DISSERTATION

PATH PLANNING FOR AUTONOMOUS AERIAL VEHICLES USING MONTE CARLO TREE SEARCH

Submitted by

Apichart Vasutapituks

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2024

Doctoral Committee:

Advisor: Edwin K. P. Chong

Mahmood Azimi-Sadjadi
Olivier Pinaud
Ali Pezeshki

ABSTRACT

PATH PLANNING FOR AUTONOMOUS AERIAL VEHICLES
USING MONTE CARLO TREE SEARCH

Unmanned aerial vehicles (UAVs), or drones, are widely used in civilian and defense applications, such as search and rescue operations, monitoring and surveillance, and aerial photography. This dissertation focuses on autonomous UAVs for tracking mobile ground targets. Our approach builds on optimization-based artificial intelligence for path planning by calculating approximately optimal trajectories. This approach poses a number of challenges, including the need to search over large solution spaces in real-time. To address these challenges, we adopt a technique involving a rapidly-exploring random tree (RRT) and Monte Carlo tree search (MCTS). The RRT technique increases in computational cost as we increase the number of mobile targets and the complexity of the dynamics. Our MCTS approach executes a tree search based on random sampling to generate trajectories in real time. We develop a variant of MCTS for online path-planning to track ground targets together with an associated algorithm called P-UAV. Our algorithm is based on the framework of partially observable Monte Carlo planning, originally developed in the context of MCTS for Markov decision processes. Our real-time approach exploits a parallel-computing strategy with a heuristic random-sampling process. In our framework, We explicitly incorporate threat evasion, obstacle collision avoidance, and resilience to wind. The approach embodies an exploration-exploitation tradeoff in seeking a near-optimal solution in spite of the huge search space. We provide simulation results to demonstrate the effectiveness of our path-planning method.

ACKNOWLEDGEMENTS

I would like to give a huge thank you to everyone who supports this research. Firstly, I would like to express my sincere thanks and deepest gratitude to my advisors, Prof. Edwin K. P. Chong for educating me as a Ph. D. student, for practicing me with the most valuable attitude in working on research, writing articles, and presenting my work, for supporting and assisting me in course works, scholarship and registration work, international student affairs, and for all suggestions in university life with spiritual care during my graduate studies at Colorado State University. It is a great pleasure and honor to study the Ph. D. program under the supervision of Prof. Edwin K. P. Chong. I would also like to convey my sincere gratitude and thanks to my Ph. D. committees, Prof. Mahmood Azimi-Sadjadi, Prof. Olivier Pinaud, and Prof. Ali Pezeshki for their instructions and dedication in my Ph.D. committees. All professors are not only my preceptors in school, but they are my valuable persons giving me precious experiences in life. I'm so proud of being a student in your classes and department.

I would like to thank Dr. Shankarachary Ragi, Dr. Yajing Liu, Dr. Yugandhar Sarkale, Somayeh Hosseini, and Dr. Mahsa Ghorbani for their exchange of courseworks knowledge and research work. I would also like to thank my friends and classmates who share their research work and university life experience, Dr. Fateh El Sherif, Dr.Tushar Ganguli, Dr. Pranav Damale , Dr.Savini Samarasinghe, Laci Rauch, Sarah Delaet and Brandon Cook, Ryan Loaiza, Sterling Krone, Vanessa Enriquez Krone, Jacob Oburke, Orozco Addie, Zenon A Kampman, Ryan Jensen, Jeff Larchar, Madeline Jekot ,Joseph Yu Zhang, Marta Camacho, Jason Moon, Billy Phillips, Emily Dalton, Oren Pierce, and Anthony Olguin. I would like to thanks specially to high school teacher and friends, Prof. Somrit Srisuwan for giving me good suggestions and education, Mr. Chaiyaploeg Meethissom with his family (father, mother, auntie) for giving me a sincere friendship and a great help from his family, Prof. Tatpong Katanyukul for advising me a study abroad at Colorado State University.

Especially, I would like to thank National Science and Technology Development Agency (NSTD) of Thailand and Prince of Songkla University (PSU) for supporting me scholarship and education to create my an opportunity of Ph.D. study. I would like to thanks you to Thai professors who support me to study in the Graduate school includes Prof. Chusak Limsakul, Prof. Krerkchai Thongnoo, Prof. Pornchai Phukpattaranont, Prof. Nattha Jindapetch, Prof. Bunjong Piyatamrong, Prof. Chom Kimpan, Prof. Pichaya Tandayya, and Prof. Thammaratt Samitalampa. Extraordinarily, I would like to thank you to Prof. Kovit Surussavadee, Dr. Chinnawat Surussavadee, Prof. Virach Wootipoom for valuable help and great support, also all professors who taught electrical engineering to me at King Mongkut's Institute of Technology Ladkrabang and all teachers who taught me at Bodindecha (Sing Singhaseni) School and Samsennok School (Pracharat Anukul).

I acknowledge my parents who mean so much to me, Mr.Kimsoon and Mrs. Panpimol Vasutapituks, for supporting me to go to school at all levels, and my three brothers, Mr.Apichet, Mr.Apichit, and Mr.Aroonchai Vasutapituks for their warm family, which certainly pushes me toward the success of a Ph.D. study.

I'm so thankful for all your support and encouragement. I strive to continue developing our research to meet practical applications in modern technology.

DEDICATION

I would like to dedicate this dissertation to my family.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

Modern technology in embedded systems plays a key role in numerous electronic devices and computers, leading to the fast development of equipment and appliances. Unmanned aerial vehicles (UAVs), or drones, utilize embedded systems technology to implement their computing components and parts. Diversified reasons for the use of embedded electronic systems are cheaper price, smaller size , and higher processing performance. A novel embedded system module leading to efficient design and implementation for UAV systems is produced rapidly with numerous relevant capabilities. Accordingly, UAV applications with new embedded technology are applicable and extensively used because of their high mobility ability, low maintenance and economic cost, and ease of control. Information processing in UAV uses embedded onboard systems, and can be exploited for autonomous positioning and tracking. These systems incorporate the global positioning system (GPS), communication capabilities, autonomous control, image processing, and artificial intelligence (AI). Nonetheless, interest persists in more intelligent UAVs to support future applications, simultaneously reducing human effort and achieving higher performance goals.

