor experiment runs the minimax search algorithm and sends the waypoints to the autopilot. The experiments were conducted in a farmland near Blacksburg, VA, USA. The measurement is obtained by a noisy GPS sensor placed on the target. In order to generate the tree, the GPS noise is modeled as a zero mean Gaussian noise with constant variance ($\delta_1 = 0.5$ and $\delta_2 = 0.1$). The starting position of the UAV was about 160 meters from the target.

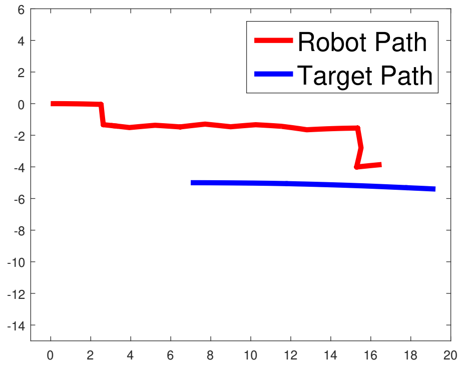
[8]The video can be found at `https://youtu.be/ATh_Vv3pgS4`



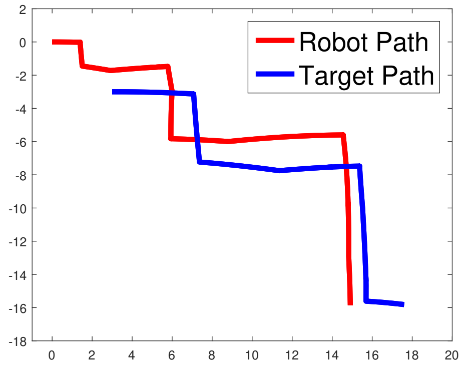
(a) Experimental environment



(b) Tracking a stationary target

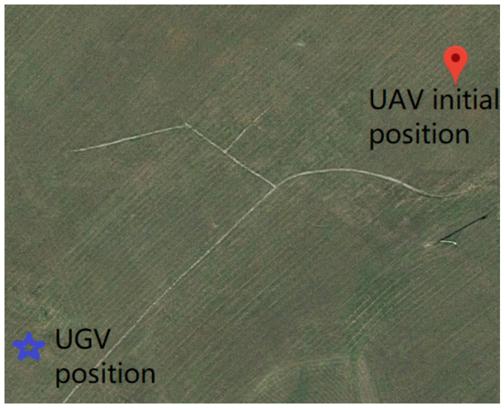


(c) Tracking a moving target

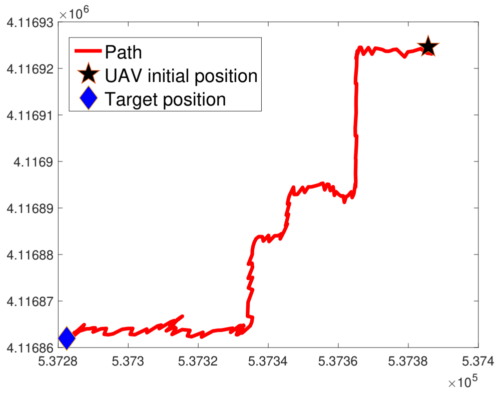


(d) Tracking an adversarial target

Figure 2.17: Real-world indoor tracking experiments.



(a) Experimental environment



(b) UAV online tracking

Figure 2.18: Real-world outdoor tracking experiment.

The UAV took off manually and then switched to the autonomous mode, where it follows the control commands given by the minimax tree. Figure 2.18 shows the resultant trajectory of the UAV produced by the algorithm. Similar to the indoor experiment, the UAV has four motion control input (forward, backward, left, right). The unit step of the UAV is set as 10 meters.

The indoor and the outdoor experiments demonstrate that the online minimax tracking algorithm along with the pruning strategy can be applied in real-time on actual hardware[9].

2.7 Summary

We investigated the problem of devising closed-loop control policies for target tracking with distance-dependent measurement noise. Unlike the state-independent noise case, the value of a candidate control law in our version is a function of the history of measurements obtained. Consequently, planning over a horizon requires taking into account all possible measurement values. We focused on minimizing worst-case uncertainty. Our solution consists of building a minimax search tree to obtain the control policy. A full enumeration tree has a size that is exponential in the number of measurements, control actions, and the planning horizon. Instead, we exploited the structural properties of Kalman filter to yield a tree with significantly less number of possible nodes without sacrificing the optimality guarantees. We also showed how two parameters, ε_1 and ε_2, can be used to yield even more computational savings at the expense of optimality. The resulting algorithm was evaluated in simulations and through real-world experiments.

One disadvantage of the generalization is that we have to discretize the set of possible future measurements. Our immediate future work is to bound the suboptimality as a function of the number of discrete samples chosen to represent the continuous set of measure-

[9]The video can be found at `https://youtu.be/ATh_Vv3pgS4`.

ments. The algebraic redundancy conditions require the states to be identical which is reasonable when operating in a discrete setting. A useful extension would be to group together nearby states, and quantify the effect of such grouping, so that we can allow for even more pruning and/or extend to the continuous regime. Another avenue of future work focuses on extending these results to multi-robot, multi-target scenarios. Our prior work [94, 101] has shown a greedy assignment of robot trajectories to targets yield provably approximate solutions for one-step planning. We will extend this to plan over a finite horizon using the results presented in this chapter.

Chapter 3: A False Data Injection Strategy to Mislead Kalman filter

In this chapter, we will continue investigating the target tracking problem. However, we will now consider an asymmetric setting. The asymmetry will be due to the fact that the adversary has the additional capability of injecting malicious signals into the measurements obtained by the robot. This chapter will focus on how much this asymmetry will affect the robot's estimation results.

3.1 Introduction

As autonomous systems proliferate, there are growing concerns about their security and safety [76, 97]. Of particular concern is their vulnerability to signal spoofing attacks [99]. As a result, many researchers are designing algorithms that enable an *observer* to detect and mitigate signal spoofing attacks (e.g., [3, 29, 34, 39, 113]). We study the problem from the opposite (i.e., the attacker's) point-of-view. Our goal is to characterize the capabilities of the attacker that is generating the spoofing signals while assuming that the observer is using a Kalman filter for state estimation.

The problem of generating spoofing attacks has been studied specifically for GPS signals. Tippenhauer et al. [99] describe the requirements as well as present a methodology for generating spoofed GPS signals. Larcom and Liu [60] presented a taxonomy of GPS spoofing attacks.

The typical approach to mitigate sensor spoofing attacks is by designing robust state estimators [14]. Fawzi et al. [35] presented the design of a state estimator for a linear dynamical system when some of the sensor measurements are corrupted by an adversarial attacker. We focus on the scenario where the observer uses a Kalman Filter (KF) for estimating the state using measurements that are corrupted by additive spoofing signals by the attackers. We study the problem of generating spoofing signals of minimum energy that can achieve any desired separation between the KF estimate with spoofing and without spoofing. We show that for many practical cases, the spoofing signals can be generated using linear programming in polynomial time.

Many recent works have undertaken research on how to spoof estimators such as LQG control system [70], GPS system [91], wireless sensor networks [69] and electric power grids [67]. In [67], the authors present false data injection attack against state estimation in electric power grids. They show how an attacker can exploit the configuration of a power system to launch such attacks to successfully introduce arbitrary errors into certain state variables while bypassing existing techniques for bad measurement detection.

Another work by Su et al. [91] is closely related to ours. The authors show how to spoof the GPS signal without triggering a detector that uses the residual in the Kalman filter. They present a 1-step (greedy) online spoofing strategy that solves a linear relaxation of a Quadratically Constrained Quadratic Program (QCQP) at each timestep. We present a strategy that plans for T future timesteps, instead of just the next timestep, while minimizing the spoofing signal energy. Furthermore, we characterize the scenarios under which our strategy finds the optimal solution in polynomial time.

The work that is most closely related to ours is by Mo et al. [69, 70]. Their goal is to design false measurement data to mislead a system with a Kalman filter [69] or an LQG control system [70]. Both our work and the aforementioned work assume that the system is linear with Gaussian noise and that a discrete Kalman filter is used to estimate the state. Mo et al. [70] define (ε, α)–attacks (see Definitions 2 and 3 in the original paper). Based on their definition, an attack sequence is successful if: (1) the difference in the estimated state of the system under attack and the true state is greater than a given value; and (2) the probability of alarm for the χ^2 failure detector is always smaller than a given threshold. They proved that a linear control system is perfectly attackable if and only if its transition matrix has an unstable eigenvalue and the corresponding eigenvector satisfies additional conditions (c.f. Theorem 2 in [70]). These conditions may be too strict. We relax the requirements of an attackable system with the goal of being applicable to more classes of systems. Specifically, we remove the second condition in the (ε, α)–attacks and instead we only consider minimizing the total injected signal by $\sum_{t=1}^{T} \gamma_t \cdot \|\varepsilon_t\|_p^p$. Nevertheless, we also show the conditions under which an attack is successful against the χ^2 detector.

Bai et al. presented a different notion of a successful attack in two relevant papers [6, 7]. They define a successful attack as ε–stealthy.[1] The goal is to maximize the mean squared error between the attacker's estimated state and the true state subject to $\lim_{t\to\infty} \frac{1}{t} D(\tilde{r}_1^t || r_1^t) < \varepsilon$. Here, $D(\tilde{r}_1^t || r_1^t)$ is the Kullback-Leibler divergence (KLD) between the the Kalman filter innovation without attack, r_1^t, and with attack, $\tilde{r}_1^t$. Their notion of a successful attack only applies when $t \to \infty$. Thus, their attack strategy can only be applied when a Kalman filter runs for a long time. Instead, we focus on a finite, possibly small, number of time steps and do not require $t \to \infty$. Furthermore, our notion of a successful attack differs from theirs and does not focus on a specific type of detector.

Various failure detectors have been proposed in the literature. Jones [46] presented one of the first work on failure detection in linear systems. Specifically, Jones [46] presented a linear filter that increases the sensitivity of the residual of the filter, which helps to improve the detection of a particular failure. Brumback et al. [21] presented a χ^2 test for fault detection in Kalman filters. Mo et al. [69] studied the effect of false data injection attacks on state estimation with a χ^2 failure detectors.

[1]This ε is not related to the ε used by Mo et al. [69, 70].

In this chapter, we study how to design spoofing signals that are agnostic to the failure detector. Instead, we minimize the magnitude of the injected signals while still ensuring the desired separation in the filter output. We provide numerical simulations to show our strategy successfully misleads the χ^2 detector.

Based on the motion model of the target and the evolution of the KF, three problems for spoofing design are formulated in Section 3.2. Section 3.3 shows the approaches to solve these optimization problems. The simulations for verifying spoofing strategies are given in Section 3.4. Section 3.5 provides a numerical example to illustrate how the proposed spoofing strategy can be applied to a system equipped with a failure detector. Finally, Section 3.6 summarizes the conclusion and future work.

3.2 Problem Formulation

Notation: We denote the set of positive real number by $\mathbb{R}^+$, the set of positive integers by $\mathbb{Z}^+$. The set of real vectors with dimension n is denoted by $\mathbb{R}^n$, $n \in \mathbb{Z}^+$, and the set of real matrices with m rows and n columns by $\mathbb{R}^{m\times n}$, $m,n \in \mathbb{Z}^+$. We write $||\cdot||_p^p$, $p \in \mathbb{Z}^+$ as the p^{th} power of L_p vector norm, $\mathbb{E}(\cdot)$ as the expectation of a random variable, I_n as the identity matrix with size n, $n \in \mathbb{Z}^+$, and $\mathscr{N}(\mu,\sigma^2)$ as the normal distribution with mean μ and variance σ^2.

We consider a scenario where an observer estimates the location of a target using a KF in an n–dimensional space. The target misleads the observer by adding spoofing signals to the observer's measurement. We define the target's model as:

$$x_{t+1} = \mathscr{F}x_t + \mathscr{G}u_t + \omega_t, \tag{3.1}$$

where $\mathscr{F}, \mathscr{G} \in \mathbb{R}^{n\times n}$, $x_t \in \mathbb{R}^n$ is the state of target, $u_t \in \mathbb{R}^n$ is the control input, $w_t \sim \mathscr{N}(0,R)$

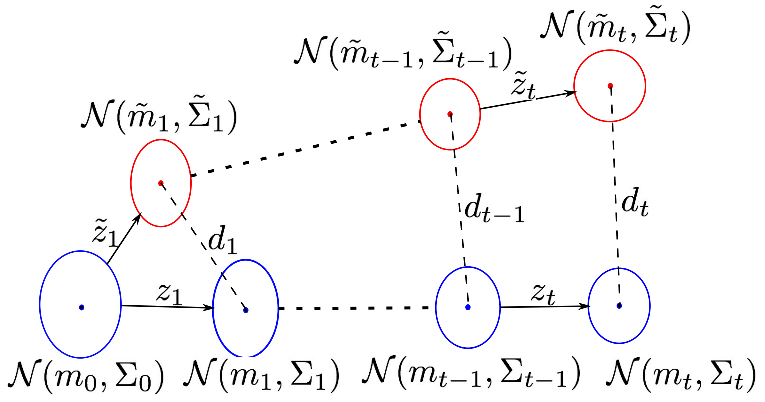

$\mathcal{N}(\tilde{m}_t, \tilde{\Sigma}_t)$
$\mathcal{N}(\tilde{m}_{t-1}, \tilde{\Sigma}_{t-1})$
$\tilde{z}_t$
$\mathcal{N}(\tilde{m}_1, \tilde{\Sigma}_1)$
d_{t-1}
d_t
$\tilde{z}_1$
d_1
z_1
z_t
$\mathcal{N}(m_0, \Sigma_0)$
$\mathcal{N}(m_1, \Sigma_1)$
$\mathcal{N}(m_{t-1}, \Sigma_{t-1})$
$\mathcal{N}(m_t, \Sigma_t)$

Figure 3.1: The evolution of KF estimate by applying z_t and $\tilde{z}_t$, respectively. Note that m_t and $\tilde{m}_t$ may also be different initially, i.e., ($\tilde{m}_0 \neq m_0$).

is the Gaussian process noise with $R \in \mathbb{R}^{n \times n}$.

The observer estimates the target's measurement using a linear measurement model:

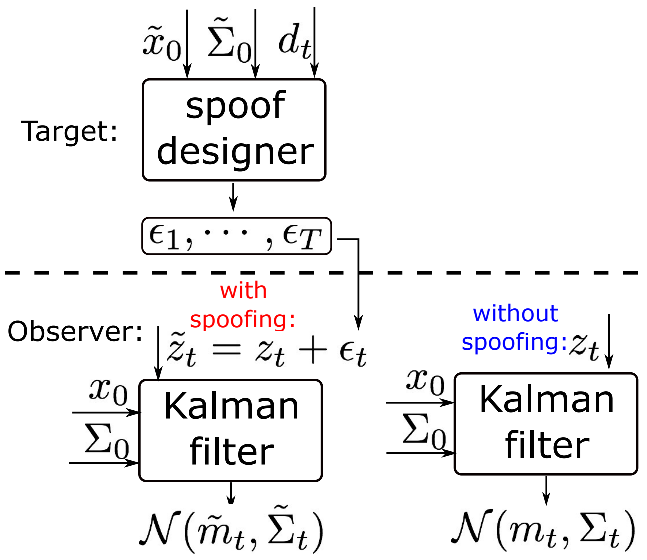
$$z_t = \mathscr{H} x_t + v_t, \tag{3.2}$$

where $\mathscr{H} \in \mathbb{R}^{n \times n}$ and $v_t \sim \mathscr{N}(0, Q)$ gives the measurement noise with $Q \in \mathbb{R}^{n \times n}$.

In order to mislead the observer, the target corrupts the observer's measurement by adding spoofing signal to mislead the observer's estimate. We assume the measurement received by the observer is $\tilde{z}_t \in \mathbb{R}^n$ with spoofing signal (Equation (3.3)) instead of the true measurement $z_t \in \mathbb{R}^n$ without spoofing signal (Equation (3.2)). The spoofing signal $\varepsilon_t := [\varepsilon_{t1}, \cdots \varepsilon_{tn}]^T \in \mathbb{R}^n$ adds additional measurement error:

$$\tilde{z}_t = z_t + \varepsilon_t. \tag{3.3}$$

The observer uses a KF to estimate the target's state with initial distribution $\mathscr{N}(m_0, \Sigma_0)$. Since it receives the spoofing measurement $\tilde{z}_t$ for updating, we denote distributions generated by the evolution of its KF as $\mathscr{N}(\tilde{m}_t, \tilde{\Sigma}_t)$ when step $t \geq 1, t \in \mathbb{Z}^+$. We also denote the distributions generated by the evolution of a KF using true measurement z_t as $\mathscr{N}(m_t, \Sigma_t)$.



$\tilde{x}_0$
$\tilde{\Sigma}_0$
d_t
Target:
spoof designer
$\epsilon_1, \cdots, \epsilon_T$
Observer:
with spoofing:
$\tilde{z}_t = z_t + \epsilon_t$
without spoofing: z_t
x_0
Σ_0
Kalman filter
$\mathcal{N}(\tilde{m}_t, \tilde{\Sigma}_t)$
$\mathcal{N}(m_t, \Sigma_t)$

Figure 3.2: Signal spoofing process and its effect on the observer's KF estimation.

The goal for the target is to set the separation between the mean estimate m_t and $\tilde{m}_t$. The target's goal is to achieve some desired separation, $d_t \geq 0$, for each step t within the planning horizon (Figure 3.1). Figure 3.2 shows the target's spoofing process where it uses the initial guess of $\mathcal{N}(m_0, \Sigma_0)$ denoted as $\mathcal{N}(\tilde{m}_0, \tilde{\Sigma}_0)$ and desired separation d_t to design spoofing signal ε_t. In order to avoid detection, the targets seeks to minimize the magnitude of the spoofing signal.

Note that, although we use the example of tracking a moving target, the state x_t can be more general. For example, it can represent the state of a power system [67], the state of a networked system[69], or the state of a GPS device [60].

We first propose two problems for offline scenarios as follows.

3.2.1 Offline Spoofing Signal Design with Known $\mathcal{N}(m_0, \Sigma_0)$

If the target knows $\mathcal{N}(m_0, \Sigma_0)$ of the KF, then the target can set $\mathcal{N}(\tilde{m}_0, \tilde{\Sigma}_0)$ equal to $\mathcal{N}(m_0, \Sigma_0)$.

Problem 2 (Offline with Known $\mathcal{N}(m_0, \Sigma_0)$)**.** *Consider a target with motion model (Equation* (3.1)*), measurement model (Equation* (3.2)*), and spoofing measurement model (Equation* (3.3)*). Assume the target knows $\mathcal{N}(m_0, \Sigma_0)$. Find a sequence of spoofing signal inputs, $\{\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_T\}$ to achieve desired separation d_t between $\tilde{m}_t$ and m_t at step t to*

$$\textit{minimize} \quad \sum_{t=1}^{T} \gamma_t \cdot \|\varepsilon_t\|_p^p$$

subject to

$$\|m_t - \tilde{m}_t\|_p^p \geq d_t^p, \quad \forall t \tag{3.4}$$

where $\gamma_t \in \mathbb{R}^+$ is a weighting parameter and $T \in \mathbb{Z}^+$ is the optimization horizon.

3.2.2 Offline Spoofing Signal Design with Unknown $\mathcal{N}(m_0, \Sigma_0)$

Next, we consider the case where the target does not know the initial condition in the KF. Instead, we assume that the initial estimate $\tilde{m}_0$ is not too far away from m_0 (in exception).

Problem 3 (Offline with Unknown $\mathcal{N}(m_0, \Sigma_0)$)**.** *Consider a target with motion model (Equation* (3.1)*), measurement model (Equation* (3.2)*), and spoofing measurement model (Equation* (3.3)*). Assume the target starts spoofing with $\tilde{m}_0$, where $\mathbb{E}(m_0 - \tilde{m}_0) = M_0$ and $\tilde{\Sigma}_0 \neq \Sigma_0$. Find a sequence of spoofing signal inputs, $\{\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_T\}$, so that the expected value of the separation ($\|\mathbb{E}(m_t - \tilde{m}_t)\|$) can achieve a desired value d_t at each step t,*

$$\textit{minimize} \quad \sum_{t=1}^{T} \gamma_t \cdot \|\varepsilon_t\|_p^p$$

subject to

$$||\mathbb{E}(m_t - \tilde{m}_t)||_p^p \geq d_t^p, \quad \forall t \tag{3.5}$$

where $\gamma_t \in \mathbb{R}^+$ is a weighing parameter and $T \in \mathbb{Z}^+$ is the optimization horizon.

3.3 Signal Spoofing Strategies

In this section, we show how to solve Problems 2 and 3 when $p = 1$ and $p = 2$. We first present the relationship between the separation $m_t - \tilde{m}_t$ and the initial bias $m_0 - \tilde{m}_0$.

Theorem 5. *Consider a target with motion model (Equation* (3.1)*), measurement model (Equation* (3.2)*), and spoofing measurement model (Equation* (3.3)*). The evolutions of the KFs by applying z_t and $\tilde{z}_t$ give the distributions $\mathscr{N}(m_t, \Sigma_t)$ and $\mathscr{N}(\tilde{m}_t, \tilde{\Sigma}_t)$, respectively. The difference, $m_t - \tilde{m}_t$ is,*

$$\begin{aligned} m_t - \tilde{m}_t = &\prod_{i=0}^{t-1} A_{t-i} \cdot (m_0 - \tilde{m}_0) + \\ &\sum_{i=0}^{t-2} \left(\prod_{j=0}^{i} A_{t-j} (B_{t-1-i} + C_{t-1-i}) \right) + B_t + C_t, \end{aligned} \tag{3.6}$$

where $A_t = \mathscr{F} - \tilde{K}_t \mathscr{H} \mathscr{F}, \quad B_t = (K_t - \tilde{K}_t)\left[z_t - \mathscr{H}(\mathscr{F} m_{t-1} + \mathscr{G} u_{t-1})\right], \quad C_t = -\tilde{K}_t \varepsilon_t.$

Before we prove Theorem 5, we can review the Kalman Filter update equations from equation (3.21), (3.18), (3.19), (3.20).