Emerging trends in UAV technology have been playing an increasingly essential role in our daily lives in various fields. It has brought plenty of benefits from UAV applications capabilities in the world today and is not piloted by humans on board. UAVs employ guidance algorithms to generate their trajectory planning, which is a major component of the UAV control system. Guidance algorithms for aerial vehicles in complex environments require a real-time algorithm for the computation of optimal trajectories. Path planning is operated mainly for a main function in guidance algorithms ,generating vehicle trajectories. A major challenge in intelligent UAV development is path planning in dynamic environments with constraints, differing significantly in some aspects from traditional robot path planning. For example, a typical UAV has a relatively large turning radius and must maintain a minimum speed.

In our work, we are interested in the problem of UAV path-planning improvement by applying real-time guidance algorithm [4, 5] ,which uses partially observable Monte Carlo planning (POMCP) based on Monte Carlo tree search (MCTS). We develop an online path-planning algorithm for tracking ground targets, called the P-UAV algorithm. More precisely, our real-time algorithm utilizes parallel processing with a heuristic random-sampling technique. The algorithm uses conceptually an exploration-exploitation tradeoff to find a near-optimal solution in a large search space. In a complex environment, the experiment results show our algorithm's performance in threat evasion, obstacle collision avoidance, and resilience to wind.

1.1 Background and Motivation

1.1.1 Introduction to Path Planning and P-UAV Framework

Path planning is a crucial process based on the concept of guidance, navigation, and control (GNC) for autonomous vehicle systems. The principal goal of path planning for UAVs is to find an optimal vehicle path from a starting location to a goal location with various obstacles, collision avoidance, and a constrained condition in real-world environments. UAV path planning is classified as a constrained optimization problem, or NP-hard, with a high computational complexity. It takes a large amount of computation time to search for an optimal path. With support in computing paths, UAVs exploiting sensor measurements by onboard computing capabilities construct an optimal trajectory that is computationally demanding. The values of forward acceleration and bank angle are used for piloting a traditional UAV in the air, subject to multiple constraints in a dynamic environment.

We formulate and solve the path-planning problem here by employing partially observable Markov decision processes (POMDPs). A POMDP is generally used to model decision problems under uncertainty. Applications of POMDPs appear in several fields, such as machine vision, search and tracking, robot navigation, autonomous control, human-robot interaction, and machine maintenance. A POMDP is a generalization of a Markov decision process (MDP), employing a MDP to model system dynamics and a hidden Markov model for the observations of unobserv-

able states. It is generally intractable to compute an optimal solution in a POMDP problem [6]. Normally, approximations are needed in obtaining POMDP solutions, such as approximate dynamic programming, heuristics, hindsight optimization, partially observable Monte-Carlo planning (POMCP) [7], and nominal belief-state optimization (NBO) [8]. Specifically, the NBO technique is designed for UAV path planning ([8–10]) using a POMDP framework. In other related work, [11] formulates an object search problem as a POMDP and applies a DESPOT system [12] for planning.

A POMDP can be converted to a conventional MDP in which the state space is the posterior distribution (called the *belief state*) of the underlying unobservable state computed from the observation history. The observation law (conditional distribution) depends on the present state and control action. The set of belief states is continuous and extremely large, even if discretized. Monte Carlo tree search (MCTS) is used in calculating approximate MDP solutions based on sampling [13]. POMCP [7] is a variant of MCTS that employs particle filters for belief-state representation in the search tree for large POMDPs [5]

Monte Carlo tree search (MCTS) was employed in AlphaGo [13] in 2006. It seeks near-optimal solutions to decision problems by combining a best-first strategy with an adaptive sampling method using the upper-confidence-bound (UCB). Particularly, MCTS applies the method of upper confidence bound for trees (UCT) to achieve a tradeoff between exploration and exploitation in order to seek near-optimal solutions in large state-spaces. The MCTS algorithm combines the generality of Monte Carlo sampling with the precision of tree search. MCTS is particularly suited to problems with expensive function evaluations. MCTS has been widely used in a variety of areas, such as planning, scheduling, image processing, and gaming. In the UAV area, the applications of MCTS range from data gathering to image recognition for online decision making.

POMCP [7] involves applying MCTS to POMDPs, as presented by Silver and Veness in 2010. The method accounts for partial observations by connecting actions with a history of observations to a node in the search tree. Each node of the search tree represents a state of the problem in terms of a belief state, containing a history with its own statistical value. The history consists of actions and observations $h_t = \{a_1, o_1, ..., a_t, o_t\}$, and the statistical value of the history is calculated using a

Monte-Carlo technique for states considered from the present belief state. POMCPs for POMDPs have been used for drones and robots. For example, [14] used an adapted POMCP technique with the traditional UCB algorithm for an autonomous urban UAV navigation problem. A target object search problem was solved using POMCP in [15].

Our present work develops an autonomous path-planning algorithm based on a method called *P-UAV*. We developed P-UAV in [4] for multitarget tracking problems. P-UAV was designed using a POMCP approach to sensibly navigate UAVs for tracking mobile ground targets in complex environments in real time. Here, we use a Kalman filter technique instead of a particle filter to update the belief-state in the search tree. Our current modified P-UAV algorithm employs a heuristic technique with the UAV acceleration and bank angle under constraints to produce an efficient non-myopic path planning method. In real-time applications, we integrate parallel computing into P-UAV to improve its computational performance.

1.1.2 Literature Review

The general approach of path-planning papers on UAVs appeared in the 1979s [16] ,with searching for path planning in UAV communication systems for varied shapes and complexities. The demands of UAVs have increased in recent years, and the research on the path planning technique has been extended from the original goal of reaching the target without any collisions to the advanced goal of autonomous aerial vehicles in real-time in response to dynamically changing conditions. Therefore, optimal path planning in real-time is essentially required for autonomous UAVs.

A variety of algorithms to solve the path-planning problem, categorized as a combinatorial optimization problem has been proposed over four decades. The most combinatorial optimization problems are considered NP-hard problems, such as planning, routing, scheduling, traveling salesman, and graph partitioning.