According to the Kalman gain update Equation (3.20), the evolution covariance matrix at step t, Σ_t, only depends on the state model parameters and the initial condition of the covariance matrix Σ_0. The Kalman gain at step t, K_t depends on the covariance matrix Σ_t. Both Σ_t and K_t do not depend on the control input series $\{u_t\}_{t=1,\cdots,k}$, measurement

$\{z_t\}_{t=1,\cdots,k}$. Thus, the covariance matrix and the Kalman gain can be predicted from the KF covariance update steps.

$$\begin{aligned}\Sigma_{t+1|t} &= \mathscr{F}\Sigma_{t|t}\mathscr{F}' + R_t,\\ \Sigma_{t+1|t+1} &= (I - K_t\mathscr{H})\Sigma_{t+1|t}.\end{aligned} \tag{3.7}$$

From Equation (3.7), the Kalman gain can be predicted from the initial condition Σ_0.

We now prove our main result.

Proof. From the update of KF, we have

$$\begin{aligned}m_t &= m_{t|t-1} + K_t(z_t - \mathscr{H}m_{t|t-1})\\ &= (I - K_t\mathscr{H})m_{t|t-1} + K_t z_t\\ &= (I - K_t\mathscr{H})(\mathscr{F}m_{t-1} + \mathscr{G}u_{t-1}) + K_t z_t.\end{aligned} \tag{3.8}$$

and

$$\tilde{m}_t = (I - \tilde{K}_t\mathscr{H})(\mathscr{F}\tilde{m}_{t-1} + \mathscr{G}u_{t-1}) + \tilde{K}_t(z_t + \varepsilon_t).$$

Recursively,

$$\begin{aligned}&m_t - \tilde{m}_t\\ =&(I - K_t\mathscr{H})(\mathscr{F}m_{t-1} + \mathscr{G}u_{t-1}) + K_t z_t - [(I - \tilde{K}_t\mathscr{H})(\mathscr{F}\tilde{m}_{t-1} + \mathscr{G}u_{t-1}) + \tilde{K}_t(z_t + \varepsilon_t)]\\ =&(\mathscr{F} - K_t\mathscr{H}\mathscr{F})m_{t-1} - (\mathscr{F} - \tilde{K}_t\mathscr{H}\mathscr{F})\tilde{m}_{t-1} - (K_t - \tilde{K}_t)\mathscr{H}\mathscr{G}u_{t-1} + [K_t z_t - \tilde{K}_t(z_t + \varepsilon_t)],\end{aligned} \tag{3.9}$$

subtract a term $\tilde{K}_t\mathscr{H}\mathscr{F}m_{t-1}$ then add the same term,

$$
\begin{aligned}
& m_t - \tilde{m}_t \\
=& (\mathscr{F} - \tilde{K}_t\mathscr{H}\mathscr{F})m_{t-1} - (\mathscr{F} - \tilde{K}_t\mathscr{H}\mathscr{F})\tilde{m}_{t-1} \\
& - (K_t - \tilde{K}_t)\mathscr{H}\mathscr{G}u_{t-1} + (K_t - \tilde{K}_t)z_t - \tilde{K}_t\varepsilon_t - (K_t - \tilde{K}_t)\mathscr{H}\mathscr{F}m_{t-1} \\
=& (\mathscr{F} - \tilde{K}_t\mathscr{H}\mathscr{F})(m_{t-1} - \tilde{m}_{t-1}) + (K_t - \tilde{K}_t)[z_t - \mathscr{H}(\mathscr{F}m_{t-1} + \mathscr{G}u_{t-1})] - \tilde{K}_t\varepsilon_t.
\end{aligned}
\tag{3.10}
$$

Define, $A_t = \mathscr{F} - \tilde{K}_t\mathscr{H}\mathscr{F}$, $B_t = (K_t - \tilde{K}_t)[z_t - \mathscr{H}(\mathscr{F}m_{t-1} + \mathscr{G}u_{t-1})]$ and $C_t = -\tilde{K}_t\varepsilon_t$. Then,

$$
\begin{aligned}
& m_t - \tilde{m}_t \\
&= A_t(m_{t-1} - \tilde{m}_{t-1}) + B_t + C_t \\
&= A_t[A_{t-1}(m_{t-2} - \tilde{m}_{t-2}) + B_{t-1} + C_{t-1}] + B_t + C_t \\
&\cdots \\
&= \prod_{i=0}^{t-1} A_{t-i} \cdot (m_0 - \tilde{m}_0) + (B_t + C_t) + A_t(B_{t-1} + C_{t-1}) + \cdots + A_t \cdots A_3A_2(B_1 + C_1) \\
&= \prod_{i=0}^{t-1} A_{t-i} \cdot (m_0 - \tilde{m}_0) + B_t + C_t + \sum_{i=0}^{t-2}\left(\prod_{j=0}^{i} A_{t-j}(B_{t-1-i} + C_{t-1-i})\right).
\end{aligned}
$$

□

Corollary 1. *The expected value of the separation is,*

$$
\begin{aligned}
& \mathbb{E}(m_t - \tilde{m}_t) \\
&= \prod_{i=0}^{t-1} A_{t-i}M_0 + \sum_{i=0}^{t-2}\left(\prod_{j=0}^{i} A_{t-j}C_{t-1-i}\right) + C_t.
\end{aligned}
\tag{3.11}
$$

Proof. From Equation 3.6, $\mathbb{E}(m_t - \tilde{m}_t)$ follows,

$$\begin{aligned}
&\mathbb{E}(m_t - \tilde{m}_t) \\
&= \mathbb{E}\left(\sum_{i=0}^{t-2}\prod_{j=0}^{i} A_{t-j}\cdot B_{t-1-i} + B_t\right) + \\
&\quad \prod_{i=0}^{t-1} A_{t-i}\mathbb{E}(m_0 - \tilde{m}_0) + \sum_{i=0}^{t-2}\left(\prod_{j=0}^{i} A_{t-j}C_{t-1-i}\right) + \tilde{K}_t\varepsilon_t.
\end{aligned}$$

The actual measurement is: $z_i = \mathscr{H}(\mathscr{F}m_{i-1} + \mathscr{G}u_{i-1} + w_i) + v_i$, where w_i and v_i are Gaussian noises with zero mean. The expected measurement value is: $\mathbb{E}(z_i) = \mathscr{H}(\mathscr{F}m_{i-1} + \mathscr{G}u_{i-1})$ for all i, thus $\mathbb{E}[z_i - \mathscr{H}(\mathscr{F}m_{i-1} + \mathscr{G}u_{i-1})] = 0$. Since $\mathbb{E}[B_i] = 0$, we have,

$$\begin{aligned}
&\mathbb{E}(m_t - \tilde{m}_t) \\
&= \prod_{i=0}^{t-1} A_{t-i}\mathbb{E}(m_0 - \tilde{m}_0) + \sum_{i=0}^{t-2}\left(\prod_{j=0}^{i} A_{t-j}\tilde{K}_{t-1-i}\varepsilon_{t-1-i}\right) \\
&\quad + \tilde{K}_t\varepsilon_t.
\end{aligned} \tag{3.12}$$

Since we assume $\mathbb{E}(m_0 - \tilde{m}_0) = M_0$ in Problem 3, the claim is guaranteed. □

Theorem 5 shows the difference between the two estimated means at step t depends on the initial means, m_0 and $\tilde{m}_0$, and the initial covariance matrices Σ_0 and $\tilde{\Sigma}_0$. This is because the Kalman gain K_t depends on the covariance matrix Σ_t. If target sets $m_0 = \tilde{m}_0$ and $\Sigma_0 = \tilde{\Sigma}_0$, we have $\Sigma_t = \tilde{\Sigma}_t$ for all t since the covariance matrix is updated through the same Kalman prediction and update equation. Thus, $B_t = 0_{2\times 2}$ and then Equation (3.6) can be simplified as:

$$m_t - \tilde{m}_t = \sum_{i=0}^{t-2}\left(\prod_{j=0}^{i} A_{t-j}C_{t-1-i}\right) + C_t.$$

As a result, $m_t - \tilde{m}_t$ is independent of the measurements $\{z_1, z_2, \cdots, z_t\}$ when $m_0 = \tilde{m}_0$ and $\Sigma_0 = \tilde{\Sigma}_0$. Thus, the target can generate spoofing signal inputs by solving Problem 2 offline.

Similarly, following Corollary 1, Problem 3 can be saved offline as well.

Problems 2 and 3 are two nonlinear programming problems for arbitrary vector norms L_p. However, when $p = 1$, they can be formulated as linear programming problems. Linear programming can be solved in polynomial time [48]. When $p = 2$, they become QCQP (Quadratically Constrained Quadratic Program). The following shows the LP and QCQP formulations.

Theorem 6. *If $p = 1$ and the elements in $\mathscr{F}$ and $I - K_t\mathscr{H}$ are all positive, then Problems 2 and 3 can be solved optimally with linear programming. If $p = 1$ and the elements in $\mathscr{F}$ and $I - K_t\mathscr{H}$ are not all positive, then Problems 2 and 3 can be solved optimally with 4^k linear programming instances. If $p = 2$ and $\{\mathscr{H}, \mathscr{F}, Q, R\}$ are diagonal matrices, then Problems 2 and 3 can be solved optimally with linear programming.*

3.3.1 Linear Programming Formulation for L_1 Vector Norm

Here, we show how to formulate Problem 2 using linear programming. A similar procedure can be applied to formulate Problem 3 as linear programming.

The constraint in Problem 2 (Equation 3.4) follows:

$$\begin{aligned} \|m_t - \tilde{m}_t\|_1 &= \left\| \sum_{i=0}^{t-2} \left(\prod_{j=0}^{i} A_{t-j} C_{t-1-i} \right) + C_t \right\|_1 \\ &= \left\| \sum_{i=0}^{t-2} \left(\prod_{j=0}^{i} A_{j+1} \cdot \tilde{K}_{t-1-i} \cdot \varepsilon_{t-1-i} \right) + \tilde{K}_t \varepsilon_t \right\|_1 \\ &\geq d_t, \end{aligned} \tag{3.13}$$

where $t = 1, 2, \cdots, T$. $\prod_{j=0}^{i} A_{t-j} \cdot \tilde{K}_i \in \mathbb{R}^{2\times 2}$ is a constant matrix for each $i \in \{1, \cdots, t-1\}$ and is calculated from the KF iteration with initial covariance Σ_0 and $\tilde{\Sigma}_0$. Since L_1 vector norm is the sum of the absolute values of the elements for a given vector, Problem 2 can be directly formulated as a linear programming problem when $p = 1$.

Then we show how to transform this constraint to a standard linear constraint form $G_t x_t \geq d_t$. To simplify the equation, we use a 2-D case as an example, with $x_t := [\varepsilon_{1x}, \cdots, \varepsilon_{tx}, \varepsilon_{1y}, \cdots, \varepsilon_{ty}]^T$. The problem can be extended to n dimensions follow the same idea. The left side of Equation (3.13) can be formulated as

$$\|m_t - \tilde{m}_t\|_1 = \left\| \begin{matrix} a_0 + a_1\varepsilon_{1x} + \cdots a_t\varepsilon_{tx} + \cdots + a_{2t}\varepsilon_{ty} \\ b_0 + b_1\varepsilon_{1x} + \cdots b_t\varepsilon_{tx} + \cdots + b_{2t}\varepsilon_{ty} \end{matrix} \right\|_1 \tag{3.14}$$

where $a_0, a_1, \cdots, a_{2t}, b_0, b_1, \cdots, b_{2t}$ are corresponding coefficients from Equation 3.6.

Lemma 2. *If the elements in matrices $\mathscr{F}$ and $I - K_t\mathscr{H}$ are positive, then $\|m_t - \tilde{m}_t\|_1$ is a linear combination of $|\varepsilon_{ix}|$ and $|\varepsilon_{iy}|$, and Problem 2 can be solved as a single LP instance.*

Proof. According to the proof of Theorem 5, all the coefficients $\{a_1, \cdots, a_{2t}, b_1, \cdots, b_{2t}\}$ are positive if the elements in matrices $\mathscr{F}$ and $I - K_t\mathscr{H}$ are positive. Therefore, the objective function and the constraints are linear in $|\varepsilon_{ix}|$ and $|\varepsilon_{iy}|$. There always exists an optimal solution where all $\varepsilon_{ix} \geq 0$ and $\varepsilon_{iy} \geq 0$ or where all $\varepsilon_{ix} \leq 0$ and $\varepsilon_{iy} \leq 0$. The objective function in both cases will be the same. Without loss of generality, we can assume $\varepsilon_{ix} \geq 0$ and $\varepsilon_{iy} \geq 0$, which can be solved using a single LP instance. $\square$

The linear programming strategy containing k constraints is presented in Algorithm 2. G denotes matrix in the linear constraint $Gx \geq D_k$ where $x := [\varepsilon_{1x}, \cdots, \varepsilon_{Tx}, \varepsilon_{1y}, \cdots, \varepsilon_{Ty}]^T$ and D_k is the collection of k nonzero separations d_t, $t \in \{1, \cdots, T\}$.

If Lemma 2 does not hold, it is possible that some elements in $a_0, a_1, \cdots, a_{2t}, b_0, \cdots, b_{2t}$ can be positive and some are negative. In general, there are four different cases depending on the sign of the first row and the second row for considering each constraint $\|m_t - \tilde{m}_t\|_1 \geq d_t$ (Equation 3.14). Then we can obtain four linear optimization problems along four different sub-constraints of each constraint $\|m_t - \tilde{m}_t\|_1 \geq d_t$. Thus, in the worst

Algorithm 2: Linear Programming Formulation

```
1 **Initial** $\leftarrow \{(x_o, \Sigma_0, \mathscr{F}, \mathscr{H}, \mathscr{G}, Q, R, u\}$
2 $G \leftarrow 0_{k\times n\cdot T}$
3 Calculate Kalman gain $\tilde{K}_1, \cdots, \tilde{K}_T$
4 **for** $i = 1$ *to the* $q_{th}$ *value in* $D_k$ **do**
5     $g = \prod_{j=i}^{T-1} A_{j+1}\tilde{K}_i$;
6     $G_{q,i}$ = sum of all rows in $g$
7 **end**
8 **Return** $G$
```

case, the optimal solution can be obtained by solving 4^k linear optimization problems. We run Algorithm 2 4^k times by changing the sign of rows in g (Line 5) appropriately.

3.3.2 Quadratically Constrained Quadratic Program Formulation for L_2 Vector Norm

When $p = 2$, Problems 2 and 3 can be formulated as QCQPs[20]:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}x_\varepsilon^T P_0 x_\varepsilon \\ &\text{s.t.} \quad -\frac{1}{2}x_\varepsilon^T D_t^T D_t x_\varepsilon + d_t^2 \leq 0, \quad \forall t \in \{1,\dots,T\} \end{aligned} \tag{3.15}$$

To simplify the equation, we use a 2-D case as an example, where $x_\varepsilon = [\varepsilon_{1x}^2, \varepsilon_{1y}^2, \cdots, \varepsilon_{Tx}^2, \varepsilon_{Ty}^2]^T$, $P_0 = I_{2T}$, and

$D_t \in \mathbb{R}^{2T\times 2T} :=$

$$\begin{bmatrix} \prod_{j=1}^{t-1} A_{j+1}\tilde{K}_0 & & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \prod_{j=t-1}^{t-1} A_{j+1}\tilde{K}_{t-1} & 0 & 0 & 0 \\ 0 & \cdots & 0 & \tilde{K}_t & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & \ddots \end{bmatrix}$$

Unfortunately, the QCQP formulations for these three problems are NP-hard since the constraint in each problem is concave. If $\mathscr{F}, \mathscr{G}, \mathscr{H}, \tilde{\Sigma}_0$ are diagonal matrices, it can be shown that D_t is also a diagonal matrix. We can transform the QCQP formulation to a linear programming problem by changing variables $\{\varepsilon_{tx}^2, \varepsilon_{ty}^2\}$, $t = \{1,2,...,T\}$, and using a procedure similar to $p = 1$.

If D_t is not a diagonal matrix, one solution is to apply the inequality $\sqrt{2}\|x\|_2 \geq \|x\|_1$ between L_1 vector norm and L_2 vector norm. The constraint can be changed to L_1 vector norm, which is a stricter constraint. A sub-optimal solution can be obtained by using the L_1 vector norm.

3.3.3 Receding Horizon: Spoofing with online measurement

Problems 2 and 3 describe the offline versions for spoofing. We also extend the offline problems to an online version. The following formulates an online spoofing scenario.

Consider a target with motion model (Equation (3.1)), measurement model (Equation (3.2)), and spoofing measurement model (Equation (3.3)). Assume the target does not know $\mathcal{N}(x_0, \Sigma_0)$. It collects a series of measurements $\{z_1^{real}, z_2^{real}, \cdots, z_{t^o}^{real}\}$ from step 1 to current step t^o. Find a sequence of spoofing signal inputs, $\{\varepsilon_{t^o}, \varepsilon_{t^o+1}, \cdots, \varepsilon_{t^o+H}\}$ to achieve desired separation d_t between $\tilde{m}_t$ and m_t (in expectation) within future H steps. Such that

$$\text{minimize} \quad \sum_{t=t^0}^{t^o+H} \gamma_t \cdot \|\varepsilon_t\|_p^p$$

$$\text{s.t.} \quad \|\mathbb{E}(m_t - \tilde{m}_t)\|_p^p \geq d_t^p, \quad \forall t \in \{t^o, \cdots, t^o + H\} \tag{3.16}$$

where $\gamma_t \in \mathbb{R}^+$ is a weighting parameter, t^o is the current time, and H is the predictive time horizon. The target applies $\varepsilon_t = \varepsilon_{t^o}$ as spoofing signal input at each step t.

3.4 Simulations

In this section, we simulate the effectiveness of spoofing strategies for Problems 2 and problem 3 and for the online case (Section 3.3.3) where a target designs spoofing signals ε_t to mislead an observer by achieving the desired separations d_t between m_t and $\tilde{m}_t$. Our code is available online.[2]

We consider the L_1 vector norm and the following models,

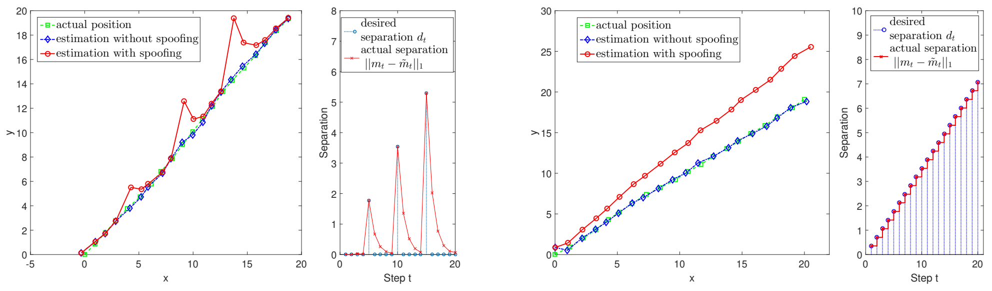
$$\mathscr{F} = I_{2\times 2}, \mathscr{G} = I_{2\times 2}, u = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, R = 0.5I_{2\times 2}, Q = 0.5I_{2\times 2}.$$

Set the weight $\gamma_t = 1$ for all t.

For Problem 2, set the initial condition for the KF as,

$$\Sigma_0 = I_{2\times 2},\ m_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}^T.$$

[2]`https://github.com/raaslab/signal_spoofing.git`



(a) Desired separations, $d_5 = 1.77$ and $d_{10} = 3.54$, with $T = 20$.

(b) Desired separations, $d_t = 0.25\sqrt{2}t$, with $t = 3$ to $T = 15$.

Figure 3.3: Offline signal spoofing with known (m_0, Σ_0).

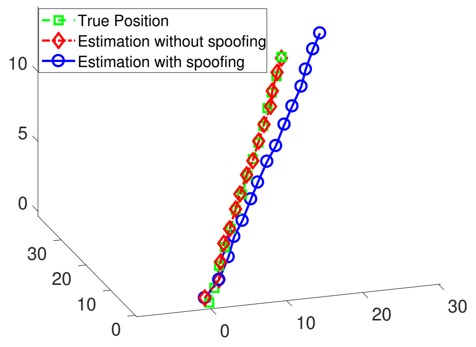
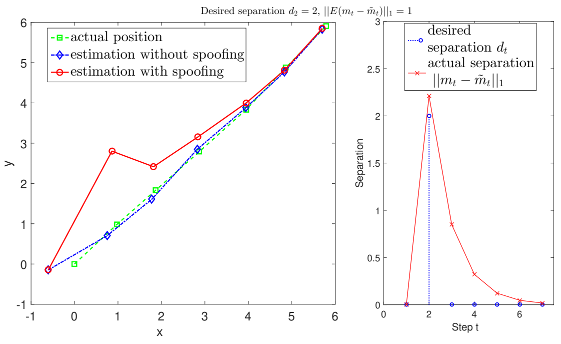


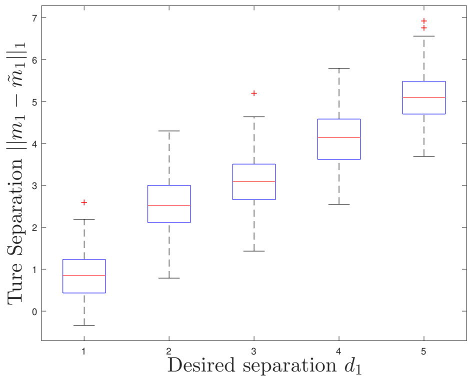
Figure 3.4: Signal spoofing with known (m_0, Σ_0) in 3D environment. Desired separations, $d_3 = 3, d_3 = 4, \cdots, d_{10} = 10$.