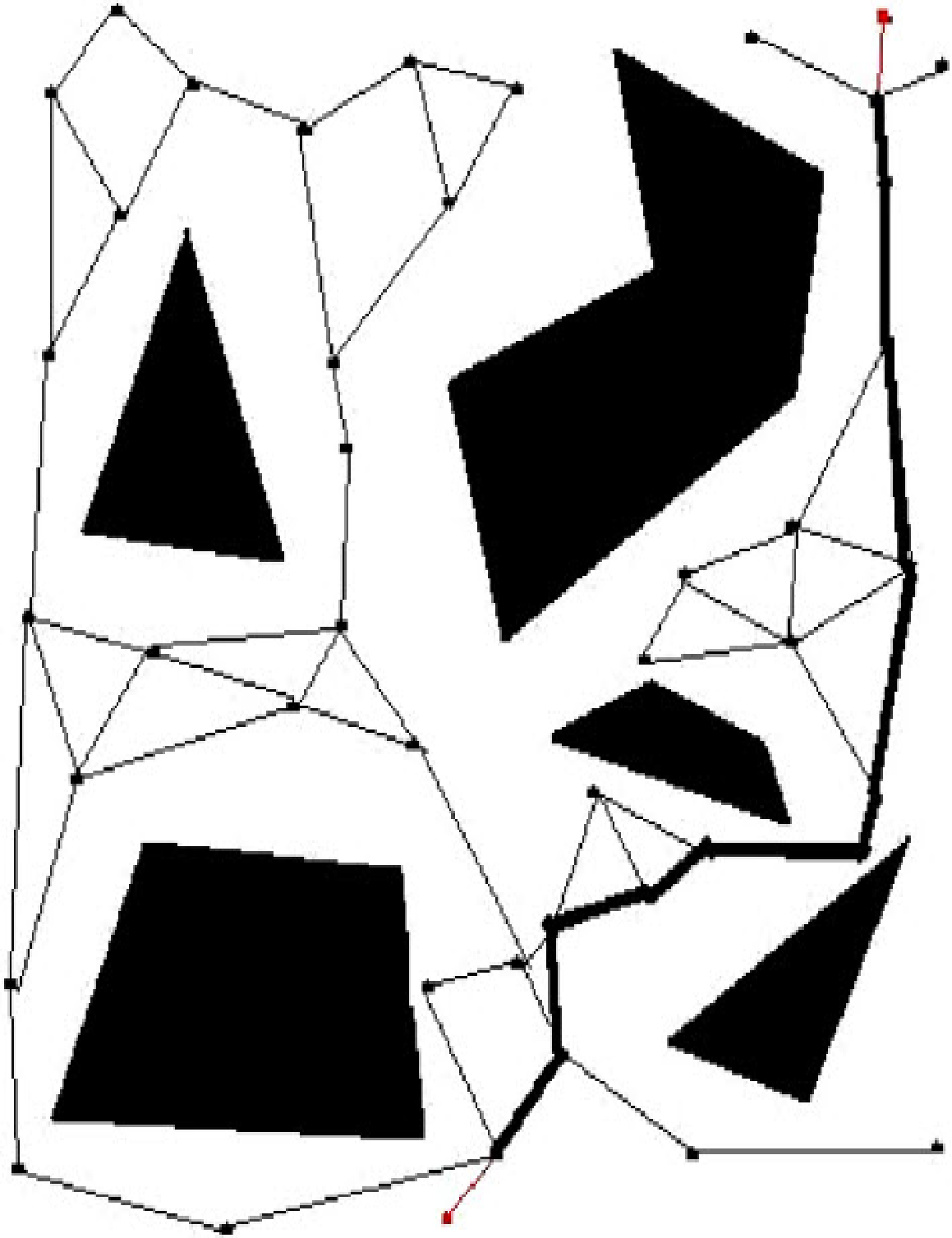

Figure 1.1: Probabilistic Roadmaps [1, 2]