Since the target knows $\mathcal{N}(x_0, \Sigma_0)$, it sets $\tilde{m}_0 = m_0$ and $\tilde{\Sigma}_0 = \Sigma_0$. We first consider a scenario where the target wants to achieve the desired separation at steps, $t = 5, 10, 15$, denoted as $d_5 = 1.77$, $d_{10} = 3.54$ and $d_{15} = 5.30$ with the optimization horizon $T = 20$. The target generates a sequence of spoofing signals $\{\varepsilon_1, \cdots, \varepsilon_{20}\}$ offline by using a linear programming solver. The spoofing performance is shown in Figure 3.3-(a) where the true separations are the same as the desired separations. The same successful spoofing is achieved when the desired separations are chosen as $d_t = 0.25\sqrt{2}t$, $t = \{3, ..., 15\}$, as shown in Figure 3.3-(b).

The problem formulation applies in higher dimensional systems as well, not just 2D. Figure 3.4 shows an example of misleading a KF in a 3D environment.



(a) Desired separation, $d_1 = 2$.



(b) Results with $d_1 = \{1,2,3,4,5\}$ for 100 trials.

Figure 3.5: Offline signal spoofing with unknown (m_0, Σ_0).

In Problem 3, the target knows $\mathbb{E}(m_0 - \tilde{m}_0) = M_0$ but does not know Σ_0. The spoofing result is no longer deterministic but holds in expectation $\|\mathbb{E}(m_t - \tilde{m}_t)\|_1 \geq d_t$. Figure 3.5-(a) shows spoofing signals for desired separations as $d_1 = 2$ with $T = 6$ and $M_0 = 1$. Set $\mathcal{N}(\tilde{m}_0, \tilde{\Sigma}_0)$ as $\mathcal{N}(0, 1.5I_2)$, m_0 as a random variable ($m_0 \sim \mathcal{N}(1,1)$) and $\Sigma_0 = I_2$. In order to see the effectiveness of the spoofing signals $\{\varepsilon_1, \cdots, \varepsilon_5\}$, we conduct 100 trials for each desired separation $d_2 \in \{1,2,3,4,5\}$. Figure 3.5-(b) shows that the $\|m_1 - \tilde{m}_1\|_1$ is no longer deterministic, but $\|\mathbb{E}(m_1 - \tilde{m}_1)\|_1$ is close to the desired value $d_1 = 2$.

For the online case, spoofing signals are continuously generated by using receding horizon optimization with new noisy measurements. We set the receding horizon as $H = 15$. Even though the offline strategy performs comparably to the online strategy (Figure 3.6), the online spoofing strategy achieves almost the same separation as desired, while the offline strategy has certain divergence. This is because the online strategy can update the measurement at each step.

3.5 Signal spoofing with failure detector

In this section, we evaluate the performance of the false data injection strategy in the

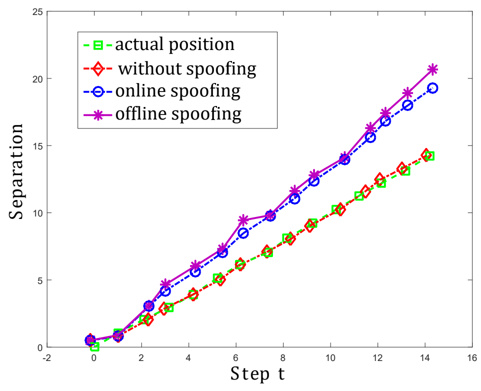


Figure 3.6: Online spoofing and offline spoofing with unknown (m_0, Σ_0).

presence of a failure detector. We show the conditions (Theorem 7) under which the generated false data can mislead a χ^2 detector. This result can be also extended to other residual-based detectors.

3.5.1 χ^2 failure detector

A χ^2 detector computes the following measure,

$$g_t = r_t^T \Sigma_{r_t}^{-1} r_t, \tag{3.17}$$

where $r_t = z_t - \mathscr{H} m_t$ is the innovation or measurement residual of the KF. Here, Σ_{r_t} is the covariance matrix of the residual [98]. The residual is Gaussian since it is the linear combination of two Gaussian random variables. It is known that g_t is χ^2 distributed with n degrees of freedom. If $g_t >$ threshold, the detector raises an alarm that the filter is under attack [21].

First, we review the Kalman Filter update equations,

$$m_{t|t-1} = \mathscr{F} m_{t-1|t-1} + \mathscr{G} u_t, \tag{3.18}$$

$$m_{t|t} = \mathscr{F} m_{t|t-1} + K_t(z_t - \mathscr{H} m_{t|t-1}), \tag{3.19}$$

where K_t is the Kalman gain and is given by,

$$K_t = (\mathscr{F}\Sigma_{t|t-1}\mathscr{F}' + R_t)\mathscr{H}'(\mathscr{H}\Sigma_{t|t-1}\mathscr{H}' + Q_t)^{-1}. \tag{3.20}$$

We use the notation, $\tilde{\cdot}$, to indicate the system under attack.

Intuitively, the lower the amount of injected attack signal, the less likely it will be detected. This is the motivation behind reducing the energy of the injected system. Nevertheless, when designing an attack sequence over a time horizon, we may have to carefully design the separation sequence $d_1, d_2, \cdots$, so that they are not too large. In the following, we modify the notion of a successful attack from [70] and show how to use that to generate a successful attack sequence. The differences between two systems are defined as,

$$\Delta m_t \triangleq \tilde{m}_t - m_t, \quad \Delta z_t \triangleq \tilde{z}_t - z_t, \quad \Delta r_t \triangleq \tilde{r}_t - r_t. \tag{3.21}$$

Definition 3.1. *Given $\delta > 0$, the χ^2 detector is successfully attacked if there exists an attack sequence $\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_T$ such that the following holds:*

$$||\Delta r_t|| < \delta, \quad \forall t,$$

where Δr_t is defined above.

Remark 3. *If Δr_t is bounded, then the difference of its quadratic form $\tilde{g}_t - g_t$ is also bounded. Also, as pointed out by [70], by linearity, we can find a $\delta' > 0$, such that*

$|P(\tilde{g}_t > threshold) - P(g_t > threshold)| \leq \delta', \quad \forall t.$

This definition of successful attack follows the (ε, α)–attack definition by Mo et al. [70]. When the probability of the alarm $P(\tilde{g}_t > threshold) - P(g_t > threshold)$ is bounded and a small enough δ, the alarm rate δ' will converge to the false alarm rate of the healthy system. Mo et al. [71] presented the relationship between δ and δ'.

Given the threshold for the χ^2 detector and δ, the question is how to set desired separations $d_1, d_2, \cdots$ such that we can avoid being detected. In the following, we give a sufficient condition for designing $d_1, d_2, \cdots$.

Theorem 7. *If the separations Δm_{t+1} and Δm_t satisfy $||K_{t+1}^{-1}|| \cdot \|\Delta m_{t+1} - \mathscr{F}\Delta m_t\| \leq \delta$, then the proposed algorithm can successfully attack the χ^2 detector at step $t+1$.*

Proof. Manipulating Equations (3.18), (3.19), (3.20), and (3.21), we can prove that,

$$\Delta m_{t+1} = K_{t+1}\Delta r_{t+1} + \mathscr{F}\Delta m_t. \tag{3.22}$$

Taking the norm of Equation (3.22), we have

$$\begin{aligned} ||\Delta r_{t+1}|| &= \left\| -K_{t+1}^{-1}\Delta m_{t+1} + K_{t+1}^{-1}\mathscr{F}\Delta m_t \right\| \\ &\leq ||K_{t+1}^{-1}|| \cdot \|\Delta m_{t+1} - \mathscr{F}\Delta m_t\| . \end{aligned} \tag{3.23}$$

Therefore,

$$||\Delta r_{t+1}|| \leq ||K_{t+1}^{-1}|| \cdot \|\Delta m_{t+1} - \mathscr{F}\Delta m_t\| . \tag{3.24}$$

We apply the condition of successful attack. If we have,

$$||K_{t+1}^{-1}|| \cdot \|\Delta m_{t+1} - \mathscr{F}\Delta m_t\| \leq \delta \tag{3.25}$$

then,

$$||\Delta r_k|| < \delta.$$

Note that $\mathscr{F}$ is a known matrix, and the Kalman gain K_t can be computed from the initial covariance matrix Σ_0. Hence, we can design the attack sequence $\varepsilon_1, \varepsilon_2, \cdots$ for a χ^2 detector given the threshold δ. □

Theorem 7 shows that if we want to attack a system with χ^2 detector, the strategy is to make the difference between two consecutive desired separations, d_{t+1} and $||\mathscr{F}||d_t$, as small as possible. In general, when we design the attack sequence, we want to increase the separation to mislead the system. Without loss of generality, we can consider the case that all the elements in $m_t - \mathscr{F}\Delta m_{t-1}$ are non-negative. Given a known separation from previous step $t-1$, we have the following condition for d_t when we design the desired separation:

$$d_t - ||\mathscr{F}\Delta m_{t-1}|| \leq \delta, \quad t > 1. \tag{3.26}$$

Remark 4. *Applying Theorem 7 and Equation (3.26) and given δ, we can design a sequence of separations $d_1, d_2, \cdots, d_T$ a priori since $d_t = ||\Delta m_t||$. For example, if we know the Kalman filter's initial condition, assuming $d_1 = 0$, we have,*

$$d_2 = \delta ||K_1||. \tag{3.27}$$

With a known Δm_2 from the proposed LP algorithm, d_3 can be designed with the following equation,

$$d_3 = (||\mathscr{F}\Delta m_2|| + \delta) \cdot ||K_1||. \tag{3.28}$$

Iteratively, we can get the desired separation for all times and guarantee a successful attack

$(||\Delta r_t|| < \delta, \quad t = 1, 2, \cdots, T.)$[3].

In the following section, we will provide an example that by increasing the separation with given condition. The simulation shows the χ^2 will not alarm when the separation is designed as Theorem 7.

3.5.2 Simulation with χ^2 detector

We consider the L_1 vector norm and the same model from the simulation section. We use the following parameters:

$$\mathscr{F} = I_{2\times 2}, \mathscr{G} = I_{2\times 2}, R = 0.1 I_{2\times 2}, Q = 0.1 I_{2\times 2}, \delta = 0.1.$$

Given $\delta = 0.1$, we can design the separation d_t. The attack result and the χ^2 detector value are shown in Figure 3.7.

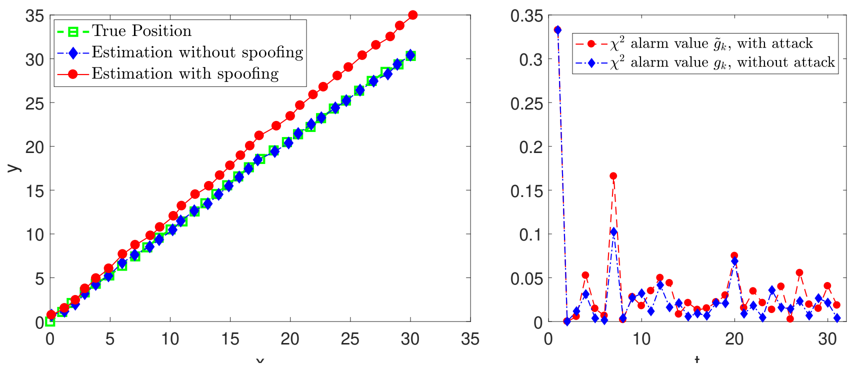


Figure 3.7: Estimated positions and the χ^2 detector's output (g_k), when the spoofing signals are injected by setting d_t based on Theorem 7.

[3]Since the inequality is conservative, if the equation converges to $d_{t+1} = d_t$, we can add a small term $O(t)$, and let $d_t = (||\mathscr{F}\Delta m_{t-1}|| + \delta) \cdot ||K_t|| + O(t)$.

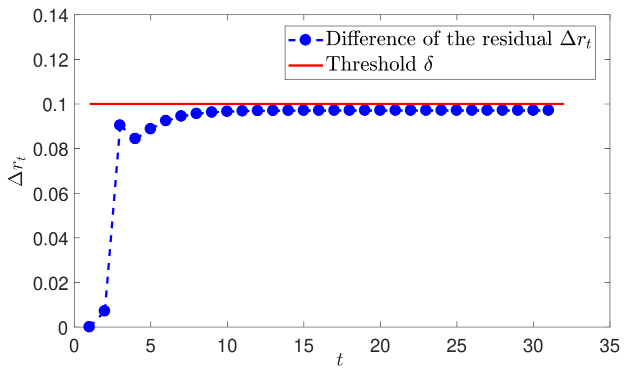


Figure 3.8: The differences of the two residual $||r_t - \tilde{r}_t||$. The threshold $\delta = 0.1$.

The differences (Δr_t) between the residual and the threshold are shown in Figure 3.8. We ran 1000 trials using this strategy as shown in Figure 3.7. The χ^2 detector detected the attack in 112 trials (The false alarm rate is equal to 11.054%; this rate indicates the χ^2 detector cannot tell whether the alarm is a false alarm or not). This is close to the actual false alarm rate without any attack. In this scenario, the system will not be able to distinguish between false and true alarms. Thus, the attack strategy is able to successfully mislead the χ^2 detector.

3.6 Summary

We study the problem of injecting spoofing signals to achieve a desired separation in the output of a Kalman filter without and with attack. We study many variants of the problem. Our main approach was to formulate the problems as nonlinear, constrained optimization problems in order to minimize the energy of the spoofing signal. We show that under some technical assumptions, the problems can be solved by linear programming optimally. We present a more computationally expensive approach to solve the problem, without the aforementioned assumptions. We also present a sufficient condition for this strategy to mislead the χ^2 failure detector.

Our immediate future work is to study the game-theoretic aspects of the problem. In this work, we did not consider any active strategy being employed by the observer to mitigate the attack. In future works, we will consider the case of designing spoofing signals that explicitly take the attack mitigation strategies into account. In all the problems considered in this chapter, the desired separations are taken as inputs provided by the user. The simulation results suggest that carefully choosing a specific profile of the desired separation can make it harder to detect by the observer. A possible extension is to automatically generate the optimal profile that not only minimizes the signal energy but also ensures that it is not detected by the observer. Another future work is to extend the strategy to more general non-linear state estimation approaches, such as the extended Kalman filter, unscented Kalman filter, and particle filters.

Chapter 4: Planning a Reconnaissance Mission Against an Adversary with Symmetric Information

In this chapter, we introduce and study the problem of planning a trajectory for an agent to carry out a reconnaissance mission while avoiding being detected by an adversarial opponent. We start with the symmetric case and present techniques to reduce existing algorithms' computational cost.

4.1 Introduction

Planning for visually covering an environment is a widely-studied problem in robots with many real-world applications, such as environmental monitoring [102], precision farming [77], ship hull inspection [52], and adversarial multi-agent tracking [42, 119]. 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) [23] and is closely related to the Art Gallery Problem (AGP) [74]. The goal in AGP is to find the minimum number of cameras required to see all points in a polygonal environment. In this chapter, 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 4.1). The agent, on the other hand, is tasked with covering the environment while avoiding detection by the opponent. This

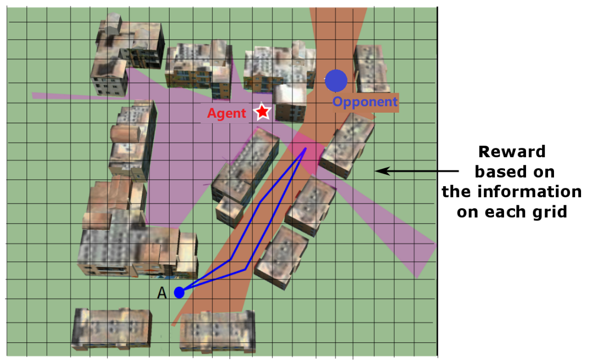


Figure 4.1: The agent (the red star) aims to maximize the total area covered within the given time horizon while at the same time minimize the number of times the opponent (the blue dot) detects it.

loosely models stealth reconnaissance missions where the agent is required to exercise caution while collecting information in the environment. We consider the version where there is a finite time horizon within which the agent's objective 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 4.1). For ease of illustration, we assume that the agent's and the opponent's sensing ranges are unlimited and obstacles in the environment can block lines-of-sight. The case of limited sensing range can be easily incorporated since it does not require any change to the algorithm but only to the way the reward and penalty are computed. 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 or the opponent moves to a cell that can see the agent).

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 sum 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 4.1), the positive reward can be the total area that is scanned by the agent along its path. In this chapter, we consider the case where the agent receives a fixed negative reward every time it is detected by the opponent. The total reward is a linear combination of the two reward functions.

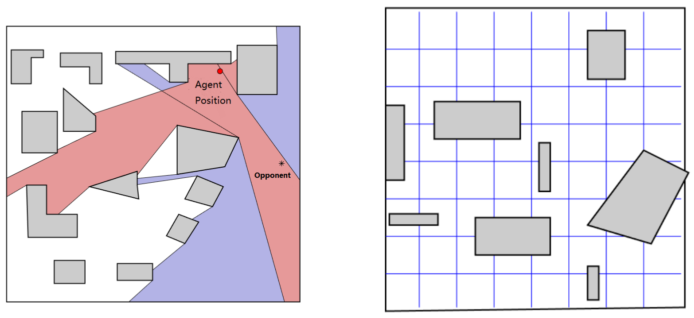
The proposed problem builds on classic pursuit evasion games [81, 82, 116] and visibility-based reconnaissance problems [68, 90]. In classic pursuit-evasion, the evader (i,e., the agent in our setting) always tries to avoid the capture of the pursuer (i,e., the opponent). In our setting, in addition 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 [17] or if the evader is surrounded by pursuers, when there are more than two pursuers present [45]. 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 differs 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. [81] 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. [79] 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. Since the objective function is different, the heuristic used to speed up the search in [81] will not be directly applicable. However, ideas from [81] can be extended to our setting. Likewise, the visibility-based decomposition from [79] cannot be directly applied since the proof of correctness specifically requires a pursuit-evasion setting where the pursuer wins when it is co-located with the evader. Nevertheless, the abstraction with suitable modifications could be useful to speed up the search in our setting.

We abstract the underlying geometry and model the problem as a discrete, sequential, two-player, zero-sum game. Minimax tree search [38] and Monte-Carlo tree search (MCTS) [86] 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 [54] and simultaneous-move games [66].

To reduce the computation time of minimax tree search and MCTS, we apply branch-and-bound [62] and the principle of optimality [12] to prune the trees. Our contributions are in applying these techniques to the adversarial reconnaissance problem and empirically evaluating the computational time with pruning. We show the resulting pruned tree still preserves optimality. To further reduce the computational time, we then employ a multi-resolution planner (similar to [65]) that allows the agent to change the spatial horizon to build a search tree with fewer levels. Our empirical evaluation shows that the resulting strategy outperforms the fixed resolution one, especially in larger environments.

The rest of the chapter is organized as follows. We begin by describing the problem setup in Section 4.2. We then describe two tree search techniques in Section 4.3 and present two approaches for improving the computational efficiency of these tree search techniques in Section 4.4. Next, we evaluate the effectiveness of the proposed approaches through extensive simulations in Section 4.5. In the end, we summarize the chapter and outline some future work in Section 4.6.



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

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

Figure 4.2: 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).

4.2 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 cell 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 chapter, we assume that both sensing ranges are unlimited for ease of illustration.[1] (4) The opponent has a sensor that can detect the agent when the agent is within its visibility region. (5) There

[1]The case of limited sensing range can be incorporated since the sensing range is used only when computing the positive and negative rewards. To incorporate a limited sensing range, we will only need to change the evaluation function of the terminal nodes. The visibility library [73] we use in our empirical evaluation can handle this case.

is no motion uncertainty associated with the agent and opponent. (6) The agent and the opponent are aware of each other's positions. 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 [15] 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 [95], occupancy maps [43], and randomized methods such as probabilistic roadmaps [51], and Rapidly-exploring Random Trees [47, 61].

The complexity of the tree search algorithm will depend on the number of vertices (or grid cells) in a given discretization. In Section 4.4, 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 4.4.3) 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 chapter. 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:

$$\max_{\pi_a(t)} \min_{\pi_g(t)} \left\{ R(\pi_a(t)) - \eta(\pi_a(t), \pi_g(t))P \right\}. \tag{4.1}$$

While the objective of the opponent is as follows:

$$\min_{\pi_g(t)} \max_{\pi_a(t)} \left\{ R(\pi_a(t)) - \eta(\pi_a(t), \pi_g(t))P \right\}, \tag{4.2}$$

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 chapter, 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.

4.3 Tree Search Techniques

We abstract the underlying geometry and model the problem as a discrete, sequential, two-player, zero-sum game. Minimax tree search [38] and MCTS [86] 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 [87], 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 model this problem as a turn-based game[2]. 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 4.5.

4.3.1 Minimax Tree Search

A minimax tree search is a commonly used technique for solving two-player zero-sum games [87]. 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.

[2]Note that this may introduce errors in boundary cases. We will discuss this in Chapter 6.

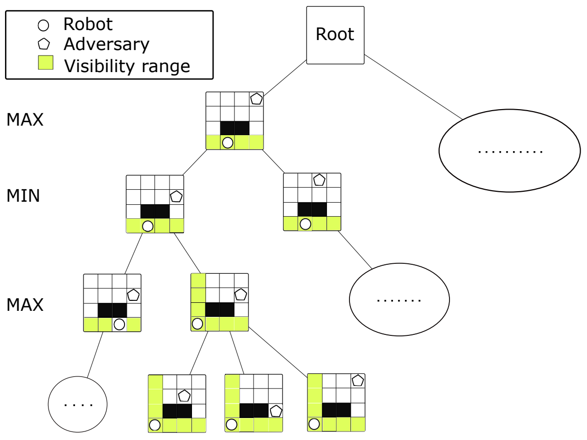


Figure 4.3: A (partial) minimax search tree. The root node contains the initial states of the agent and the opponent. Two successive levels of the tree correspond to one time step. The agent moves first to an available position in order to maximize the reward (MAX level). The opponent moves subsequently to a neighboring cell to minimize the agent's reward (MIN level).