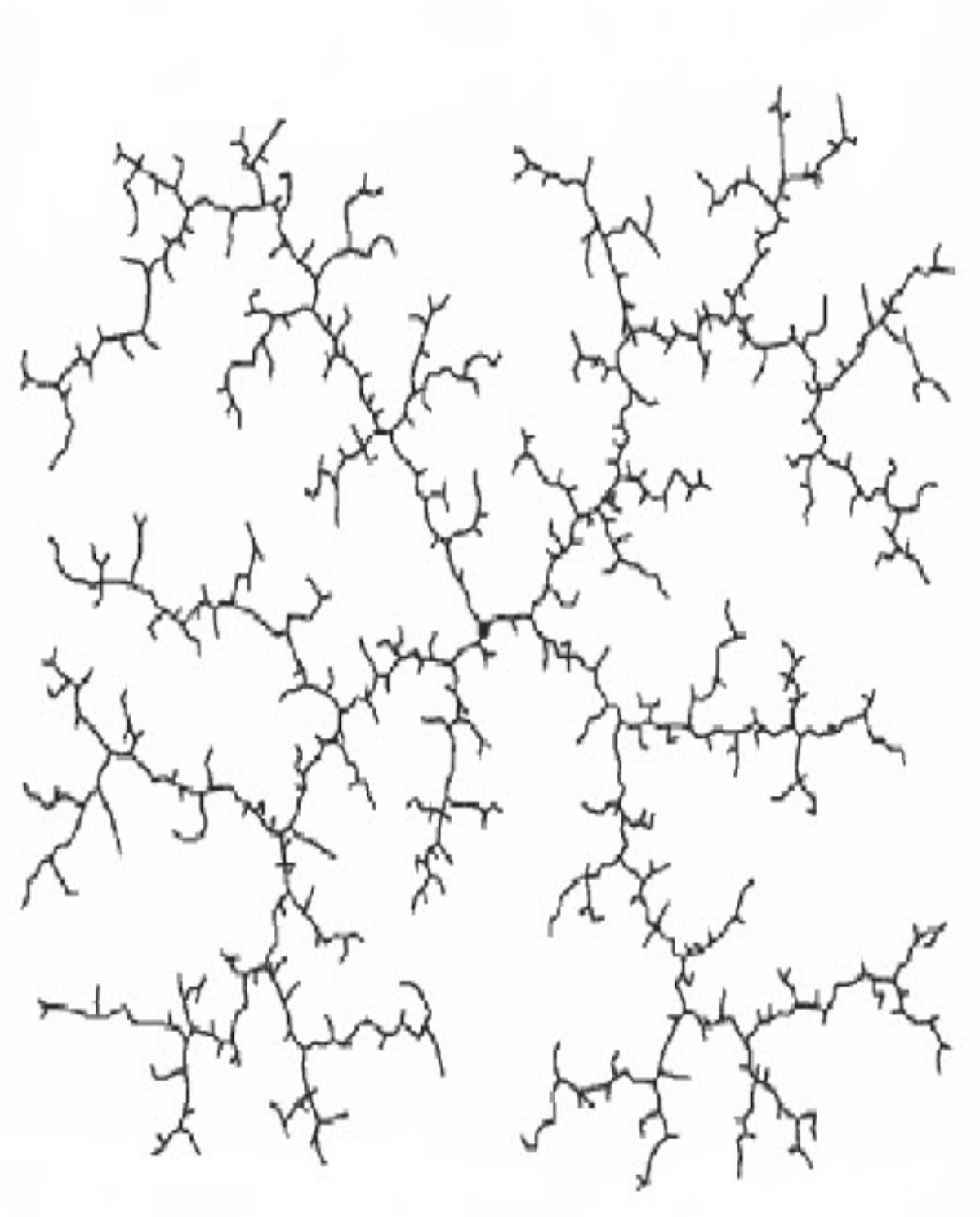

Figure 1.2: Rapid-exploring Random Trees Algorithm [1]

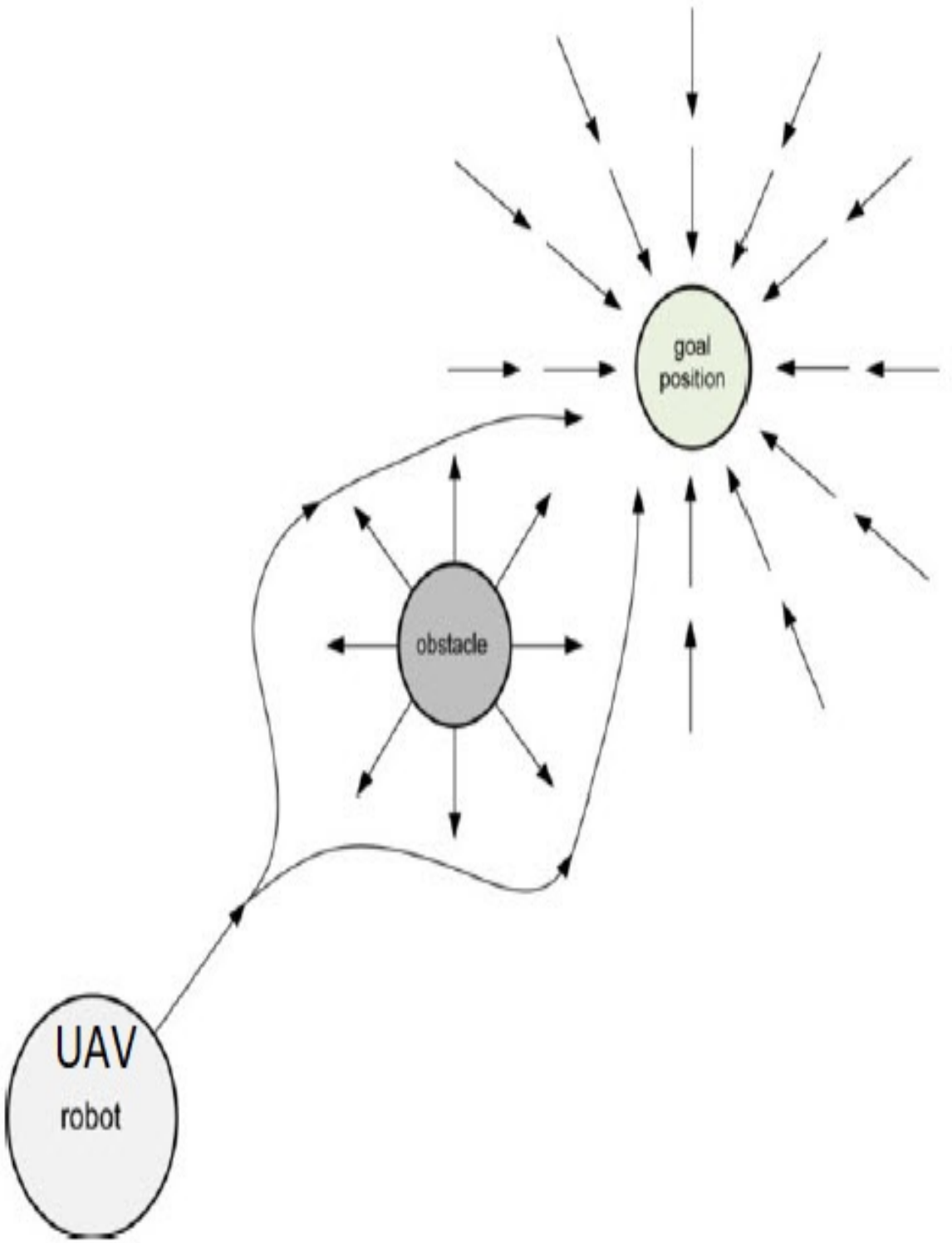

Figure 1.3: Potential field method [3]

The problem of path-planning being generally intractable, especially in high-dimensional search spaces, has led to the proposal of a sample-based technique by Kavraki, Svestka, Latombe, and Overmars in 1996, called probabilistic roadmaps (PRM) [17], and Kuffner and LaValle in 2000, called rapidly exploring random trees (RRT) [18] .

- **Probabilistic RoadMaps (PRM)**: generates randomly a path from a starting point to a goal point while avoiding obstacles, and then employs a graph search in finding the shortest path by exploiting A* Search, as shown in Figure 1.1.

- **Rapidly exploring random trees (RRT)**: builds two trees rooted in the starting and goal points with random path generation in both directions while avoiding obstacles, uses a search technique for both trees to find a path, as shown in Figure 1.2.

The PRM algorithm uses a search tree, with the random nodes representing the vehicle position in environments, and the straight edges representing a UAV path. In Figure 1.1, the shortest distance of the vehicle is indicated as the bold line [19]. A variant of PRM called kinodynamic PRM is combined non-linear predictive control for solving trajectory planning [20]. The PRM technique is appropriate for huge search spaces, but it can not optimize the path-planning problem, has a low speed of convergence rate [21–23]. This technique [24–26] can generate poor solutions with impractical operations such as obstacle closeness. Nevertheless, traditional PRM has been developed by making it suitable for practical operations.

The standard RRT algorithm by calculating fixed distances in one step can not be guaranteed to find optimal path planning but it's an efficient algorithm for high-dimensional spaces. RRT became the most famous path-planning method for a simple planner [27]. The modified RRT algorithm [28] is designed for path planning in complex environments with threats. RRT-Connect [18] is a variant of RRT with a greedy computation employed to avoid UAV collisions. RRT Connect [29] incorporates with an artificial potential field to optimize trajectory planning, and obtain optimal

solutions. Likewise, the RRT technique combines with receding horizon (RH) [30] to compute real-time path planning in threat evasion.

Another path-planning algorithm based on a sample-based technique, called the potential field method (PFM) [31], assigns an object in the environment as a particle and finds the UAVs path exploiting the resultant fields. However, the traditional PFM can not reach the target when facing a local minima problem, as shown in Figure 1.3.

The other algorithms used to compute path planning for UAVs are categorized as artificial intelligence (AI) techniques. These algorithms include meta-heuristic search, brute force search (or depth first search), local search, and artificial neural networks. There are considerable meta-heuristic search algorithms such as the genetic algorithm (GA), particle swarm optimization (PSO), ant colony optimization(ACO), artificial neural networks (ANN), and greedy algorithm. The genetic algorithm has three basic operators with selection, crossover, and mutation operator, to find problem solutions. Many research papers on UAV path planning use the genetic algorithm [32–35] to find solutions, but without an optimal path similar to the colony optimization algorithm exploited in research [36–39]. The particle swarm optimization algorithm has the ability to search for an optimal solution, utilized in the development work [40–43]. Artificial neural networks have a basic component of an input layer, processing layers, and an output layer, to process data for the optimal path solutions without real-time response [44, 45]. The greedy algorithm is employed to solve problems with high computation time and is a classical technique integrated with other algorithms to enhance algorithm performance in work [46–49].

1.2 Our Contributions

In Chapter 2, we introduce our POMDP framework for the path-planning problem and define the basic components of the problem based on the POMDP framework. We indicate that the control actions of forward acceleration and bank angle are two main variables used for algorithm computation in order to optimize the problem. Our POMDP framework applies a discrete-time model with a suitable rate for problem solving and computing.

- The path-planning problem is defined as UAVs, targets, and obstacles in two-dimensional space with x and y axes.

- A variety of sensors of the UAV system are addressed with functions and technical details for information processing.

- The POMDP components are mainly composed of states, actions, observation and observation law, state-transition law, cost function, and belief state.

- The principle and background of Monte Carlo tree search are introcuced conceptually with an exploration-exploitation tradeoff, and standard MCTS stages including selection, expansion, simulation, and backpropagation stage.

In Chapter 3, we introduce the POMDP methodology and its implementation to our P-UAV algorithm.

- The POMDP concept is explained with more details in equation, compared with the NBO technique.

- The tracking problem for ground targets obtain the belief state sequence of targets from the Kalman filter.

- The initial cost function can be calculated by the mean-squared error between the tracks with the belief state and the targets with their real locations.

- The P-UAV Algorithm to avoid obstacle is shown as the cost function for obstacle collision avoidance, with the initial cost function adding with the obstacle penalty.

An exploration-exploitation concept based on partially observable Monte Carlo planning (POMCP) to choose an action by using the upper confidence bounds (UCB1) updates the cost function for each node in a search tree. The parallel-computing strategy is computed with a heuristic random-sampling process. This heuristic of control for the UAV direction uses three groups of angles with positive, zero, and negative, and uses the same value of forward acceleration for all nodes in a search tree. We propose wind factor especially in complex environment with obstacle avoidance. Threat avoidance is caculated to protect for UAVs in complex environment with obstacle avoidance.

In Chapter 4, our research work proposes an online path-planning algorithm [4] based on Monte Carlo tree search (MCTS) and uses the POMDP framework called POMCP for navigating intelligently UAVs to track ground targets only in real-time implementation. We create our online path-planning system by bring each component of the POMDP framework from Chapter2-3. With an appropriate implementation, the method can be used in real time for realistic practical scenarios. The method uses a Kalman filter to update the belief state and incorporates a heuristic approach P-UAV combines a best-first selection scheme and an adaptive Monte Carlo sampling technique using the UCB approach. Parallel processing can enhance the performance of P-UAV by increasing the number of CPU cores. P-UAV potentially outperforms NBO, especially with sufficient number of CPU cores, in more complex dynamic environments for online processing.

In Chapter 5, this research work develops the P-UAV algorithm [5] from Chapter 4 to guide UAVs for tracking mobile ground targets in complex environments with obstacles avoidance. We build on the framework of partially observable Markov decision process (POMDP) from Chapter2-3. Our P-UAV algorithm is based on partially observable Monte Carlo planning (POMCP) using Monte Carlo tree search (MCTS), originally developed for Markov decision process (MDP) problems. Combining a heuristic method with parallel computing, the algorithm is applicable to real-time applications. Our experimental results demonstrate that our autonomous path-planning algorithm is able to achieve near-optimal performance in large complex environments by appropriately tuning the exploration-exploitation tradeoff.

In Chapter 6, the experiment results of P-UAV from Chapter 5 shows that this algorithm can track mobile ground targets in more complex environments with obstacles avoidance, wind factor, and threat avoidance. Also, the experiment shows that the performance of P-UAV with a heuristic method is more efficient than P-UAV without a heuristic method. Our experimental results show that P-UAV generally outperforms NBO.

In Chapter 7, conclude our P-UAV algorithm have the ability to track mobile ground targets in complex environments with obstacle and threat collision avoidance, and wind factor. The computational performance was enhanced using heuristics and parallel computing. The computational performance can be improved by increasing the number of CPU cores. Also, we discuss some future research.

Chapter 2

Problem Definition

The UAV path-planning problem for mobile ground targets tracking is defined by the numerous features of the system.

The targets move on the ground in two dimensions, and UAVs fly at a constant altitude over the ground with a simple UAV motion model. Each UAV is controlled by applying the control action of forward acceleration and bank angle using position coordinates in two dimensions. A camera is mounted on the UAV with a machine-vision system for visual measurements, assumed to have random errors without false alarms and missed detections. The measurement error covariance of a target depends on the relative position of UAVs and targets. Obsctacles are modeled as occlusions with a variety of shapes, such as rectangles and circles, that block the UAV paths in the operation area, as shown in Figure **??**. UAV velocities are constrained within a prespecified interval, with bounded lateral acceleration in the plane.