- *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 4.3 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.

Algorithm 3: 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
```

4.3.2 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 *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.

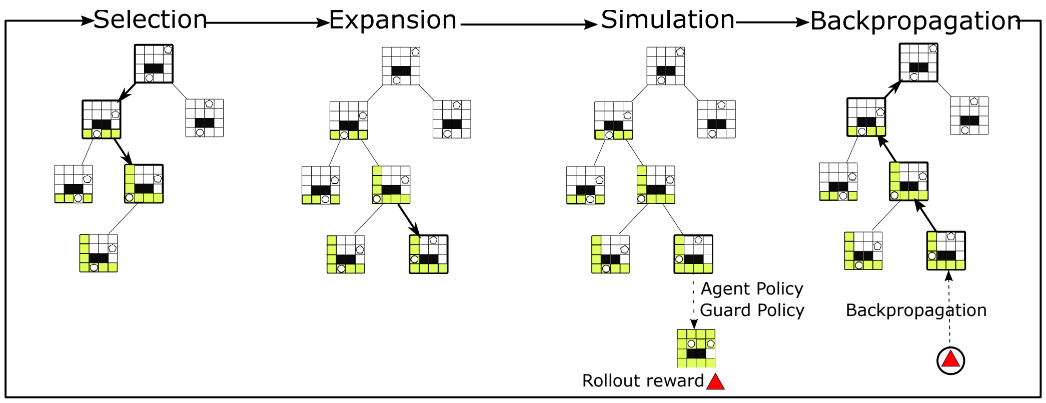


Figure 4.4: Four iteration steps in Monte-Carlo search tree.

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 [27] (Figure 4.4). Note that we present the pseudo-code of MCTS for completeness; however, this is not a novel contribution of our work. Our contribution is the application and empirical evaluation of minimax and MCTS with pruning to the adversarial reconnaissance problem.

- **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) [54] 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.

- **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** (Lines 13-17, Algorithm 2): The total reward found is then used to update the reward value stored at each of the predecessor nodes.

Algorithm 4: Monte-Carlo Tree Search

```
1  function MCTS(Tree, Initial agent and opponent state)
2      Create root node $v_0$ with initial opponent and agent state $s_0$;
3      while maximum number of iterations not reached do
           // Selection
4          $v_i \leftarrow$ Monte_Carlo_Selection($Tree, v_0$)
           // Expand or rollout
5          if level($v_i$) $= T$ and $n(v_i) = 0$ then
               // Expand
6              Tree $\leftarrow$ Expand($Tree$,v˙i)
7              if Newly added node can be pruned then
8                  break
9              end
10         else
               // Rollout
11             $R \leftarrow$ Rollout($v_i$);
12         end
           // Backpropagation
13         while $v_i \neq$ NULL do
               // Update total reward value
14             $Q(v_i) \leftarrow Q(v_i) + R$
15             $n(v_i) \leftarrow n(v_i) + 1$
16             $v_i \leftarrow$ parent of $v_i$
17         end
18         $N \leftarrow N + 1$
19     end
20     return Tree
21 end
```

Algorithm 5: MCTS selection

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

Algorithm 6: 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
```

Given a sufficient number of iterations, the MCTS with UCT is guaranteed to converge to the optimal policy [8, 66]. 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 computation time.

4.3.3 Online Planning with Search Tree

After searching up to a finite horizon, the agent can execute the action returned. If we are using minimax tree search, at the root node, the agent executes the first action along the optimal path (for the planning horizon) 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.

4.4 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.

4.4.1 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 4.5.

In the case of the minimax search tree, we can apply a classical pruning strategy called *alpha-beta pruning* [86]. 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 [86] 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.

The pruning techniques we will discuss next 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.

We first apply branch-and-bound [62], to our application scenario to prune nodes that are guaranteed to not be part of the optimal solution. Walsh et al. [108] presented the Forward Search Sparse Sampling algorithm which combines MCTS with a branch-and-bound style pruning algorithm. We apply a similar strategy for the adversarial reconnaissance problem. 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.

Remark 5. *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.

In addition to branch-and-bound, we can also apply the principle of optimality that is employed in dynamic programming and other graph-search techniques [108] to reduce the search space. The main idea in this pruning strategy (i,e., Corollary 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. The following is a direct corollary of the principle of optimality [12].

Corollary 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. □

4.4.2 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 $\mathscr{K}$ 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 $\Theta((ab)^{T/2})$ [53].

With pruning techniques proposed in Remark 5 and Corollary 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[3] nodes are always searched first. In Corollary 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 pruned away. In an environment with $\mathscr{K}$ grid points/cells, there are $\mathscr{K}^2$ possible combinations of the agent and the opponent's positions. Thus, at most $\mathscr{K}^2$ nodes are listed at each level of the search tree in the best case. The size of the tree is lower bounded by $\Omega(\mathscr{K}^2 \cdot 2T)$. In the trivial case where $a, b = O(\mathscr{K})$, we see that the best case is realized. Therefore, in the best case the size of the tree with pruning will be $\Theta(\mathscr{K}^2 \cdot 2T)$.

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 4.5.

4.4.3 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 4.5). In-

[3]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.

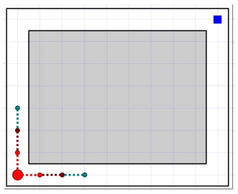
stead, we apply a multi-resolution planning strategy [65] that changes the spatial resolution at different levels of the tree. The key idea of multi-resolution planning in this context is that we plan with a higher resolution closer to the agent and the opponent and with lower resolution in the space that is further away from the agent and the opponent. We define the resolution as follows: Consider a search tree $\mathscr{T}$ 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.

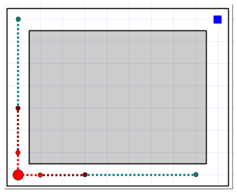
Traditionally, we fix the resolution for all levels as one, e.g., $C(k) = 1$, as shown in Figure 4.5 (a). All the nodes expand with the same resolution. The agent (red square) looks ahead for only three steps in this 8×7 environment. The agent at least needs to plan for seven steps to discover the opponent (blue square) located in the top right corner.

In contrast, we apply the multi-resolution approach, as shown in Figure 4.5 (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 computational cost. In Figure 4.5 (c), we show an example that increasing the resolution makes the agent miss the small corridor, which could have led the agent to a larger, unscanned environment. However, our empirical results suggest that this does not happen often in the space of problems we ran, 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


(a) Expand the search tree with fixed resolution.


(b) Expand the search tree with changing resolution.

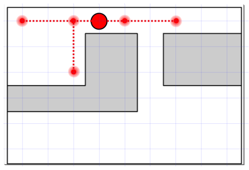

(c) Disadvantage: Expand the search tree with changing resolution will lose some accuracy. The agent could miss the corridor and turn left by mistakenly thinking the environment is larger on the left.

Figure 4.5: 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.

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.

4.5 Evaluation

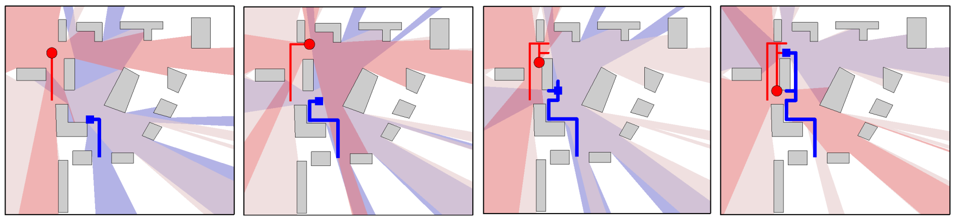
In this section, we evaluate the proposed techniques in the context of a reconnaissance mission. We assume the visibility range of the agent and the opponent are both unlimited (only restricted by the obstacles in the environment). We assume that the opponent is reasoning independently using the same minimax/MCTS strategy to plan for its own actions. In the first set of experiments, the motion model of the agent and the opponent obeys the modeling assumptions we make. As such, it is only the fact that the planning horizon does not extend until the end of the episode requires online replanning. In the second set of experiments in Gazebo, we do not restrict the agent and the opponent to follow all the assumptions and study how well the techniques extend to a more realistic setting.

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 [73] 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.

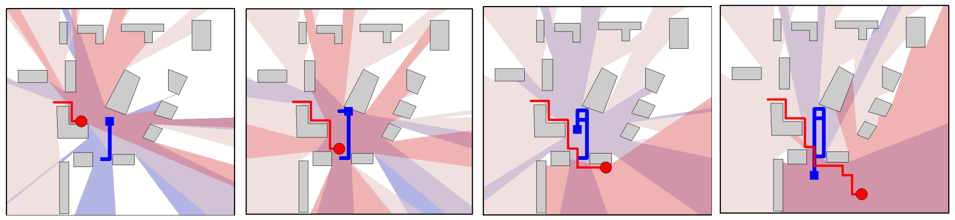
4.5.1 Varying Penalty

Both the minimax tree search and MCTS can find the same optimal solution for these instances. Figures 4.6 and 4.7 show two examples of the policy found by MCTS method, using high and low negative penalty values (P in Equation 4.1), 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 4.6). With a lower negative reward $P = 3$, the agent prefers to explore more area (Figure 4.7).



(a) $t = 5$ (b) $t = 10$ (c) $t = 15$ (d) $t = 20$

Figure 4.6: 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 4.7 shows the case with a lower penalty.



(a) $t = 5$ (b) $t = 10$ (c) $t = 15$ (d) $t = 20$

Figure 4.7: 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), $t = 5, 10, 15, 20$.

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

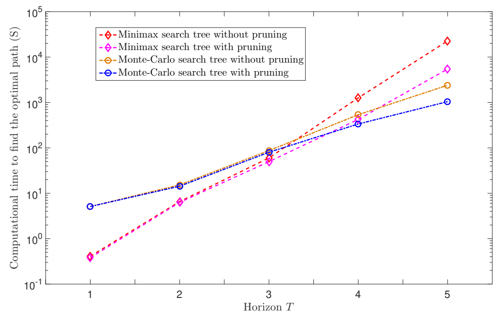


Figure 4.8: Comparison of the time required to find the optimal solution with the minimax tree and the MCTS, with and without pruning. Note that the y axis is in log scale.

4.5.2 Pruning Techniques

MCTS: We evaluate the computational time required to find the optimal solution by varying the time horizon T. Figure 4.8 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 4.8, 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.

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 4.1, 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 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 4.1 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.

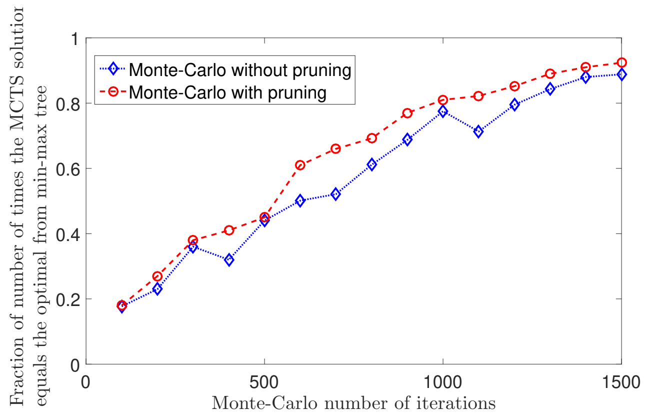


Figure 4.9: Effect of increasing the number of iterations in MCTS, with and without pruning, on the the likelihood of finding the optimal solution. The *y*–axis shows the fraction of the number of trials (out of 50 trials) MCTS was able to find the optimal solution given by the minimax tree for $T = 3$.

Table 4.1: 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

Figure 4.9 shows the fraction of the times we find the optimal solution as a function

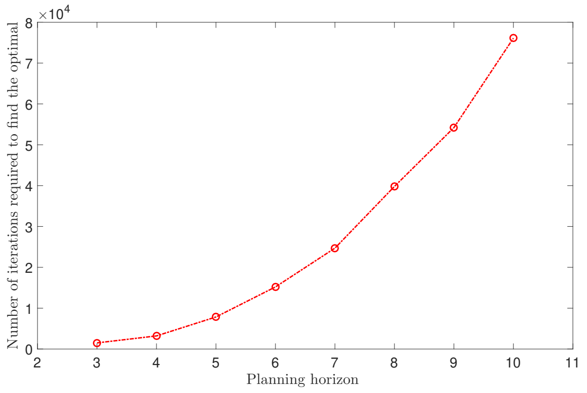


Figure 4.10: Effect of the planning horizon on the number of iterations required to find the optimal solution for MCTS with pruning.

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.

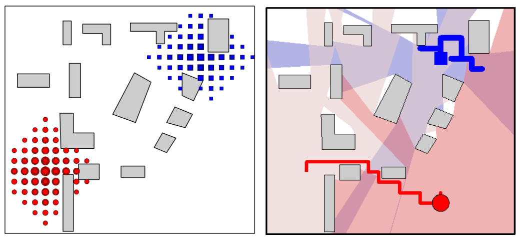
We make the following observations from Figure 4.9: (1) The proposed pruning strategy increases the (empirical) likelihood of finding the optimal solution in the same number of iterations; and (2) The probability of finding the optimal solution grows as the number of iterations grows.

The number of iterations required to find the optimal solution also depends on the planning horizon. Figure 4.10 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, Here, we use 80% as the criterion for success, chosen arbitrarily. A different threshold can be chosen depending on the application requirements. For example, if optimality is the goal, then a higher threshold can be used for analysis. We expect the trend shown here to hold — prior work [26, 50] have shown that the performance of MCTS improves non-linearly as the number of iterations increases. From our preliminary results, we find that the number of iterations required to find success 80% or more times increases exponentially as we vary the planning horizon.

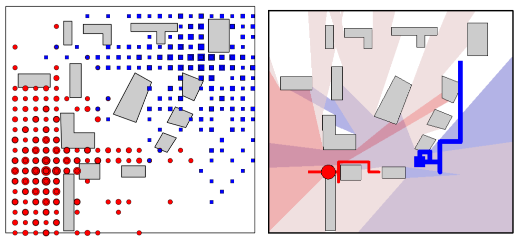
4.5.3 Changing Resolution Approach

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 4.11, 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 4.11. Figure 4.11(a) shows the limitation of the traditional approach: 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 4.11(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.



(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".



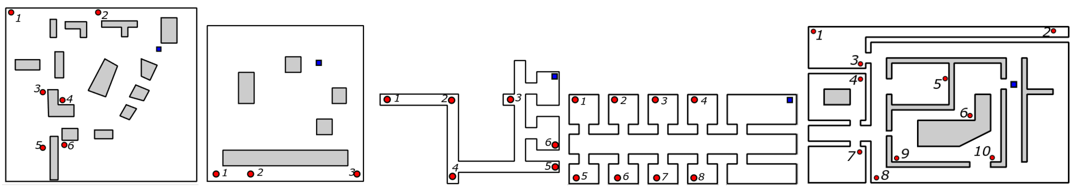
(b) Search tree with increasing resolution. The search depth in the tree remains the same. The Monte-Carlo search tree returns a simulation result that the opponent can move closer to the agent, and the agent avoids the opponent, even when they are far away initially.

Figure 4.11: 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.

The right figure shows the online path for 20 steps. With a changing resolution in the search process, Figure 4.11(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 4.13, we compare the collected reward between fixed resolution with changing resolution. We test the results in different simulation environments that are shown in Figure 4.12. We compute the difference between the average reward collected various initial positions for the agent (marked as red dots in Figure 4.12). 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.

From Figure 4.13, 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



(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).

Figure 4.12: 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.

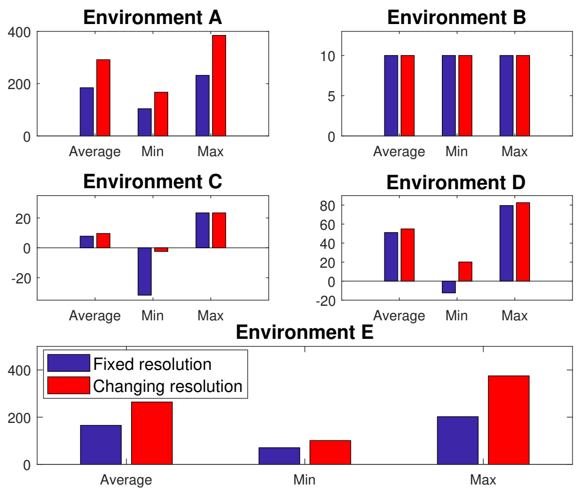


Figure 4.13: Fix resolution vs. changing resolution. From environment A to E, the planning horizon for each steps are 20, 5, 15, 20, 30.

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.

4.5.4 Gazebo Experiments

The previous simulation results show the proposed algorithm can be applied in the visibility-based reconnaissance problem. However, some of the assumptions made for 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 [55].

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.

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 is 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 until one of them completes it. That is, the agent and the opponent use their respective current positions to choose their actions. They continue to execute this action until one of them completes the action.

Figure 4.14 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,

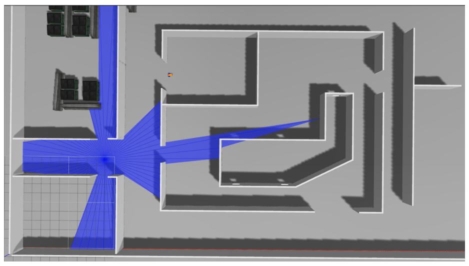


Figure 4.14: Gazebo simulation environment. The agent and the opponent are simulated as differential drive robots equipped with a 360-degree lidar scanner to generate the visibility polygon (we only plot one robot's lidar scanner in blue).

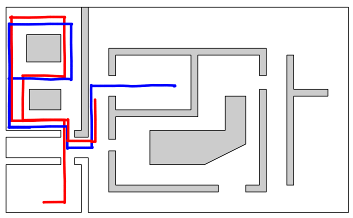
we take the agent and the opponent's goal positions as the input. After the agent reaches its current goal position, MCTS is terminated, and the agent will execute the best action generated by the MCTS.

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 4.15 and the attached video.[4] 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.

4.6 Summary

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 evaluating the performance of pruning techniques using branch-and-bound and the principle

[4]`https://youtu.be/_UuawB8CZ-E`


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

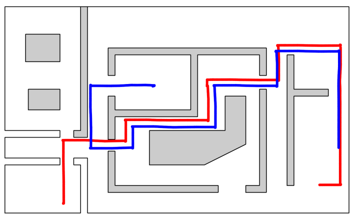

(b) With a lower penalty $P = 3$.

Figure 4.15: 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.

of optimality to 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. There has been recent progress in Monte Carlo Tree Search methods for planning with partial observability [93]. Nevertheless, partial observability remains a computationally challenging problem.

Chapter 5: Planning in Adversarial Environments with Asymmetric Information

In the previous chapter, we studied the problem of adversarial planning in the symmetric setting, i.e., when the robot and the adversary have the same evaluation functions. In this chapter, we will continue focusing on the reconnaissance mission. However, we will investigate an asymmetric scenario where we study what happens when one player has more information than the other.

In this chapter[1], we extend the adversarial reconnaissance mission to a setting where our agent of interest can have more information than its opponent about features of the environment that affect their rewards, and the adversary can become aware of (some of) these features depending on which states the environment passes through over the course of the agent's interactions. Specifically, there are stationary, high-value assets in the environment. Thus, while our agent knows of the existence of the assets and where they are in the environment, the adversary only becomes aware of their existence when/if it sees them in its line-of-sight. The adversary can *collect* an asset by visiting the location of that asset once it becomes aware of the asset's location. Because of the high value of the asset, the agent would prefer that the assets do not get collected. This is in addition to the objectives that we already considered in Chapter 4, i.e., scanning the environment and avoiding detection by the adversary. This makes the problem asymmetric: while both the agent's and

[1]This chapter is a collaboration with Rob Brady, Edmund H. Durfee, Jonathon M. Smereka, and Pratap Tokekar.

the adversary's objective account for the area scanned and detection penalty for the agent, only the agent's objective function accounts for the assets being collected, at least initially. Furthermore, the objective for the adversary is also dynamic: once an asset gets revealed, the objective for the adversary will be updated to include a term for collecting that asset. The agent can anticipate which moves of the adversary can lead to the revealation of the assets leading to an extra level of optimization in its planner.

In Chapter 4, we applied minimax search [87], a commonly-used technique to plan in a turn-based game. As a symmetric game, the agent and the opponent are respectively maximizing and minimizing the same reward function. In this chapter, we apply our new algorithm, DM1, for the above adversarial asymmetric information planning problem. We investigate the effect of the DM1 algorithm and present qualitative and quantitative results where DM1 with an accurate opponent model outperforms minimax by exploiting the asymmetry. In application, we show that the DM1 paths successfully manage to *mislead* the opponent into following the agent and thereby prevent the opponent from discovering important information (assets) in the environment.

The rest of the chapter is organized as follows. We first introduce the related work in Section 5.1. Then, we formulate the asymmetric information with non-stationary opponent evaluation function problem in Section 5.2. We then present the background on an existing algorithm, M* [25], that we build on, in Section 5.3. In Section 5.4, we present the DM1 algorithm for the problem presented in Section 5.2. In Section 5.6 we investigate the effect of the DM1 algorithm and present qualitative and quantitative results where DM1 with an accurate opponent model outperforms minimax by exploiting the asymmetry. In the end, we summarize this chapter in Section 5.7.

5.1 Related Work

The application follows the adversarial games in Chapter 4 and builds on the visibility-based routing problems [2, 68, 75, 85, 90, 110].

Our new algorithm is an extension of the M* algorithm, which was first proposed by Carmel and Markovitch [24] in 1993 for solving asymmetric games. The M* algorithm starts with the M^1 algorithm, which operates with two reward functions, one for the agent and one for the agent's model of the opponent's reward function. M^2 operates with three reward functions, where the agent uses its own objective and models the opponent as using the M^1 algorithms (M^1 is with two reward functions), and so on. M^* is defined as the algorithm that includes every M^n algorithm as a special case. They prove that the strategy returned by M* cannot be worse than the minimax strategy when the opponent model is accurate in a turn-based game. This belongs to the general class of recursive modeling of opponents [40].