The goal of P-UAV is to minimize the mean-squared error between tracks and targets and generate control actions for UAVs to keep track of the ground targets. In centralized control, the P-UAV algorithm collects data from all the sensors and fuses them in the tracker for updating tracks. The algorithm then calculates UAV control action commands to supervise the incremental UAV trajectories. We assume a discrete-time model with a suitable rate.

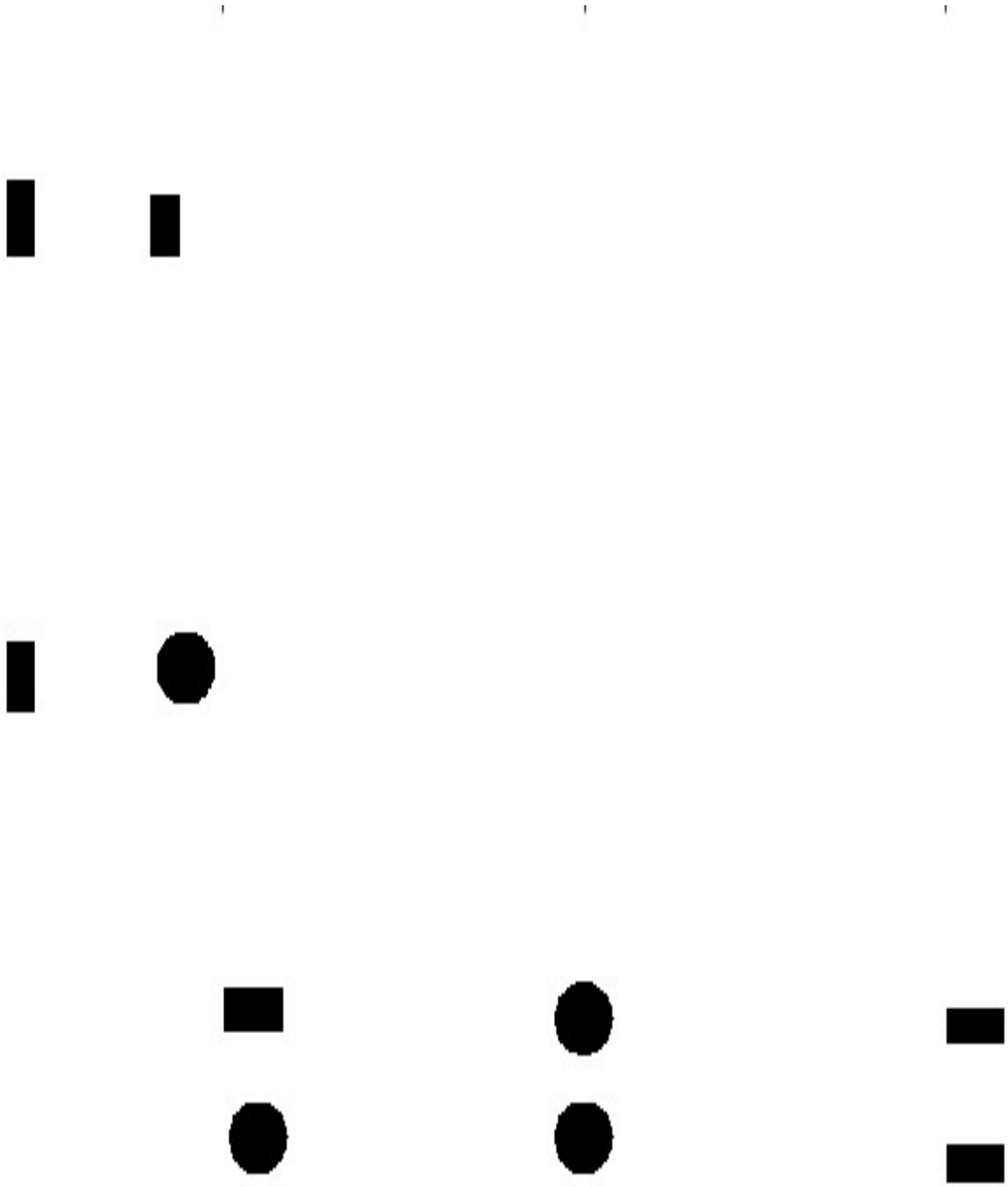

Figure 2.1: Obstacles in surveillance area

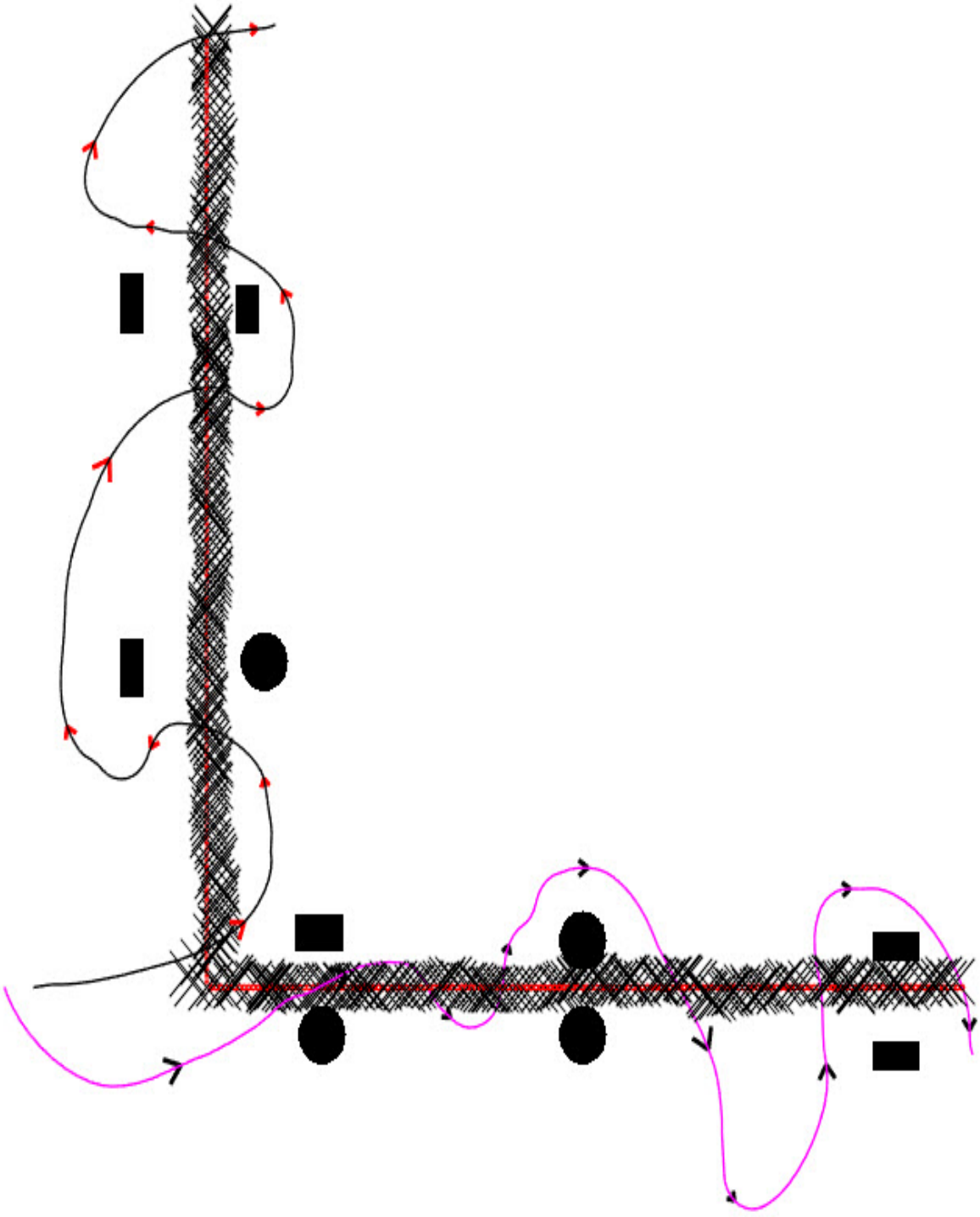

Figure 2.2: UAVs following targets through obstacles

2.1 Partially Observable Markov Decision Process Approach

The original POMDP formulation in [8] is used for defining the UAV path-planning problem, as described below.

States. The POMDP state used in the P-UAV algorithm includes three segments according to the UAV sensors, the targets, and the tracker. The state at time j is given by $\mathbf{x}_j = (c_j, tg_j, e_j, \mathbf{B}_j)$, where c_j denotes the sensor state, tg_j denotes the target state, and $(e_j, \mathbf{B}_j)$ denotes the tracker state. The sensor state includes the UAV positions and velocities. The target state includes the target positions, velocities, and accelerations. The tracker utilizes a simple Kalman filter, and therefore the tracker state is defined as the Kalman-filter state, where e_j indicates the posterior mean vector and $\mathbf{B}_j$ indicates the posterior covariance matrix.

Actions. The P-UAV algorithm generates each control action for guiding the UAVs as forward acceleration and bank angle. The action at time j is given by $d_j = (f_j, \alpha_j)$, where f_j is the acceleration vector and α_j is the bank-angle vector.

Observation and Observation Law. Our P-UAV algorithm assumes the tracker and the sensor states to be fully observable, but the target states are partially observable. The observation of the position of the ith target at time j is $z_j^i = H_j tg_j^i + w_j^i$, where matrix H_j applies to every target, and w_j^i represents additive zero-mean Gaussian noise.

State-Transition Law. The state-transition law in the P-UAV algorithm is divided into separate laws for sensors, targets, and trackers. The transition law of each sensor is represented as $c_{j+1} = f_c(c_j, d_j)$, where a function f_c represents a simple kinematic function, defined later. The control action at time j for target i is $d_j^i = (f_j^i, \alpha_j^i)$, where the acceleration is represented as f_j^i and the bank angle is represented as α_j^i. The joint probabilistic data association (JPDA) approach is utilized in the Kalman filter for the tracker-state evaluation.

The UAV direction is forced by a heading angle $\omega_{j+1}^i = \omega_j^i + (GT\tan(\alpha_j^i)/V_j^i)$, where V_j^i indicates the speed, G indicates the gravitational constant, and T indicates the time-step duration. The UAV speed is updated by $V_{j+1}^i = [V_j^i + f_j^i T]_{V_{\min}}^{V_{\max}}$, where $[v]_{V_{\min}}^{V_{\max}} = \max\{V_{\min}, \min(V_{\max}, v)\}$, $V_{\max}$ in-

dicates the UAV maximum speed, and $V_{\min}$ indicates the UAV minimum speed. The UAV position is updated by $m^i_{j+1} = m^i_j + V^i_j T\cos(\omega^i_j)$ and $n^i_{j+1} = n^i_j + V^i_j T\sin(\omega^i_j)$.

The updated target state is defined by $tg_{j+1} = f_t(tg_j) + v_j$, where the target dynamics is based on linear motion with zero-mean noise: $tg^i_{j+1} = \mathbf{F}_j tg^i_j + v^i_j$, $v^i_j \sim \mathcal{N}\left(0, \mathbf{Q}_j\right)$, $i \in \{1,\ldots,N_{\text{tar}}\}$, where $\mathbf{F}_j$ is the speed transition matrix.

Cost Function. The P-UAV algorithm aims to minimize the mean-squared error of the tracks and the targets according to $C(\mathbf{x}_j, d_j) = \mathrm{E}_{v_j, w_{j+1}}\left[||tg_{j+1} - e_{j+1}||^2 \,\middle|\, \mathbf{x}_j, d_j\right]$.

Belief State. The P-UAV algorithm employs the belief state $b_j = (b^c_j, b^{tg}_j, b^e_j, b^B_j)$. The belief state of the fully observable sensor and tracker states is given by $b^c_j = \delta(c - c_j)$, $b^e_j = \delta(e - e_j)$, and $b^{\mathbf{B}}_j = \delta(\mathbf{B} - \mathbf{B}_j)$ sequentially. The belief state of the partially observable target state is computed from the normal distribution $b^{tg^i}_j = \mathcal{N}\left(e^i_j, \mathbf{B}^i_j\right)$.