M^* (also called as opponent model search in some papers) was further investigated by various game researchers. Carmel et al. [25] further proved a sufficient condition for a pruning strategy and presented the $\alpha\beta^*$ algorithm, which returns the M^* value of a tree while searching only necessary branches. Gao et al. [36] studied a generalization of opponent model search by considering that two players may plan using trees with different depths, called $(D,d)-OM$ search, where D stands for the depth of search by player 1 and d for player 2's depth of search. Donkers et al. [33] studied an extended model that includes the uncertainty of the opponent's state, called the probabilistic opponent model search (PrOM).

Our algorithm is a generalization of M^1, which is a special case of M^*. The key difference between our work and the previously mentioned work is that we account for the dynamically evolving of the opponent model. We will discuss more details about the difference in Section 5.4.

5.2 Problem formulation

We study the problem of planning in a two-player, alternating-turns game, where two players sequentially act on their joint state to generate a sequence of actions through the state space. One player is trying to maximize the final reward while the other player acts as a minimizer. Each player may have different evaluation functions. Each player acts rationally based on its own evaluation function and the evaluation function it ascribes to the other player.

In the symmetric stationary version of this problem, player 1 believes both players use the same evaluation function. Player 1 uses this function to compute values of actions to identify its move to maximize value, where (lacking any other information about its opponent) it predicts its opponent's moves will be to minimize value. The widely-known minimax search algorithm solves this problem.

In our problem of interest, however, player 1 believes player 2 has a different evaluation function. Without loss of generality, we assume player 1 has more information about its opponent, player 2. It knows (or at least believes it knows) the evaluation function that player 2 is using at the outset. Player 1 considers player 2's initial evaluation function inferior to its own (otherwise, it would replace its own function with the one it ascribes to player 2). Furthermore, player 1 (believes it) knows how and when player 2 can improve its evaluation function over the course of states reached during this game.

We define this problem in a generalized framework that incorporates changing opponent models into an asymmetric information game. We assume that the cumulative effects of gaining information relevant to player 1's reward model are summarized in player 1's evaluation function.

Definition 5.1. *The state of the game is represented by a pair $\langle s, e \rangle$ where s is the joint state of the shared environment that players 1 and 2 are in, and e is the evaluation function that player 1 ascribes to player 2.*

We define the initial state as $\langle s_0, e_0 \rangle$. Player 1 models the effects of actions that it and its opponent (player 2) take a transition function

$$\langle s_t, e_t \rangle \xrightarrow{a_t^1, a_t^2} \langle s_{t+1}, e_{t+1} \rangle,$$

where, $a_t^1 \in \mathscr{A}_1, a_t^2 \in \mathscr{A}_2$ are the actions of player 1 and player 2 at time t. Here, s_{t+1} is the resulting joint state of the environment arising from taking action a_t^1, a_t^2 in state s_t, and e_{t+1} is the (possibly different) evaluation function that player 1 believes the opponent will adopt after the action a_t^1, a_t^2 is taken.

Intuitively, when an action leads to a new state, that state might reveal new information to player with less knowledge (player 2), leading player 2 to update its evaluation function. For example, the action leads to a state where a previously unknown (to player 2) asset becomes visible. We make no claim in this case that e_{t+1} is superior to e_t, although we expect a rational player 2 to improve its evaluation function upon acquiring new information generally.

We define the evaluation functions as a partially-ordered set (poset), with a single maximum element, which is player 1's own evaluation function (which must be the best function player 1 knows about), and a single minimum element which is the least-informed function that it believes possible.

Formally, the evaluation-function poset has $k+1$ elements e_0 (minimum element) through e_k (maximum element), where, when a pair of functions (e_i, e_j) is ordered, $i < j$ means the information exploited by e_j contains the same information exploited by e_i or subsumes that of e_i . For example, in our reconnaissance application, an evaluation function with knowledge of subset P of the assets is ordered (possibly transitively) above the evaluation functions with knowledge of strict subsets of P. The poset defines a directed graph over the evaluation functions, rooted at e_0 and converging to e_k, where the sequence of evaluation functions in any state trajectory traverses some path in that directed graph.

To summarize, the problem we solve is that of computing an optimal next move for player 1 (the player with more knowledge) in an asymmetric game, from state $\langle s_t, e_t \rangle$ given a transition model for how actions affect the physical state s_t and the opponent's evaluation function e_t, where the transition model for the evaluation function adheres to the directed graph (monotonicity) defined by the poset.

5.3 Background: Asymmetric information game with a static model (M^*)

Before we describe our new algorithm, DM1, we go into more details about the M* algorithm [25] that DM1 is based on. We will highlight how M* (or, more specifically, a particular version of M* called M^1) cannot handle the non-stationary case described in Section 5.2.

Carmel and Markovitch [24] introduced a way to give players different models in turn-based sequence game playing called M* search. M* is a generalization of the minimax algorithm that can handle an opponent's model, which differs from player 1's model. The M* algorithm uses different evaluation functions at a different level of a search tree, for both player 1 and one for player 2. The M* algorithm is a generalization of the M^n algorithm and starts with the M^1 algorithm, which operates with multiple reward functions, M^1 means player 1 acts rationally based on evaluation function f_1, and assumes player 2 plans with minimax using evaluation function f_0 (minimax can be considered as M^0). M^2 assumes player 1 acts rationally based on evaluation function f_2, and assumes player 2 plans with M^1 using evaluation functions f_1, f_0. M^n is about player 1 acts rationally based on evaluation function f_n, and assume player 2 plans with M^{n-1} using evaluation functions $f_{n-1}, f_{n-2}, \cdots, f_0$.

Note that in the M^*, $\{f_1, f_2, \cdots\}$ are different from the evaluation functions $\{e_1, e_2, \cdots\}$ referred to in Section 5.2. In our case, $e_1, e_2, \cdots$ change over time as new information is revealed. In M^*, f_1 is the evaluation function about "how player 1 thinks player 2 thinks", and f_2 is the evaluation function to model "how player 1 thinks player 2 thinks player 1 thinks", and so on.

The M* algorithm expands the search tree similarly to the minimax tree. However, the minimax search can be considered as M^0, which will only use the reward value computed by one evaluation function. In a general M* search tree, each node will store the reward values computed in the current node according to player 1's evaluation function and the evaluation functions it ascribed to player 2.

However, M* cannot directly be applied to the problem we introduced in the previous section. In our case, player 2's evaluation function for a given joint state is a function of the current joint state and the past joint states leading to the current one. Even if two nodes are at the same level of the search tree, player 2's evaluation function could be different in these nodes because of the different history of joint states. In our problem, player 2 always uses minimax and will not use $M^n, M^{n-1}, \cdots, M^1$ to play against the behavior of player 1 throughout the planning horizon. Our presented algorithm is a generalization of M^1 where player 2's evaluation function changes dynamically throughout the planning horizon. We thus call our algorithm dynamic M^1 (DM1).

5.4 DM1 Algorithm

Prior work on minimax and M* suggest a solution to our stated problem. Certainly, player 1 with superior knowledge could ignore its adversary's weaker (and possibly evolving) evaluation function and run standard minimax using its own (e_k) evaluation function. This maximizes its minimum reward, assuming that the opponent is fully informed, but does not take advantage of it knowing of the opponent's weakness and thus is not optimal with respect to the player's knowledge. Similarly, player 1 could directly use M^1 given that it knows the adversarial player's weaker initial evaluation function e_0, thereby taking advantage of its superior knowledge, but this overlooks its additional knowledge that the adversary's evaluation function can change depending on what states the environment passes through, and thus this solution method will, in general, underestimate the adversary's future decisions. In the rest of the section, we introduce the new algorithm to handle the scenario where the opponent's model could change dynamically.

5.4.1 Expanding the Search Tree

Similar to the minimax search tree, we build a search tree to list all possible actions of player 1 and its opponent, player 2. Each node stores the joint states of the players and a set of evaluation functions used in the trajectory of states from the root to the current node. In our application example, we also store the history of the joint state since they are also needed to compute the reward value.

Root node: The root node contains the current joint state of player 1 and player 2, and the evaluation function of these two players. Without loss of generality, we assume that the root node is a MAX node.

MAX level: The MAX (i.e., player 1) level expands the tree by creating a new branch for each action that can be taken in player 1's state in its parent node from the previous level (which is a node at MIN level). We will update player 1's state based on the available actions a_t^1 at step t.

MIN level: The MIN (i.e., player 2) level expands the tree by creating a new branch for each action that can be taken in player 2's state in its parent node (always a MAX level node). Player 2's state is updated at each level based on the various actions a_t^2 at step t. We also record all the evaluation functions that each MIN node will use based on each node's current and past joint state as we expand the search tree. Recall that the evaluation functions form a poset which correspond directly to a Directed Acyclic Graph (DAG). Therefore, a trajectory of joint states from the root node to any node in the tree will correspond to a path (of evaluation functions) in the DAG. We add this evaluation function to the set, which stores the history of the evaluation functions from its parent if it is not in the set.

Terminal nodes: A terminal node is always the child of a MIN node, without loss of generality. When the tree is fully generated (i.e., both player 1 and player 2 reach the finite planning horizon H), we compute the reward values of the terminal node using all the evaluation functions stored at the node (i.e., all evaluation functions used by the opponent along the path from the root, as shown in Figure 5.1). For example, if player 2's evaluation function was e_0 initially, then changed to e_1 at a terminal node, we will compute and store the reward value computed by e_0 and e_1, respectively. While for another terminal node, player 2's evaluation function remains the same as the root node, then we only need to compute and store the reward value computed by e_0. Note that with a planning horizon H, player 2 can use at most $H+1$ different evaluation functions at each terminal node. Note that e_k, as the evaluation function of player 1, always needs to be applied to compute the reward value.

Figure 5.1 shows an example of the expanded search tree for a planning horizon of 2.

Algorithm 7: The DM1 search.

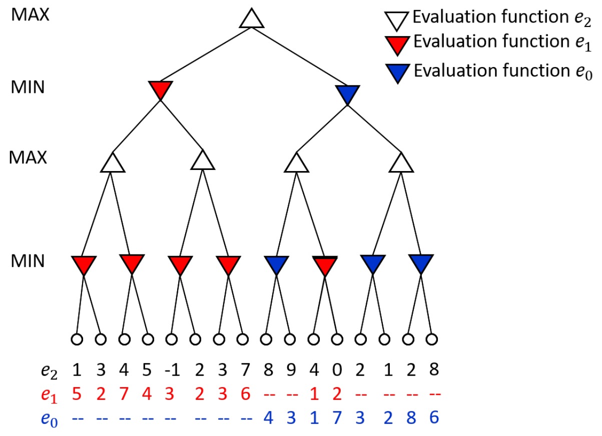
```
function DM1($(node, depth, Node\_state, Joint\_state\_history, Evaluation\_state\_history)$)
    if $Node\_state$ is terminal then
        // Compute the reward value using evaluation function $e_k$
        $r_k \leftarrow Compute\_reward(e_k$,node$)$
        for $j$ from 0 to k-1 do
            if $e_j$ in $Evaluation\_state\_history$ then
                // Compute the reward value using $e_j$
                $r_j \leftarrow Compute\_reward(e_j$,node$, Joint\_state\_history)$
            else
                $r_j \leftarrow null$
            end
        end
        return $\{r_0, r_1, \cdots, r_k\}, null$
    else if $Node\_state$ is $MAX$ then
        return $DM1\_MAX(node, depth, Joint\_state\_history, Evaluation\_state\_history)$
    else
        return $DM1\_MIN(node, depth, Joint\_state\_history, Evaluation\_state\_history)$
    end
end
```

Algorithm 8: The MAX level policy in DM1 search.

```
function DM1_MAX($(node, depth, Joint\_state\_history, Evaluation\_state\_history)$)
    $\{r_0, r_1, \cdots, r_k\} \leftarrow \{-\infty, -\infty, \cdots, -\infty\}$
    Expand the search tree with player 1's actions
    for each child node $v$ do
        Joint_state_history.push_back(joint state of $v$)
        $\{r'_0, r'_1, \cdots, r'_k\}, action\_of\_v \leftarrow$
          DM1$(v, depth-1,$ MIN$, Joint\_state\_history, Evaluation\_state\_history))$
        if $r'_k > r_k$ then
            $action \leftarrow move\_to\_v$
        end
        $\{r_0, r_1, \cdots, r_k\} \leftarrow \{\max\{r_0, r'_0\}, \max\{r_1, r'_1\}, \cdots, \max\{r_k, r'_k\}\}$
    end
    return $\{r_0, r_1, \cdots, r_k\}, action$
end
```

Algorithm 9: The MIN level policy in DM1 search.

```
function DM1_MIN($node, depth, Joint\_state\_history, Evaluation\_state\_history$)
    // Check the evaluation function player 2 uses.
    $e_l \leftarrow$ Check_Eva(node)
    Evaluation_state_history.push_back($e_l$)
    $\{r_0, r_1, \cdots, r_k\} \leftarrow \{+\infty, +\infty, \cdots, +\infty\}$
    Expand the search tree with player 2's actions
    for each child node $v$ do
        Joint_state_history.append(joint state of $v$)
        $\{r'_0, r'_1, \cdots, r'_k\}, action\_of\_v \leftarrow$
          DM1($v, depth - 1$, MAX, $Joint\_state\_history, Evaluation\_state\_history$))
        $\{r_0, r_1, \cdots, r_{k-1}\} \leftarrow \{\min\{r_0, r'_0\}, \min\{r_1, r'_1\}, \cdots, \min\{r_{k-1}, r'_{k-1}\}\}$
        if $r_l > r'_l$ then
            $r_k \leftarrow r'_k$
            $action \leftarrow move\_to\_v$
        end
    end
    return $\{r_0, r_1, \cdots, r_k\}$, $action$
end
```



MAX
MIN
MAX
MIN
Evaluation function e_2
Evaluation function e_1
Evaluation function e_0
e_2 1 3 4 5 -1 2 3 7 8 9 4 0 2 1 2 8
e_1 5 2 7 4 3 2 3 6 -- -- 1 2 -- -- -- --
e_0 -- -- -- -- -- -- -- -- 4 3 1 7 3 2 8 6

Figure 5.1: Compute that all the reward values of the terminal node based on the number of all possible evaluation functions. In this figure, we present an example with three evaluation functions (e_0, e_1, e_2).

5.4.2 Backing Up Values

The reward values are backed up from the terminal node to the root node. In the symmetric minimax algorithm, the minimax policy chooses an action that maximizes (MAX level) or minimizes (MIN level) the backed up reward. In asymmetric DM1, the reward values are backed up in a different fashion as given in Algorithm 7, Algorithm 8 (MAX level), and Algorithm 9 (MIN level). We use a simplified way of keeping track of the backed up values for ease of exposition. Specifically, in the following, we just keep track of a fixed length of $k+1$ evaluations instead of a variable sized set of at most $H+1$ values. The entries corresponding to the evaluation functions that are not used are simply marked as null. However, in practice, one could use a set of variable lengths (depending on the relative sizes of k and H) for efficiency purposes.

MAX level: For the nodes at the MAX level, the DM1 policy chooses an action that maximizes based on the reward computed by the evaluation function of player 1. Since player 1 assumes it has superior knowledge, the action returned in the MAX level is based on reward computed by player 1's evaluation function value e_k. Then, in this node, we store the rest of the k reward values computed by e_0, e_1 to e_{k-1} by taking the maximum reward value computed by each evaluation function in all of its child nodes. Specifically, let r_i be the reward value computed by evaluation function e_i. The node at the MAX level will compare all the reward values computed by the evaluation function e_i in all its child nodes and store the maximum value as r_i in the MAX level. Note that if the evaluation function e_i is not applied in one of its child nodes, r_i in that child node will not be used.

MIN level: We have already determined which evaluation function player 2 is using when we expand the search tree for all nodes at the MIN level. Suppose the node at the MIN level uses evaluation function e_j. In that case, the node at the MIN level will select an action to go to the child node which contains the minimum reward value computed by the

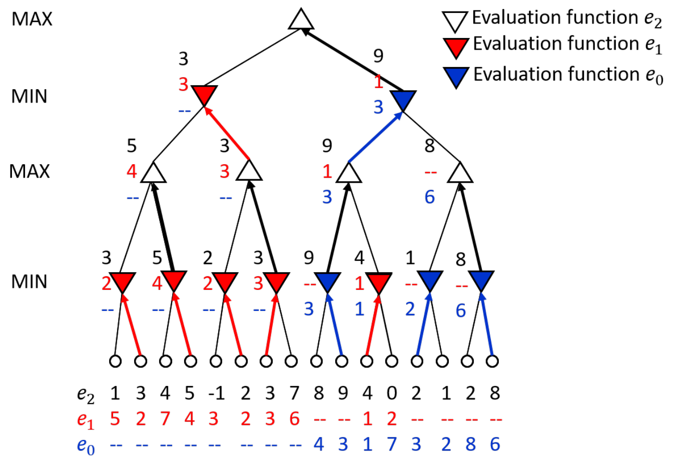

MAX
MIN
MAX
MIN
Evaluation function e_2
Evaluation function e_1
Evaluation function e_0
e_2 1 3 4 5 -1 2 3 7 8 9 4 0 2 1 2 8
e_1 5 2 7 4 3 2 3 6 -- -- 1 2 -- -- -- --
e_0 -- -- -- -- -- -- -- -- 4 3 1 7 3 2 8 6

Figure 5.2: Reward value backed up by asymmetric cases with the dynamic model (DM1). See Figure 5.3 for the minimax version.

evaluation function e_j. We also back up the reward for player 1 (i.e., r_k) stored in that child node. For all other reward values, we will back up the minimum of the rewards of the child nodes (i.e., for all $r_i, i \neq k$, we will back up the minimum value of the reward corresponding to e_i stored in the children). Also, if one of the evaluation functions e_i is not applied in that child node, r_i can be backed up as a empty element.

Figure 5.2 presents an example of the two-step DM1 search tree finding an optimal solution assuming the opponent is doing its best with its evaluation function. We consider the case where the opponent could have three evaluation functions.

In symmetric minimax, only one reward value in the terminal nodes will be backed up to the root node as the optimal move. After considering the opponent's model, the DM1 algorithm will compute the reward by player 1's model (e_2, Figure 5.2), and the opponent has two possible models (e_1 marked with red and e_0 marked with blue). At the MIN level (player 2) nodes, the node chooses the action that minimizes the reward based on the value computed by its current evaluation function (e_0 or e_1). At the MAX level (player 1) nodes, player 1 takes the action that maximizes the reward computed by e_2.

Figure 5.3 shows an example if we ignore the asymmetric information of e_0, e_1 and

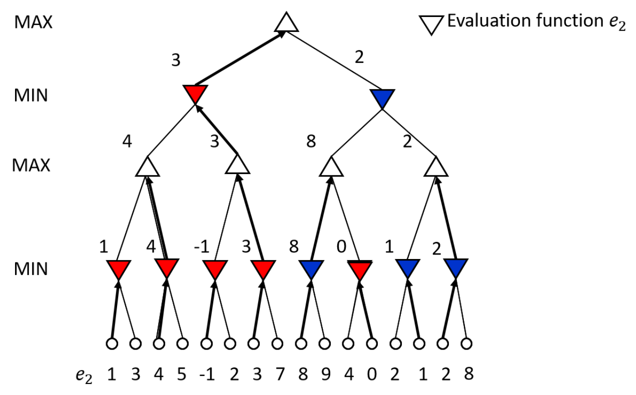

MAX
MIN
MAX
MIN
Evaluation function e_2
3
2
4
3
8
2
1
4
-1
3
8
0
1
2
e_2 1 3 4 5 -1 2 3 7 8 9 4 0 2 1 2 8

Figure 5.3: Reward values backing up using a minimax tree, assuming player 2 uses the same evaluation function as player 1 (e_2). The minimax value returned (3) is less than that with DM1 (9) as shown in Figure 5.2.

directly apply e_2 to player 2 in the minimax tree. Player 1 will overestimate the reward backed up in the MIN level, eventually, choose a different action at the root node, and get lower reward values compared to $DM1$.

5.4.3 Properties of DM1

The agent's policy in DM1 comes from a best-response reasoning. The agent does not know the actual policy that the adversary is going to use. However, the agent can infer a policy a *rational* adversary will use from any state $\langle s_t, e_t \rangle$. As described in Section 5.2, e_t is the evaluation function that the agent believes the adversary will use to plan its own actions from state s_t. Therefore, a rational adversary will choose an action that minimizes the rewards as evaluated by e_t assuming that the agent will choose its own action to maximize the rewards evaluated by e_t. That is, a rational adversary will choose an action that is returned by employing minimax using e_t with s_t as the root node planning up to the rest of the horizon H. Thus, the best-response policy for the agent is to choose an action that will maximize the rewards given by e_k assuming that the adversary will choose an action that is returned by minimax (for the rest of the horizon) using e_t as the evaluation function.

Note that the policy rooted at a node $\langle s_t, e_t \rangle$ and at a node $\langle s_t, e'_t \rangle$ would generally be different: even though the physical state of the agents is the same at both roots, the different evaluation functions can lead to different policies. As a result, the minimax policy used by a rational agent from a node that stores the state $\langle s_t, e_t \rangle$ may not be the same as that used in another node $\langle s_{t'}, e_{t'} \rangle$. Consider our adversarial reconnaissance application. The minimax policy the agent thinks a rational adversary will use at the root node when the adversary is not aware of any assets may be different from the policy the agent thinks a rational adversary will use from a successor node after one or more assets have been revealed. This is in addition to the fact that the depth of the minimax search employed by the rational adversary will be different for nodes at different levels.

In DM1, the agent always chooses an action corresponding to a child node with the largest r_k value. Thus, to show that DM1 is a best-response to a rational adversary, we need to show that the r_k values at each MIN node are correct.

Lemma 3. *Consider any node, A, with state $\langle s_t, e_t \rangle$ in the DM1 search tree. Let $\mathscr{T}$ be the subtree rooted at A. If e_i is one of the evaluation functions that is used by a node lying on the path from the root to A, then the r_i value returned in A equals the value returned by running minimax in $\mathscr{T}$ using the evaluation function e_i.*

Proof. First consider the case that $i \neq k$. Note that since A or some predecessor of A uses the evaluation function e_i, all the terminal nodes in $\mathscr{T}$ will compute r_i using evaluation function e_i. Therefore, the r_i values will not be null for any node in $\mathscr{T}$. When $i \neq k$, DM1 always returns the minimum and maximum of the r_i values of the children for a MIN and MAX node, respectively (Algorithm 9 and Algorithm 8). Thus, the r_i value returned at A will be the same as that returned by running minimax using e_i.

Now consider that $i = k$. For a MAX node, DM1 will always return the maximum r_k value amongst the children of that node. For a MIN node, DM1 will return the r_k value of the child node with the minimum r_t. However, since $i = k$ and because of our assumption that e_k is the singular maximal element of the poset, we will have $e_t = e_k$. Therefore, even in this case the $r_{i=k}$ value returned at A will be the same as that returned by running minimax using $e_{i=k}$. $\square$

Theorem 8. *DM1 correctly finds the best-response strategy for the agent against a rational adversary that chooses actions using the evaluation function e_t from any state of the game $\langle s_t, e_t \rangle$.*

Proof. Consider a MIN node with state $\langle s_t, e_t \rangle$. From Lemma 3, we know that the r_t value returned at this node corresponds to the optimal minimax value using e_t rooted at this node. Therefore, the r_t value comes from a child node that has the minimum r_t value. Now, the r_k value that will be returned by this node also comes from a child node that has the minimum r_t value. Therefore, at MIN nodes the value of r_k returned corresponds to the actions that will be chosen by a rational adversary.

Note that it is possible that there are multiple nodes that have the same minimum r_t value. In the absence of any other information, any of these children nodes are equally likely to be chosen by the adversary. Irrespective of how the ties are broken, there exists a rational adversary that will choose the same child node as that chosen by DM1 to back up the r_k value.

For MAX nodes with state $\langle s_t, e_t \rangle$, we always return the r_k value as the maximum of the r_k values of the children nodes. The action chosen by DM1 for the agent corresponds to the child node that has the maximum r_k value. As a result, DM1 always chooses an action for the agent that is the best-response strategy to maximize the rewards obtained by e_k against a rational adversary. □

In the following section, we show how DM1 can be used in the asymmetric reconnaissance mission that motivates our work.

5.5 Application: Planning in Reconnaissance Mission Against Adversary with Asymmetric Information

This section evaluates the online DM1 tree search algorithm in the context of a reconnaissance task. There may be multiple stationary assets in the environment. We focus on scenarios where the adversary is not initially aware of the presence of the assets.

Consider an agent (represented by player 1) and an opponent (player 2) moving on a graph that represents the environment. The graph could be a grid or a roadmap constructed from the environmental map. Each node in the graph has associated with it a reward value. The positive reward is dependent on the application scenario. For example, for a reconnaissance mission, the positive reward can be a function of the new area visible from that node. For a patrolling mission, the positive reward could depend on when the area visible from the node was last covered. We also have negative rewards (or penalties) when the agent is visible to the opponent in a position not seen before and when the assets are collected by the opponent (see Figure 5.4). We do not penalize if the opponent detects the agent at a previously seen position. This is to model the cases where new information is only revealed when observing the agent for the first time at a location. In our problem setting, we abstract the low-level details of how the agent gathers information and other physical, low-level actions the agent may take at a given location. By observing the agent at a new location, the opponent may be able to gain additional tactical information about the agent specific to that location. While we abstract the details of this, we model this in the form of

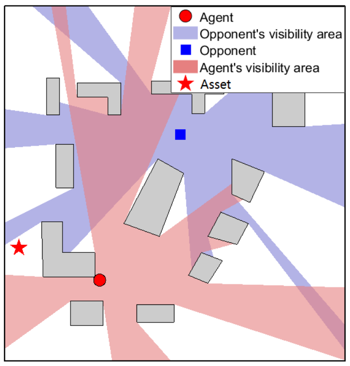


Figure 5.4: Visibility-based reconnaissance: A negative penalty will be added if the agent is inside the opponent's visibility polygon (i.e., the blue) and in a position that was not being detected before. In a reconnaissance mission, the area of the agent's visibility polygon (i.e., the red) is considered as a positive reward. An asset (red star) is in the environment, and only the agent knows of the asset's existence (at least initially). Once the opponent sees the asset, it will wise up and consider the asset's effect in this game. And the agent will be penalized once the opponent collects the asset.

a one-shot penalty for observing the agent at a new location. Other forms of penalty could also be used, such as penalty only for observing the agent at specific locations or penalty every time the agent is observed irrespective of the history. However, the latter makes the results more sensitive to the choice of time horizon since the total and unrestricted penalty could far outweigh the positive reward.

We make the following assumptions: (1) The map is graph-based, and the agent and the opponent move by taking turns on the graph at discrete time steps. (Note that in reality, they move simultaneously, but we model this as a turn-based game divided into discrete time steps.) (2) Both the agent and the opponent know the environment and the position of all obstacles. (3) There is no uncertainty in the agent's and opponent's actions. (4) Both the agent and the opponent know each other's position. (5) There exist n assets in the environment, and only the agent knows of the existence and position of the assets initially. (6) The opponent will update its evaluation function immediately upon obtaining a line of sight of one or more assets.

Assumption (4) is the same as assumption (7) that we make in Chapter 4. Like before, we envision scenarios where external sources (such as aerial vehicles or other sensors) may be able to give positional information about the agents. However, the agents need to detect each other from closer ranges to actually reveal information, which is what results in the penalty. An important line of future work is to relax this assumption using for example, information states [81, 82].

The evaluation function for the agent for time step t is:

$$e_k(\pi_a^t, \pi_o^t) = R(\pi_a^t) - \eta_{agent}(\pi_a^t, \pi_o^t) P_{agent} - \sum_{i=1}^{n} \eta_{asset}^i(\pi_o^t) P_{asset}^i, \tag{5.1}$$

where $R(\cdot)$ denotes the new positive reward collected by the agent at time step t. P_{agent} is the negative penalty for the agent whenever it is visible to the opponent in a place that it was not being detected before. $\eta_{agent}(\cdot)$ is an indicator function to count the total number of times that the agent is observed in a position that was not being detected before, given the agent's path π_a^t and the opponent's path π_o^t at step t. Similarly, $\eta_{asset}^i(\cdot)$ is a indicator function to check whether the i-th asset is collected given the opponent's path π_o^t.

For the opponent, since we assume it does not (at least initially) know of the existence of the asset, its decision model will not consider the effect of the assets,

$$e_0(\pi_a^t, \pi_o^t) = R(\pi_a^t) - \eta_a(\pi_a^t, \pi_o^t) P_{agent}. \tag{5.2}$$

However, as the opponent moves, if it detects some of the assets, it will update its evaluation function by considering the effect of the assets. For example, at state s_t, if a collection of assets $\mathscr{J}$ have been observed by the opponent during the trajectory from s_0, the agent models the opponent's evaluation function as,

$$e_t(\pi_a^t, \pi_o^t) = R(\pi_a^t) - \eta_{agent}(\pi_a^t, \pi_o^t)P_{agent} - \sum_{j \in \mathscr{J}} \eta_{asset}^j(\pi_o^t)P_{asset}^j \tag{5.3}$$

where $\eta_{asset}^j(\pi_a^t, \pi_o^t)$ is the opponent's indicator function to check whether asset j is collected given the opponent's path π_o^t.

The agent's problem is to find its best solution, assuming the opponent will execute what the agent thinks that the opponent thinks are the worst possible actions for the agent. We consider planning for an episode of duration D. Normally, we would use the episode duration D as the planning horizon for DM1 ($H = D$). But when D is large, that may be infeasible. So we solve the next action for the agent using a finite horizon of H $(H < D)$ and replan after every step.

5.5.1 Online Planning with Search Tree

When the planning horizon $H < D$, we use the following strategy. Once the tree is built, the agent can execute the first step of the policy. After the agent executes that step and observes the new position of the opponent, the agent will update the position of the opponent (based on new measurement) in the new root node and rebuild the search tree.

When the planning horizon $H < D$, we can use a heuristic to measure the potential of different terminal nodes to collect the future reward beyond H time steps. The intuition behind the heuristic is: 1) The agent prefers to choose a terminal node that is closer to the unexplored area. 2) The agent prefers a terminal node that is far away from the opponent. 3) The agent prefers a terminal node that is less likely to expose one or more unrevealed assets.

First, we compute and store the shortest path between all pairs of points in the environment. For each unexplored grid location, center a (positive reward) Gaussian pdf at that location. For each uncollected asset i, center a (negative reward) Gaussian probability density function (pdf) at that asset's location. Also, center a (negative reward) Gaussian pdf at the position of the opponent.

Then, given the precomputed distances, we compute the heuristic as a mixture Gaussian by taking the sum of all the Gaussian pdfs at the agent's position. The first term in Equation 5.4 gives the potential positive reward, so for each terminal node, we compute the mixed Gaussian pdf value based on the distance between the agent's position at the terminal node and each unexplored grid point. In other words, if the agent's position in one terminal node is closer to more unexplored regions, the heuristic gives more weight to go in that direction. The second part is a negative Gaussian pdf based on the distance between the agent and the opponent, which drives the agent to move away from the opponent. To measure the potential of revealing the assets, the last part of the heuristic is a set of negative Gaussian pdf based on the distance between the agent and each asset.

For instance, given a Gaussian pdf function $pdf(\cdot)$, and a terminal node with the position of the agent p_a, the position of the opponent p_o, the position of the j assets $p^1_{asset}, \cdots, p^j_{asset}$, and the position of q unexplored grid points $p^1, p^2, \cdots, p^q$, the heuristic computed is this terminal node is,

$$heuristic = \sum_{i=1}^{q} pdf(d(p_a, p^i)) - \alpha \cdot pdf(d(p_a, p_o)) - \beta \cdot \sum_{i=1}^{j} pdf(d(p_a, p^i_{asset})), \tag{5.4}$$

where $d(p_i, p_j)$ returns the minimum distance between points p_i and p_j in a graph-based environment. α, β are the weight parameters.

5.6 Evaluation

In this section, we evaluate DM1 in this asymmetric reconnaissance problem. We assume the visibility range of the agent and the opponent are both unlimited (only restricted by the obstacles in the environment). The software was written in MATLAB R2019a and used the VisiLibity library [73] to compute the visibility polygons.

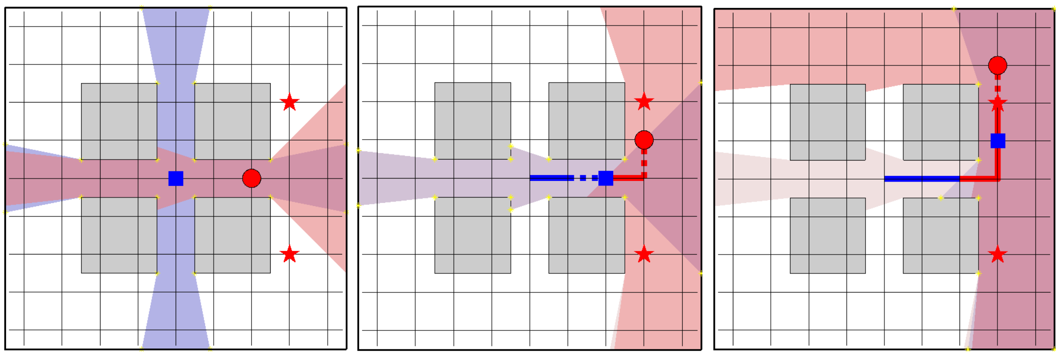
In following section, we first present qualitative examples for DM1 and minimax, as well as the effect of the heuristic. Then, we compare the performance of online DM1 and minimax with different settings.

5.6.1 Qualitative Results

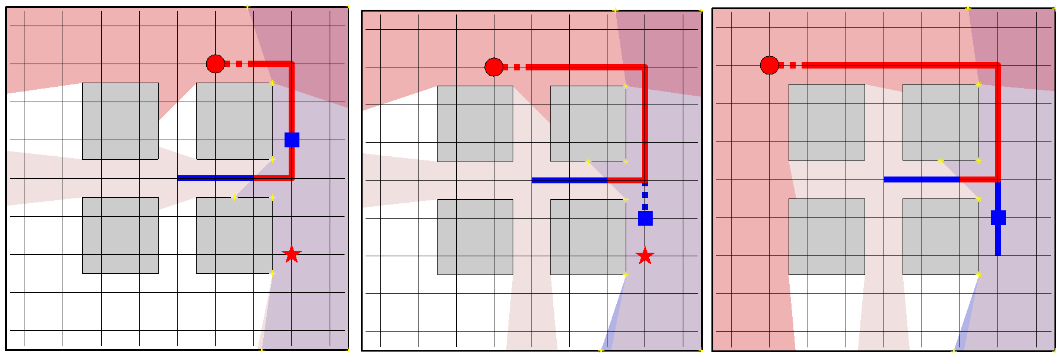
We first show qualitative examples of the policy found by the online DM1 and minimax. We evaluate the performance using accumulated rewards over time. In this qualitative example, the positive reward R is the area scanned by the agent along its path. Once the opponent observes the agent in a position that was not detected before, the agent's penalty will be $P_{agent} = 0.3$. If one of the assets is collected by the opponent, the agent receives a high penalty of 30.

For all the experiments in this section, we will use the heuristic unless explicitly stated. When the heuristic is not applied and if two or more nodes have the same reward value, we apply the sequence of "staying still, moving east, moving north, moving west, moving south" to break ties.

Figure 5.5 and Figure 5.6 are small examples to illustrate the choices being made by the agent using the symmetric minimax and asymmetric DM1 when $H = D = 10$. Figure 5.5 illustrates the result if the agent is planning using minimax. In both cases, the opponent uses a minimax tree to plan its own actions. The evaluation function in the minimax tree is based on the number of assets the opponent is aware of at that time instance. Since this is

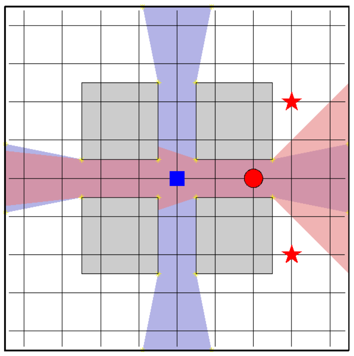


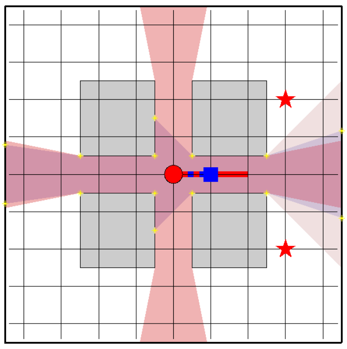
(a) T = 1. (b) T = 3. (c) T = 5.

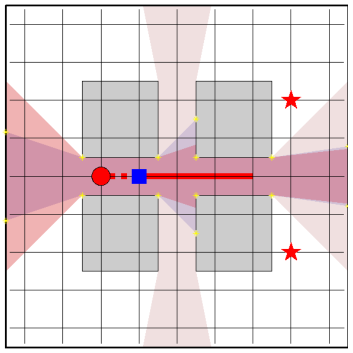


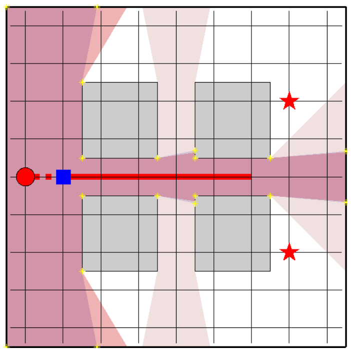
(d) T = 7. (e) T = 9. (f) T = 11.

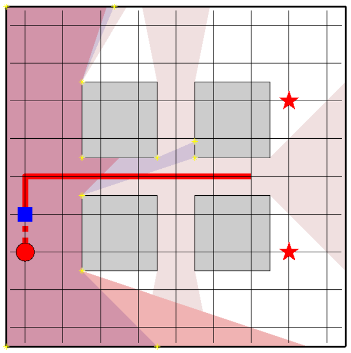
Figure 5.5: Minimax: Using minimax, the agent cannot choose actions to prevent the opponent from detecting the assets. The opponent collects both assets at the end of the episode. From left to right, the figures show the episode when $T = 1,3,5,7,9,11$. The accumulated positive rewards at each step are 26.7, 27.2, 40.6, 41.6, 47.8, 58.7. The accumulated negative penalties at each step are 0, -0.9, -1.5, -1.8, -31.8, -61.8. At the end of the episode, the total reward collected by the agent is -3.1.


(a) T = 1.


(b) T = 3.


(c) T = 5.


(d) T = 7.


(e) T = 9.

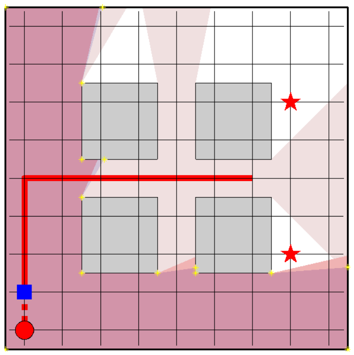

(f) T = 11.

Figure 5.6: DM1: The agent leads the opponent away from the asset, and the agent stands to benefit more by collecting a higher total reward. The accumulated positive rewards at each step are 14.6, 23.3, 26.7, 41.9, 44.8, 52.9. The accumulated negative penalties at each step are 0, -0.9, -1.5, -2.1, -2.7, -3.3. At the end of the episode, the total reward collected by the agent is 49.6.

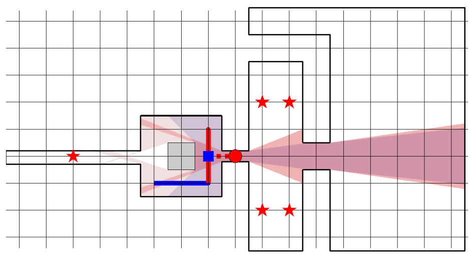
minimax, the agent treats the opponent being as informed as itself. The symmetric planning algorithm ends with all the assets being detected (the agent overestimates the opponent and thinks it could not stop the detection and so only cares to explore more areas). The agent focuses on maximizing its other rewards by directly moving east, and it unwittingly leads the opponent to discover the assets.

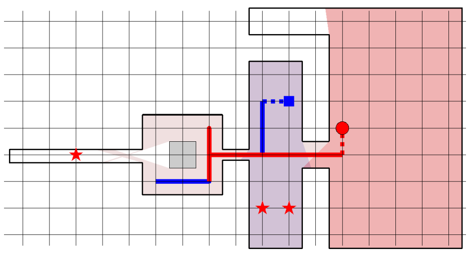
Figure 5.6 illustrates how the DM1 can help the agent to exploit the correct knowledge of asymmetry and the non-stationary opponent's model. When the agent correctly knows the opponent's weaker model, DM1 plans actions by moving towards the opponent, preventing the opponent from discovering the assets. At the end of the episode, the agent using DM1 collects more total reward (49.6 to -3.1) by sacrificing some exploration in order to avoid revealing any of the assets.

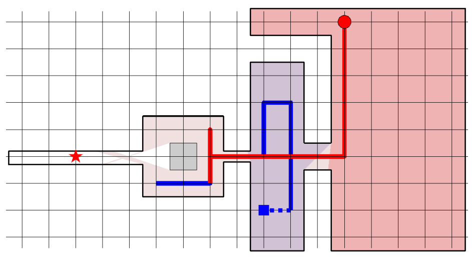
This qualitative example shows how DM1 can take advantage of the asymmetric knowledge. By moving unexpectedly towards the opponent initially, the agent is able to lead the opponent away from the assets that may cause more negative rewards. In contrast, minimax overestimates the opponent and moves away from the opponent. Despite the fact that the agent is detected fewer times by the opponent, it eventually exposed all the assets. The final summed reward collected by minimax is less than the reward collected by DM1.

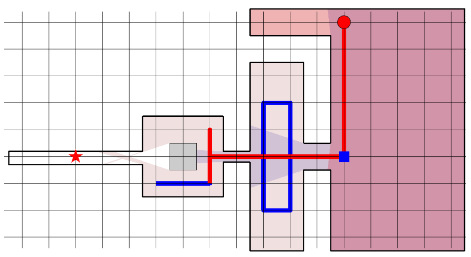
Another qualitative result in Figure 5.7 and Figure 5.8 illustrates how DM1 can help the agent to exploit the correct knowledge of asymmetry by deliberately exposing one asset in the corner to create separation from the opponent, and eventually collect more reward by taking advantage of that separation. In this example we set $H = 13$ and $D = 30$.

Figure 5.7 illustrates the result when the agent is planning using minimax. The agent directly moves towards the larger area, which exposes four of the assets. The symmetric planning algorithm ends with four assets being detected because the agent overestimates the opponent and thinks it could not stop the detection. Then the rational moves are to explore the larger area first.


(a) T = 5.


(b) T = 10.


(c) T = 15.


(d) T = 20.

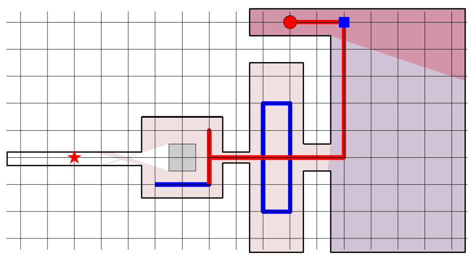
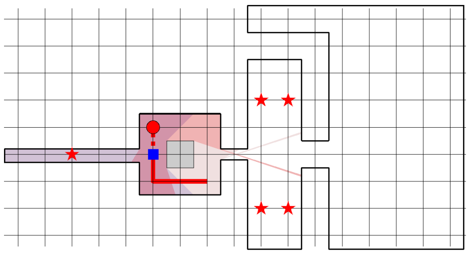
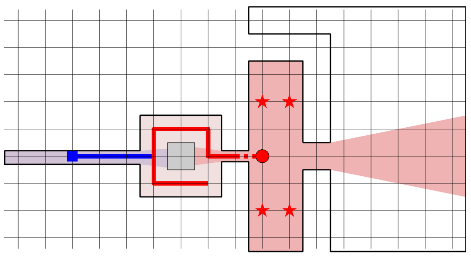

(e) T = 25.

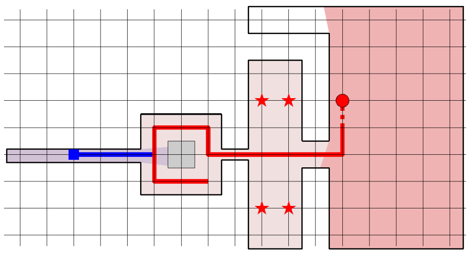
Figure 5.7: Minimax: planning with symmetric minimax, the agent chooses actions directly to explore more areas. The opponent collects four assets at the end of the episode. From left to right. The accumulated positive rewards at each step are 20.3, 68.9, 72.4, 72.4, 72.4. The accumulated negative penalties at each step are -5.0, -68.0, -128.0, -129.0, -131.0. At the end of the episode, the total reward collected by the agent is -58.6.



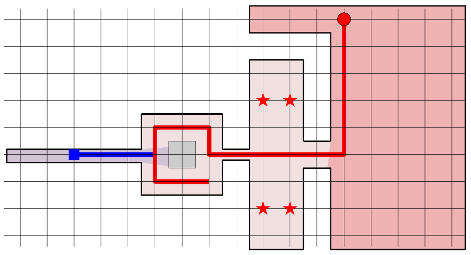
(a) T = 5.



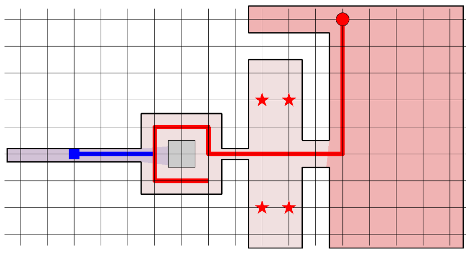
(b) T = 10.



(c) T = 15.



(d) T = 20.



(e) T = 25.

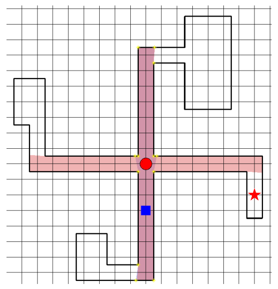
Figure 5.8: DM1: by considering the opponent's dynamic model. The agent deliberately exposes one asset in the corner to create separation from the opponent. The agent can explore a larger area and also avoid exposing four assets by creating this separation. The accumulated positive rewards at each step are 10.9, 36.4, 71.8, 75.2, 75.2. The accumulated negative penalties at each step are -5.0, -35.0, -35.0, -35.0, -35.0, -35.0. At the end of the episode, the total reward collected by the agent is 40.2.

Figure 5.8 illustrates how DM1 can help the agent to exploit asymmetric knowledge. DM1 moves to the narrow corridor first and exposes one asset in the remote corner. Despite the fact that one of the assets gets collected (along with its negative penalty), the agent creates separation between the opponent and can explore the rest of the area without exposing any more assets. This is because, after the opponent using minimax goes to collect the first asset, it finds itself having no chance to catch up with the agent, and thus (unless influenced by heuristics) will choose to stay in place. DM1 can take advantage of that and finds the path to collect the most rewards.

5.6.2 Effect of the heuristic

In this subsection, we use a qualitative result to show the effect of the heuristic. When the planning horizon $H < D$, we can use a heuristic to measure the potential of different nodes to collect the future reward. A heuristic helps break ties for the agent and the opponent and can help bias the agent and the opponent to better intermediate nodes when planning to only a limited horizon.

Figure 5.9 shows a result of the agent planning using DM1, with and without the heuristic when looking ahead only $H = 5$. Figure 5.9-(a) shows the initial position of the assets. By only looking ahead for five steps, the agent cannot detect the new area in the search tree and stays in the initial position because the tie-breaker prefers staying in place (e.g., to conserve energy). In contrast, as shown in Figure 5.9-(b), although the agent cannot detect the new area at an intermediate node when looking ahead only $H = 5$, the heuristic gives more weight on moving north first because it is closer to the largest unexplored area. As we can see, with the help of the heuristic, the agent is able to move to the largest room first then explore the second largest room.


(a) Without heuristic. When $H = 5$, the agent cannot detect any new area in the search tree given the initial position of the agent. Thus, the agent ends up staying

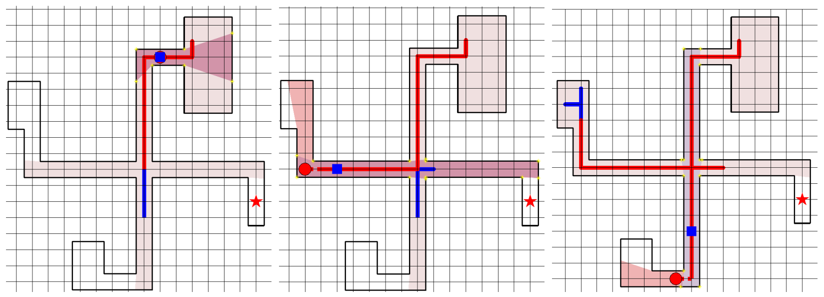

(b) With heuristic. When $H = 5$, despite the agent not detecting any new area within the limited look-ahead search tree. The heuristic value enables the agent to explore the largest room first and then continue to explore the search largest room. The figures show the portions of the agent when $T = 15, 30, 45$.

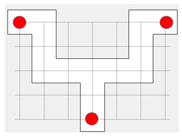
Figure 5.9: Without heuristic vs with heuristic.

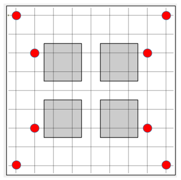
5.6.3 Quantitative Results

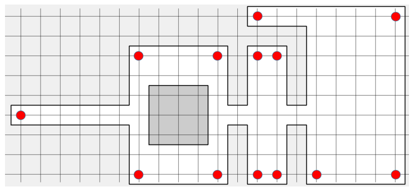
In this subsection, we compare the performance of DM1 and minimax in an online reconnaissance problem with different settings. We use the environments shown in Figure 5.10 to generate the following results.

For the first quantitative experiment, we initially set the episode duration equal to the planning horizon ($D = H = 9$), the penalty for the agent being detected as $P_{agent} = 0.3$, and penalty for an asset being collected as $P_{asset} = 30$. We randomly chose the possible starting locations for the agent, opponent, and asset(s). Also, we make sure the distance between the agent, opponent, and the assets is less than the episode duration (distance between any two of them is less than 9). Table 5.1 shows the comparison of the final reward collected in different environments when $H = D$. We did not need to apply the heuristic for the experiment in this table.

From the experimental results in Table 5.1, when $H = D = 9$, the DM1 algorithm always performs better or finds the same reward value compared to the minimax algorithm.


(a) Environment A (7×5)


(b) Environment B (10×10)


(c) Environment C (23×12)

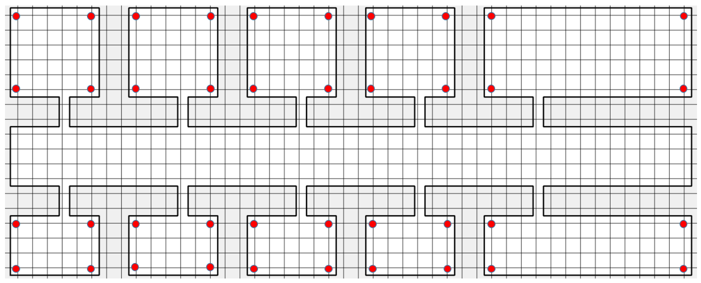

(d) Environment D (50×20)

Figure 5.10: Environments for the quantitative simulations. Red dots are the different initial positions for the assets in the second and third parts of the quantitative experiments.

The results confirm our hypothesis that DM1 improves the agent's performance by taking advantage of the opponent's model's asymmetric and non-stationary knowledge.

In the second quantitative experiment, we set $H < D$, and D is set to be twice the x and y spans of the environment. We remove the condition that the distances between the agent, opponent, and the assets are less than the planning horizon. Instead, we uniformly at random generate the agent and the opponent's positions. The assets' positions are randomly picked from the set shown as the red dots in Figure 5.10. Note that when $H < D$, it is possible that the optimal local solution in H steps is not the globally optimal one. Thus, minimax may collect more rewards compared to DM1. For example, in one of these experiments, DM1 tried to protect one of the assets and stop exploring while minimax continues to explore although the opponent detects the asset. However, the unexplored area that is H steps away can provide a more positive reward compared to the penalty. It is possible that this problem will be alleviated with a better heuristic and longer planning horizons. As we

Table 5.1: Comparison final reward collected in different environments when $H = D = 9$ (30 trials for each environment).

Envir -onment	Num. of Assets	DM1 Strictly Performs Better (%)	Minimax Strictly Performs Better (%)	DM1 vs minimax average reward when DM1 performs better	DM1 vs minimax average reward when minimax performs better
A	1	37%	0	4.5 vs -15.4	–
B	2	53%	0	47.6 vs -2.6	–
C	2	63%	0	51.7 vs 11.3	–
D	2	27%	0	177.4 vs 137.7	–

Table 5.2: Comparison final reward collected in different environments when $H < D$ (50 trials for each environment).

Envir-onment	H, D	Num. of assets	Num. of times, DM1 performs better	Num. of times, minimax performs better	Average reward when DM1 performs better (DM1 vs minimax)	Average reward when minimax performs better (DM1 vs minimax)
B	H=8, D=40	2	42%	6%	37.4 vs 4.3	2.7 vs 5.1
C	H=8, D=70	2	72%	18%	77.4 vs 33.7	9.3 vs 28.4
D	H=8, D=140	8	54%	14%	411.7 vs 203.4	167.5 vs 211.2

can see in Table 5.2, there are some cases where minimax outperforms DM1 when $H \ll D$. Nevertheless, DM1 still improves the agent's performance on average, and it is more likely to see that DM1 strictly performs better than minimax. The results in environment C also suggest that when minimax collects more reward, the gap between the two reward values is less compared to the difference when DM1 performs better. However, we see that as the environment size increases (environment D) without an increase in H, the differences between minimax and DM1 shrink. This is expected as DM1 is unable to take advantage of the asymmetry without looking sufficiently far ahead in that environment.

In the third quantitative experiment, we investigate how different levels of "asymmetric information" affect the performance of DM1. In this experiment, we use the different numbers of assets to represent a different level of "asymmetric information". We use environment C in Figure 5.10, and generate the position of the assets randomly from the highlighted red dots. We consider two scenarios. First where the assets are generated uniformly at random throughout the environment. Second where the assets are generated clustered by adding a constraint that a newly generated asset must be within the visibility region of all the existing assets. We set H, D, and the penalty parameters in the same way that we did for environment C in the previous experiments. The results are included in Table 5.3. We see that DM1 is more likely to outperform minimax when the assets are spread. The gap between the average reward collected by DM1 and minimax increases as the number of assets increases. However, the agent is also more likely to have a worse final reward when applying minimax. When the assets are spread throughout the environment, DM1 has more opportunities to selectively reveal some of them for gaining a greater advantage in the longer run. DM1 can either protect all the assets or cannot protect any of them when assets are clustered. The results suggest that as the level of "asymmetric information" increases, DM1 can better take advantage of the asymmetric knowledge.

Table 5.4 shows the number of nodes generated for the four environments for various values of H. Note that size of the tree will vary depending on the initial position of the agents. DM1 does not use any pruning. Improving the computational time with pruning, for example, is an important avenue for future work.

5.7 Summary

We introduce a new game-theoretic problem of planning in an adversarial environment with asymmetric nonstationary information. The main contribution of this chapter is a tree-search algorithm called DM1 that provably solves this problem. We test this algorithm in

Table 5.3: Comparison final reward collected in environments C with a different number of assets. (50 trials for each case). $H = 8$, $D = 70$.

Num. of assets	DM1 performs better (times)	DM1 performs worse (times)	Average reward when DM1 performs better (DM1 vs minimax)	Average reward when minimax performs better (DM1 vs minimax)
2 (no constraint)	42%	6%	37.4 vs 4.3	2.7 vs 5.1
6 (clustered)	44%	4%	21.4 vs -156.7	-127.4 vs -133.2
6 (spread)	74%	12%	-26.7 vs -116.1	-47.8 vs -32.7
10 (clustered)	52%	8%	13.7 vs -206.4	-177.3 vs -162.4
10 (spread)	78%	10%	-38.4 vs -164.2	-117.3 vs -102.5

Table 5.4: The number of nodes needed to find the action in DM1 with different H.

	Environment	$H = 5$	$H = 6$	$H = 7$	$H = 8$
Num. of nodes generated	A	3.50×10^5	9.17×10^5	6.02×10^6	5.19×10^7
	B	1.87×10^6	3.35×10^7	7.04×10^8	1.47×10^{10}
	C	5.32×10^6	1.38×10^8	2.97×10^9	7.90×10^{10}
	D	7.32×10^6	1.81×10^8	3.95×10^9	9.87×10^{10}

a reconnaissance problem and demonstrate the online planner's feasibility through simulations.

Our experiments show the DM1 algorithm is a promising solution for asymmetric non-stationary agent games. We first use qualitative examples to show how the asymmetric DM1 can outperform symmetric minimax. We then provide quantitative results and show the DM1 algorithm always performs better or finds the same reward value compared to the minimax when the episode duration equals the planning horizon ($H = D$). We also show it is possible that minimax may collect more reward compared to DM1 when ($H < D$), but DM1 still can improve the agent's performance on average.

The immediate future work is to improve the efficiency of the algorithm. Currently, the algorithm is limited to smaller H because of the exponential increase in the tree size. Domain specific characteristics could be used (similar to the problems studied in Chapter 4) to improve efficiency. The algorithm also needs to be evaluated more substantially in larger, more varied environments.

Chapter 6: Conclusions and Future work

6.1 Summary of Contributions

One of the advantages of robots is that they can be used not only for tasks in known and safe areas but also in environments that may have adversaries. The research reported in this dissertation was motivated by scenarios where robots are operating in the presence of adversaries. Our focus was on designing planning algorithms with an overarching goal of understanding how asymmetric conditions between the robot and the adversary affect the outcome of this game.

In general, an adversarial problem can be modeled as a minimax game adversarial planning problem, where we can refer to the robot and the adversary as MAX and MIN players, respectively. At each step, the robot takes actions to maximize the total reward while the adversary moves to minimize the total reward.

Our focus in this dissertation was on settings where it may not be suitable to model the problem as minimax directly. Specifically, we focused on scenarios where the robot and the adversary have different capabilities or information available to them. For example, suppose the robot is an aerial vehicle, and the adversary is ground-based. In that case, it is possible that the robot may have more information about the environment using its onboard sensors than the adversary. The robot may be able to exploit this information in this asymmetric setting. Also, robots and adversaries may have different capabilities, and the player with superior capabilities may be able to exploit them. If the adversary is capable of corrupting the robot's information (while the robot may not be able to do the same to the adversary's information), then this asymmetry can be exploited by the adversary to its advantage. In this dissertation, we make progress towards the understanding of the asymmetry of information and capabilities for robot planning in adversarial settings. Since robots are usually required to plan online, computation efficiency is critical in the planning process. In this dissertation, we also investigated whether we can improve the computational cost by using structure properties of the underlying game.

In this dissertation, we showed how asymmetry in information and capabilities can be exploited by the respective agents. We also showed how the underlying structural properties can be exploited by us to improve computational performance. We used adversarial target tracking and reconnaissance as our underlying application scenarios. In both problems, the robot and the adversary need to plan their actions so as to track or get away from the other agent. In addition, in the reconnaissance mission, the robot also has the additional objective of gathering information about the environment. These problems are representative of many practical scenarios and allowed us to study the effect of asymmetry in terms of information and capabilities, which was the goal of this work.

We first studied the symmetric information game of active target tracking with distance-dependent noise using minimax search in Chapter 2. Our solution consists of building a minimax search tree to obtain the control policy for the robot. To overcome the difficulty of high computational cost, we studied how the structural properties of the Kalman Filter affect the robot's planning process and the outcome of the game.

Our main contribution here was to prove a set of conditions based on algebraic redundancy that allows us to reduce the size of the search tree while preserving optimality. We also presented sub-optimal pruning strategies that can be used to yield even more computational savings at the expense of optimality. Our empirical results show that these pruning strategies lead to search trees with only 1% of the nodes of a full tree without losing optimality. Our main technical result assumes distance-dependent measurement noise (Equation 2.3). We expect that this result (Theorems 2) can be extended to more general state-dependent measurement noise models. The main property that we proved was that the monotonicity of the Kalman Ricatti equation holds even in the distance-dependent case. We expect this property to also hold for a more general state-dependent case. This will allow our result to generalize to more practical sensor noise models, such as when the measurement noise is a non-linear function of the relative positions of the robot and target or depends on the viewing angle.

We then considered an asymmetric case where the adversary can inject false sensing data in the robot's measurement in Chapter 3. Clearly, the adversary can exploit this asymmetric capability and mislead the robot's estimation. However, the challenge here is that if the adversary injects a large magnitude of false sensing data, then the robot would be able to detect it and could possibly reduce the effect of this asymmetric advantage. We showed how the adversary can take advantage of this capability without triggering the robot's false measurement alarm. Our results show that the adversary can achieve any desired deviation in the output of a Kalman filter given sufficient time without triggering the robot's false data detection alarm.

This result suggests the need for better false data detection algorithms for the robot to mitigate such attacks. We focused only on the estimation aspect of the problem. However, it is quite likely that the estimated adversary state is actually used for some higher-level objective, such as planning the motion of the robot to better track the adversary. The effect of such attacks on these higher-level objectives needs to be better studied. However, using a strategy like the one reported in Chapter 2 will lead to the robot being mislead and going away from the true position of the adversary. In such cases, additional sources of information or measurements may be needed to mitigate such attacks.

Chapters 4 and 5 focused on how the availability of information affects a reconnaissance mission. We started with the symmetric information game, where the information available to the robot and adversary is the same. We presented and compared the performance of minimax as well as MCTS for this problem. Our main contributions here are we applying branch-and-bound to this adversarial reconnaissance setting to improve the computational efficiency of minimax and MCTS. We also empirically evaluated the performance of a multi-resolution planning strategy in this setting.

We extended the symmetric case to an asymmetric information game by considering the case where the information available to the robot and the adversary may not be the same. We extended the M^1 algorithm, which was designed for a static case, to a dynamic scenario where the information available to the adversary may change over time. We evaluated this algorithm using the reconnaissance problem and demonstrated the online planner's feasibility through experiments. The results indicate that our algorithm, called DM1, is able to correctly exploit the asymmetric information. In reconnaissance missions, the paths planned with DM1 can protect assets that are unknown to the adversary or selectively reveal them when it is advantageous.

The DM1 algorithm can be used in any setting where the adversary gains more information during the game play and thus will update their model. The key requirement here is that the robot is able to predict (correctly) what model the adversary will be using at any given point. Note that this is different from repeated games where the same set of players are playing multiple games and can improve/change their strategies after each game. Instead, DM1 is applicable for the single-shot game played for a finite horizon, where one player can improve its strategy because it gains information that changes its evaluation function. Applications where information that is hidden to one player gets revealed over time (or due to the players' actions) can be solved via DM1.

6.2 Future Work

Planning for autonomous systems in adversarial environments is still a topic that needs to be studied further. While this dissertation is more focused on the application of tree search techniques, there are other techniques to solve related adversarial problems. For example, resilient target tacking [119], which is to optimize the robots' tracking performance with potential failures, robust control [111], which also plans for the worst-case scenario, and randomized strategies [47], which offers anytime properties, allow real-time implementations for adversarial pursuit-evasion problems in online settings, etc.

The main bottleneck of the tree search techniques is still the trade-off between the computational cost and accuracy. For example, when it comes to scenes with thousands of grid points and the requirement of online planning, it is still difficult to find a feasible solution in real-time, even if we use pruning and heuristics to improve computation efficiency. Thus, one direction is, continue to improve the computational efficiency of the algorithm.

In addition, we assume that the robot and its adversary play a two-player turn-based game. However, the agent and the opponent are moving simultaneously in the real world. This is not an inherent limitation of the tree search algorithms used in this dissertation but rather in their application to the specific problem settings (such as target tracking and adversarial reconnaissance). Figure 6.1 shows an example of a potential pitfall of solving a simultaneous move game as a turn-taking game. Even though literature [31][66][18] suggests tree search algorithms such as MCTS can be applied in simultaneous move games, they can eventually limit the deployment of such systems for high-stake applications. Thus, analyzing the connection between the turn-based game and simultaneously moving game could also improve the system's robustness.

Also, note that we model the robot's motion as a discrete system when applying the tree search techniques, but the action of the robot and the adversary are both continuous

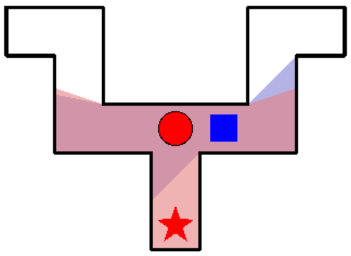


Figure 6.1: An example of one potential pitfall of solving a simultaneous move game as a turn-base game. Given the current position, in a turn-taking game, if the agent (red dot) assumes it moves first, then it may reason that if it moves right, then the opponent (blue square) may choose to stay in place since it has no other incentive to move. Thus, the agent may believe that it may be able to prevent the asset (red star) from being revealed. In reality, if the agent and the opponent move simultaneously, it is possible that the opponent may start moving left as the agent moves right making it difficult to predict whether the agent can protect the asset or not.

in the real world. Likewise, the measurements (as seen in Chapter 2) are also continuous. Discretization of the state/action spaces always us to solve these problems using tree search techniques. There are existing algorithms to solve versions of these problems in continuous spaces (although not the exact version we consider). Kokolakis et al. [56] develop a coordinated target tracking framework in the continuous space through a non-equilibrium game-theoretic approach. Zengin et al. [112] present a gradient search algorithm for real-time target tracking in continuous adversarial environments. Ideas from these papers could be leveraged here. Furthermore, we could also employ sampling-based algorithms to solve a continuous problem with a tree search algorithm. State of the art solvers [59] for Partially Observable Markov Decision Processes in continuous state/action/measurement spaces employ sampling-based routines and then use Monte Carlo Tree Search algorithms.

Another direction that is promising is the continuous refinement of the discrete trajectories found by the tree search algorithms. In fact, our results reported in Chapter 2 show that trajectory optimization results in improved performance. Instead of naively choosing initial trajectories that are needed by the continuous optimization algorithms (such as CHOMP [84] and trajopt [88]), we can use the solutions given by the discretized discrete tree search algorithms as the starting point.

The robustness of the algorithm needs to be considered more thoroughly. For example, suppose the adversary does not follow the robot's model. In that case, DM1 may perform even worse than minimax in asymmetric games. It is possible that the agent may try to exploit an asymmetry that does not exist, thereby, performing even worse than what a conservative policy would do. Currently, it is also possible to come up with imperceivable adversarial noise to fool the prediction system of the robot, as we have shown in Chapter 3. In addition, the real behavior of an adversary is usually hard to predict. More thorough analysis of the robustness of the algorithms proposed in this dissertation is required.

Learning techniques have shown progress in solving adversarial problems as well as improving the robustness recently. Our recent work [28] also shows reinforcement learning (RL) is promising in a relevant persistent monitoring problem, and our experimental results show that given sufficient training time, the RL approach may be better than non-RL baselines. RL could handle the scenario where we need to look ahead for a longer planning horizon. Also, the efficiency and robustness of the tree search techniques can be improved with learning. For example, AlphaGo [89] used MCTS with deep learning to solve the game of Go by combining the structure of MCTS with the learned knowledge from past human played games and self-play

Bibliography

[1] How to accelerate artificial intelligence in the defense department. `https://acquisitiontalk.com/2020/07/how-to-accelerate-artificial-intelligence-in-the-defense-department/`. Accessed: June 30, 2021.

[2] Robotics: Science and systems workshop in adversarial robotics. `http://hcr.mines.edu/2018-rss-workshop/`. Accessed: June 30, 2018.

[3] Mohammad Al Faruque, Francesco Regazzoni, and Miroslav Pajic. Design methodologies for securing cyber-physical systems. In *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*, pages 30–36. IEEE Press, 2015.

[4] James Arnold, SW Shaw, and HENRI Pasternack. Efficient target tracking using dynamic programming. *IEEE transactions on Aerospace and Electronic Systems*, 29(1):44–56, 1993.

[5] Nikolay Atanasov, Jerome Le Ny, Kostas Daniilidis, and George J Pappas. Information acquisition with sensing robots: Algorithms and error bounds. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6447–6454. IEEE, 2014.

[6] Cheng-Zong Bai, Vijay Gupta, and Fabio Pasqualetti. On kalman filtering with compromised sensors: Attack stealthiness and performance bounds. *IEEE Transactions on Automatic Control*, 62(12):6641–6648, 2017.

[7] Cheng-Zong Bai, Fabio Pasqualetti, and Vijay Gupta. Data-injection attacks in stochastic control systems: Detectability and performance tradeoffs. *Automatica*, 82:251–260, 2017.

[8] Hendrik Baier and Mark HM Winands. Monte-carlo tree search and minimax hybrids. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8, 2013.

[9] Tirthankar Bandyopadhyay, Yuanping Li, MH Ang, and David Hsu. A greedy strategy for tracking a locally predictable target among obstacles. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2342–2347. IEEE, 2006.

[10] Yaakov Bar-Shalom, X Rong Li, and Thiagalingam Kirubarajan. *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2004.

[11] Tamer Başar and Geert Jan Olsder. *Dynamic noncooperative game theory*. SIAM, 1998.

[12] Richard E Bellman and Stuart E Dreyfus. *Applied dynamic programming*. Princeton university press, 2015.

[13] Gines Benet, Francisco Blanes, José E Simó, and Pascual Pérez. Using infrared sensors for distance measurement in mobile robots. *Robotics and autonomous systems*, 40(4):255–266, 2002.

[14] Nicola Bezzo, James Weimer, Miroslav Pajic, Oleg Sokolsky, George J Pappas, and Insup Lee. Attack resilient state estimation for autonomous robotic systems. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 3692–3698. IEEE, 2014.

[15] Deepak Bhadauria and Volkan Isler. Capturing an evader in a polygonal environment with obstacles. In *IJCAI*, pages 2054–2059, 2011.

[16] Anil Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distribution. *Bull. Calcutta Math. Soc*, 1943.

[17] Shaunak D Bopardikar, Francesco Bullo, and Joao P Hespanha. Sensing limitations in the lion and man problem. In *American Control Conference, 2007. ACC'07*, pages 5958–5963. IEEE, 2007.

[18] Branislav Bošanskỳ, Viliam Lisỳ, Marc Lanctot, Jiří Čermák, and Mark HM Winands. Algorithms for computing strategies in two-player simultaneous move games. *Artificial Intelligence*, 237:1–40, 2016.

[19] M Boutayeb, H Rafaralahy, and M Darouach. Convergence analysis of the extended kalman filter used as an observer for nonlinear deterministic discrete-time systems. *IEEE transactions on automatic control*, 42(4):581–586, 1997.

[20] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[21] B Brumback and M Srinath. A chi-square test for fault-detection in kalman filters. *IEEE Transactions on Automatic Control*, 32(6):552–554, 1987.

[22] Lucian Buşoniu, Jihene Ben Rejeb, Ioana Lal, Irinel-Constantin Morărescu, and Jamal Daafouz. Optimistic minimax search for noncooperative switched control with or without dwell time. *Automatica*, 112:108632, 2020.

[23] Svante Carlsson and Bengt J Nilsson. Computing vision points in polygons. *Algorithmica*, 24(1):50–75, 1999.

[24] David Carmel and Shaul Markovitch. Learning models of opponent's strategy game playing. 1993.

[25] David Carmel and Shaul Markovitch. Incorporating opponent models into adversary search. In *AAAI/IAAI, Vol. 1*, pages 120–125, 1996.

[26] Hung-Jui Chang, Chih-Wen Hsueh, and Tsan-sheng Hsu. Convergence and correctness analysis of monte-carlo tree search algorithms: A case study of 2 by 4 chinese dark chess. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 260–266. IEEE, 2015.

[27] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008.

[28] Jingxi Chen, Amrish Baskaran, Zhongshun Zhang, and Pratap Tokekar. Multi-agent reinforcement learning for persistent monitoring. *arXiv preprint arXiv:2011.01129*, 2020.

[29] Yingying Chen, Wade Trappe, and Richard P Martin. Detecting and localizing wireless spoofing attacks. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON'07. 4th Annual IEEE Communications Society Conference on*, pages 193–202. IEEE, 2007.

[30] Timothy H Chung, Geoffrey A Hollinger, and Volkan Isler. Search and pursuit-evasion in mobile robotics. *Autonomous robots*, 31(4):299, 2011.

[31] Niek GP Den Teuling and Mark HM Winands. Monte-carlo tree search for the simultaneous move game tron. *Univ. Maastricht, Netherlands, Tech. Rep*, 2011.

[32] Wenjie Dong. Tracking control of multiple-wheeled mobile robots with limited information of a desired trajectory. *IEEE transactions on robotics*, 28(1):262–268, 2012.

[33] HHLM Donkers, JWHM Uiterwijk, and HJ van den Herik. Investigating probabilistic opponent-model search. In *BNAIC'00: Proceedings of the Twelfth Belgium-Netherlands Artificial Intelligence Conference, De Efteling, Kaatsheuvel, November 1-2, 2000*, pages 337–338. Tilburg University, 2000.

[34] Xiaoyuan Fan, Liang Du, and Dongliang Duan. Synchrophasor data correction under gps spoofing attack: A state estimation based approach. *IEEE Transactions on Smart Grid*, 2017.

[35] Hamza Fawzi, Paulo Tabuada, and Suhas Diggavi. Secure state-estimation for dynamical systems under active adversaries. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 337–344. IEEE, 2011.

[36] Xinbo Gao, Hiroyuki Iida, Jos WHM Uiterwijk, and H Jaap van den Herik. A speculative strategy. In *International Conference on Computers and Games*, pages 74–92. Springer, 1998.

[37] Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[38] Sylvain Gelly and Yizao 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.

[39] Stephanie Gil, Swarun Kumar, Mark Mazumder, Dina Katabi, and Daniela Rus. Guaranteeing spoof-resilient multi-robot networks. *Autonomous Robots*, 41(6):1383–1400, 2017.

[40] Piotr J Gmytrasiewicz, Edmund H Durfee, and David K Wehe. A decision-theoretic approach to coordinating multi-agent interactions. In *IJCAI*, volume 91, pages 63–68, 1991.

[41] Dongbing Gu. A game theory approach to target tracking in sensor networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(1):2–13, 2011.

[42] Geoffrey Hollinger, Sanjiv Singh, Joseph Djugash, and Athanasios Kehagias. Efficient multi-robot search for a moving target. *The International Journal of Robotics Research*, 28(2):201–219, 2009.

[43] Stefan Hrabar. 3d path planning and stereo-based obstacle avoidance for rotorcraft uavs. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 807–814. IEEE, 2008.

[44] Baoqi Huang, Lihua Xie, and Zai Yang. Tdoa-based source localization with distance-dependent noises. *IEEE Transactions on Wireless Communications*, 14(1):468–480, 2015.

[45] Shiyuan Jin and Zhihua Qu. A heuristic task scheduling for multi-pursuer multi-evader games. In *Information and Automation (ICIA), 2011 IEEE International Conference on*, pages 528–533. IEEE, 2011.

[46] Harold Lee Jones. *Failure detection in linear systems.* PhD thesis, Massachusetts Institute of Technology, 1973.

[47] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for a class of pursuit-evasion games. In *Algorithmic foundations of robotics IX*, pages 71–87. Springer, 2010.

[48] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[49] Nikhil Karnad and Volkan Isler. Modeling human motion patterns for multi-robot planning. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3161–3166. IEEE, 2012.

[50] Bilal Kartal, Pablo Hernandez-Leal, and Matthew E Taylor. Action guidance with mcts for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, pages 153–159, 2019.

[51] Lydia E Kavraki, Mihail N Kolountzakis, and J-C Latombe. Analysis of probabilistic roadmaps for path planning. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 4, pages 3020–3025. IEEE, 1996.

[52] Ayoung Kim and Ryan M Eustice. Active visual slam for robotic area coverage: Theory and experiment. *The International Journal of Robotics Research*, 34(4-5):457–475, 2015.

[53] Donald E Knuth and Ronald W Moore. An analysis of alpha-beta pruning. *Artificial intelligence*, 6(4):293–326, 1975.

[54] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[55] Nathan Koenig and Andrew 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*, volume 3, pages 2149–2154. IEEE, 2004.

[56] Nick-Marios T Kokolakis, Aris Kanellopoulos, and Kyriakos G Vamvoudakis. Bounded rational unmanned aerial vehicle coordination for adversarial target tracking. In *2020 American Control Conference (ACC)*, pages 2508–2513. IEEE, 2020.

[57] Andreas Kolling and Stefano Carpin. Pursuit-evasion on trees by robot teams. *IEEE Transactions on Robotics*, 26(1):32–47, 2010.

[58] Panqanamala Ramana Kumar and Pravin Varaiya. *Stochastic systems: Estimation, identification, and adaptive control*, volume 986. Prentice Hall Englewood Cliffs, NJ, 1986.

[59] Hanna Kurniawati. Partially observable markov decision processes (pomdps) and robotics. *arXiv preprint arXiv:2107.07599*, 2021.

[60] Jonathan A Larcom and Hong Liu. Modeling and characterization of gps spoofing. In *Technologies for Homeland Security (HST), 2013 IEEE International Conference on*, pages 729–734. IEEE, 2013.

[61] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[62] Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

[63] Frank L Lewis and Vassilis L Syrmos. *Optimal control*. John Wiley & Sons, 1995.

[64] X Rong Li and Vesselin P Jilkov. Survey of maneuvering target tracking. part i. dynamic models. *Aerospace and Electronic Systems, IEEE Transactions on*, 39(4):1333–1364, 2003.

[65] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009.

[66] Viliam Lisy, Vojta Kovarik, Marc Lanctot, and Branislav Bosansky. Convergence of monte carlo tree search in simultaneous move games. In *Advances in Neural Information Processing Systems*, pages 2112–2120, 2013.

[67] Yao Liu, Peng Ning, and Michael K Reiter. False data injection attacks against state estimation in electric power grids. *ACM Transactions on Information and System Security (TISSEC)*, 14(1):13, 2011.

[68] V. Macias, I. Becerra, R. Murrieta-Cid, H.M. Becerra, and S. Hutchinson. Image feedback based optimal control and the value of information in a differential game. *Automatica*, 90:271–285, April 2018.

[69] Yilin Mo, Emanuele Garone, Alessandro Casavola, and Bruno Sinopoli. False data injection attacks against state estimation in wireless sensor networks. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 5967–5972. IEEE, 2010.

[70] Yilin Mo and Bruno Sinopoli. False data injection attacks in control systems. In *First Workshop on Secure Control Systems, CPS Week, Stockholm, Sweden*, 2010.

[71] Yilin Mo and Bruno Sinopoli. On the performance degradation of cyber-physical systems under stealthy integrity attacks. *IEEE Transactions on Automatic Control*, 61(9):2618–2624, 2015.

[72] Michael Montemerlo, Joelle Pineau, Nicholas Roy, Sebastian Thrun, and Vandi Verma. Experiences with a mobile robotic guide for the elderly. In *AAAI/IAAI*, pages 587–592, 2002.

[73] K. J. Obermeyer and Contributors. The visilibity library. `https://karlobermeyer.github.io/VisiLibity1/`.

[74] Joseph O'rourke. *Art gallery theorems and algorithms*. Oxford University Press Oxford, 1987.

[75] Darren Pais and Naomi E Leonard. Pursuit and evasion: evolutionary dynamics and collective motion. In *AIAA Guidance, Navigation and Control Conference*, pages 1–14, 2010.

[76] Simon Parkinson, Paul Ward, Kyle Wilson, and Jonathan Miller. Cyber threats facing autonomous and connected vehicles: Future challenges. *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[77] Cheng Peng and Volkan Isler. View selection with geometric uncertainty modeling. *arXiv preprint arXiv:1704.00085*, 2017.

[78] Zhirong Qiu, Shuai Liu, and Lihua Xie. Distributed constrained consensus with distance-dependent measurement noises. *IFAC-PapersOnLine*, 49(22):234–239, 2016.

[79] Alberto Quattrini Li, Raffaele Fioratto, Francesco Amigoni, and Volkan 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*, pages 1693–1701, 2018.

[80] Steven AP Quintero, Francesco Papi, Daniel J Klein, Luigi Chisci, and Joao P Hespanha. Optimal uav coordination for target tracking using dynamic programming. In *49th IEEE Conference on Decision and Control (CDC)*, pages 4541–4546. IEEE, 2010.

[81] Eric Raboin, Ugur Kuter, and Dana 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*, pages 1201–1202, 2012.

[82] Eric Raboin, Dana S Nau, Ugur Kuter, Satyandra K Gupta, and Petr Svec. Strategy generation in multi-agent imperfect-information pursuit games. In *AAMAS*, pages 947–954, 2010.

[83] BSY Rao, Hugh F Durrant-Whyte, and JA Sheen. A fully decentralized multi-sensor system for tracking and surveillance. *The International Journal of Robotics Research*, 12(1):20–44, 1993.

[84] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pages 489–494. IEEE, 2009.

[85] Günter Rote. Pursuit-evasion with imprecise target location. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 747–753. Society for Industrial and Applied Mathematics, 2003.

[86] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 2009.

[87] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[88] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[89] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[90] Nicholas M Stiffler and Jason M O'Kane. Complete and optimal visibility-based pursuit-evasion. *The International Journal of Robotics Research*, 36(8):923–946, 2017.

[91] Jie Su, Jianping He, Peng Cheng, and Jiming Chen. A stealthy gps spoofing strategy for manipulating the trajectory of an unmanned aerial vehicle. *IFAC-PapersOnLine*, 49(22):291–296, 2016.

[92] Wei Sun, Yunpeng Pan, Jaein Lim, Evangelos A Theodorou, and Panagiotis Tsiotras. Min-max differential dynamic programming: Continuous and discrete time formulations. *Journal of Guidance, Control, and Dynamics*, 41(12):2568–2580, 2018.

[93] Zachary N Sunberg and Mykel J Kochenderfer. Online algorithms for pomdps with continuous state, action, and observation spaces. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 2018.

[94] Yoonchang Sung, Ashish Kumar Budhiraja, Ryan Williams, and Pratap Tokekar. Distributed simultaneous action and target assignment for multi-robot multi-target tracking. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA)*, 2018.

[95] Pavel Surynek. A novel approach to path planning for multiple robots in bi-connected graphs. In *2009 IEEE International Conference on Robotics and Automation*, pages 3613–3619. IEEE, 2009.

[96] Kei Takada, Hiroyuki Iizuka, and Masahito Yamamoto. Reinforcement learning to create value and policy functions using minimax tree search in hex. *IEEE Transactions on Games*, 12(1):63–73, 2019.

[97] Vrizlynn LL Thing and Jiaxi Wu. Autonomous vehicle security: A taxonomy of attacks and defences. In *Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2016 IEEE International Conference on*, pages 164–170. IEEE, 2016.

[98] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[99] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. On the requirements for successful gps spoofing attacks. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 75–86. ACM, 2011.

[100] Pratap Tokekar, Elliot Branson, Joshua Vander Hook, and Volkan Isler. Tracking aquatic invaders: Autonomous robots for invasive fish. *IEEE Robotics and Automation Magazine*, 2013.

[101] Pratap Tokekar, Volkan Isler, and Antonio Franchi. Multi-target visual tracking with aerial robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[102] Pratap Tokekar and Vijay Kumar. Visibility-based persistent monitoring with robot teams. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 3387–3394. IEEE, 2015.

[103] Pratap Tokekar, Joshua Vander Hook, and Volkan Isler. Active target localization for bearing based robotic telemetry. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[104] Jur Van Den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, 2012.

[105] Joshua Vander Hook and Volkan Isler. Pursuit and evasion with uncertain bearing measurements. In *CCCG*, 2014.

[106] Rene Vidal, Omid Shakernia, H Jin Kim, David Hyunchul Shim, and Shankar Sastry. Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *IEEE transactions on robotics and automation*, 18(5):662–669, 2002.

[107] Michael P. Vitus, Wei Zhang, Alessandro Abate, Jianghai Hu, and Claire J. Tomlin. On efficient sensor scheduling for linear dynamical systems. *Automatica*, 48(10):2482–2493, October 2012.

[108] Thomas Walsh, Sergiu Goschin, and Michael Littman. Integrating sample-based planning and model-based reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 2010.

[109] W Willman. Formal solutions for a class of stochastic pursuit-evasion games. *IEEE Transactions on Automatic Control*, 14(5):504–509, 1969.

[110] Shuang Wu, Xiaoqiang Ren, Yiguang Hong, and Ling Shi. Max-min fair sensor scheduling: Game-theoretic perspective and algorithmic solution. *arXiv preprint arXiv:1902.03594*, 2019.

[111] I Yaesh and U Shaked. Game theory approach to state estimation of linear discrete-time processes and its relation to h-optimal estimation. *International Journal of Control*, 55(6):1443–1452, 1992.

[112] Ugur Zengin and Atilla Dogan. Real-time target tracking for autonomous uavs in adversarial environments: A gradient search algorithm. *IEEE Transactions on Robotics*, 23(2):294–307, 2007.

[113] Jiangfan Zhang, Rick S Blum, Lance M Kaplan, and Xuanxuan Lu. Functional forms of optimum spoofing attacks for vector parameter estimation in quantized sensor networks. *IEEE Transactions on Signal Processing*, 65(3):705–720, 2017.

[114] Zhongshun Zhang, Joseph Lee, Jonathon M Smereka, Yoonchang Sung, Lifeng Zhou, and Pratap Tokekar. Tree search techniques for minimizing detectability and maximizing visibility. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8791–8797. IEEE, 2019.

[115] Zhongshun Zhang, Jonathon M Smereka, Joseph Lee, Lifeng Zhou, Yoonchang Sung, and Pratap Tokekar. Game tree search for minimizing detectability and maximizing visibility. *Autonomous Robots*, 45(2):283–297, 2021.

[116] Zhongshun Zhang and Pratap Tokekar. Non-myopic target tracking strategies for non-linear systems. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 5591–5596. IEEE, 2016.

[117] Zhongshun Zhang and Pratap Tokekar. Tree search techniques for adversarial target tracking with distance-dependent measurement noise. *IEEE Transactions on Control Systems Technology*, 2021.

[118] Zhongshun Zhang, Lifeng Zhou, and Pratap Tokekar. Strategies to design signals to spoof kalman filter. In *2018 Annual American Control Conference (ACC)*, pages 5837–5842. IEEE, 2018.

[119] Lifeng Zhou, Vasileios Tzoumas, George J Pappas, and Pratap Tokekar. Resilient active target tracking with multiple robots. *IEEE Robotics and Automation Letters*, 4(1):129–136, 2018.