2.2 Monte Carlo Tree Search

The MCTS process is basically an approach to solve a MDP. MCTS can be applied to a POMDP by using the belief state as the state of a MDP, called the *belief MDP*. A decision tree in a MCTS process is utilized in which states correspond to nodes and state transitions correspond to directed edges.

The two policies used in the MCTS approach are the *tree* policy and the *default* policy [13]. Finding an optimal solution involves an exploration-exploitation tradeoff in the tree policy. *Exploration* involves trying out new actions to learn something new about their impact, whereas *exploitation* involves actions to optimize the rewards. The tradeoff corresponds to finding the right balance between trying new things and enjoying the rewards of actions taken. We use the upper confidence bound for trees (UCT) method to achieve this tradeoff. In the simulation process, the default policy is utilized to compute approximately the values of the cost function at the leaf nodes.

There are four stages in the MCTS method, which begins by creating the root node of the search tree, the current state, and actions leading to child nodes. After that, MCTS uses the following four stages to find optimal solutions.

Selection stage: This stage begins at the root node of the search tree and chooses a child node using the upper confidence bound for trees (UCT) method until arriving at a leaf node.

Expansion stage: If a leaf node chosen in the selection stage is not a terminal node, the expansion stage is used to generate new child nodes by using a random sampling method. In particular, the MCTS method applies the expansion stage according to the depth of a node.

Simulation stage: From the child node in the expansion stage, the simulation stage evaluates the value of the node by applying a random policy until a terminal node is reached.

Backpropagation stage: The backpropagation stage updates the numerical values of the node evaluated by the previous simulation stage along the path back to the root node. After this stage is finished, the MCTS method returns to the selection stage.

Chapter 3

POMDP Methodology

3.1 Preliminaries

POMDPs are used for planning problems with partial observations. A POMDP is a stochastic optimal control problem and is hard to solve exactly. Formulated as an equivalent MDP, the states in a POMDP (belief state), are distributions, leading to an uncountably infinite state space. The usual approach is to apply certain approximations. First, the states can be approximated in terms of finite-dimensional objects. Second, we can directly approximate the Q-value $Q(b,d)$, where b represents a belief state and d represents an action. Alternatively, we can indirectly approximate the Q-value by estimating the cost-to-go value $J^*(b)$, where b represents a belief state.

For example, in the Nominal Belief-State Optimization (NBO) method [8], the Q-value $Q(b,d)$ is approximated by approximating the cost-to-go value $J^*(b)$ as $J^*(b) \approx \min_{(d_j)} \sum_j cost(\hat{b}_j, d_j)$. Building on this, in [15], the Q-value with action d taken at belief state b_0 is computed according to

$$Q_H(b_0,d) = cost(b_0,d) + \mathrm{E}\left[J^*_{H-1}(b_1) \,\middle|\, b_0,d\right]. \tag{3.1}$$

The approximate Q-value in the NBO technique is calculated from a belief-state sequence $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{H-1}$ and action sequence $d_1, d_2, \ldots, d_{H-1}$ corresponding to the tracking model, Gaussian distribution, and the data association method.

In the tracking problem, the belief state sequence of the ith target is approximated in terms of the track $(\hat{e}^i_j, \hat{\mathbf{B}}^i_j)$ from the Kalman filter equations without noise: $\hat{b}^{tg^i}_j = \mathcal{N}(\hat{e}^i_j, \hat{\mathbf{B}}^{\mathbf{i}}_{\mathbf{j}})$, $\hat{e}^i_{j+1} = \mathbf{F}_j \hat{e}^i_j$, and

$$\hat{\mathbf{B}}^i_{j+1} = \begin{cases} \left[[\hat{\mathbf{B}}^i_{j+1|j}]^{-1} + \mathbf{C}^i_{j+1}\right]^{-1}, & \text{the observation available,} \\ \hat{\mathbf{B}}^i_{j+1|j}, & \text{in other ways,} \end{cases} \tag{3.2}$$

where

$$\hat{\mathbf{B}}^i_{j+1|j} = \mathbf{F}_j \hat{\mathbf{B}}^i_j \mathbf{F}^{\mathrm{T}}_j + \mathbf{Q}_j,$$
$$\mathbf{C}^i_{j+1} = \mathbf{H}^{\mathrm{T}}_{j+1} \left[\mathbf{R}_{j+1}\left(\hat{e}^i_{j+1}, c_{j+1}\right)\right]^{-1} \mathbf{H}_{j+1},$$

and $c_{j+1} = f_c(c_j, d_j)$. The cost function is given by the mean-squared error between the tracks and the targets, calculated using

$$cost(\hat{b}_j, d_j) = \sum_{i=1}^{N_{\mathrm{tar}}} \mathrm{Tr}\, \hat{\mathbf{B}}^i_{j+1}. \tag{3.3}$$

In equation (3.1), the second term is generally difficult to calculate because the belief states are probability distributions over the state space. Accordingly, belief states are approximated as finite objects.

3.2 The P-UAV Algorithm with Obstacle Collision Avoidance

To incorporate obstacle collision avoidance, we assume that each UAV flies at an altitude that is lower than the height of the obstacles while following the targets that move on the ground in the presence of these obstacles. We use a penalty term in the cost function in [7], where the value of this term increases as a UAV flies closer to an obstacle within a distance of 100 meters, as shown in detail in Algorithm 1. The minimum distance between the nth UAV, UAV_n, and any obstacle is calculated in the first step in line 4. The penalty is applied only at distance of less than 100 meters. If the distance to an obstacle is smaller than 100 meters, the square difference of this distance from 100 meters is multiplied by a constant and added to the distance to obtain the penalty value (in line 6). The cost function is then calculated as

$$cost(\hat{b}_j, d_j) = \sum_{i=1}^{N_{\mathrm{tar}}} \mathrm{Tr}\, \hat{\mathbf{B}}^i_{j+1} + \sum_{k=1}^{N_{\mathrm{sen}}} Obs^k_{j+1}, \tag{3.4}$$

where $\sum_{k=1}^{N_{\mathrm{sen}}} Obs^k_{j+1}$ is the penalty term described above.

Algorithm 1 Penalty calculation for obstacle collision avoidance

```
1: procedure PENALTY
2:     Obs ← 0
3:     for y ← 1,n do/*n indicates the number of UAVs
4:         Obs ← min (distance of UAV_n and all obstacles)
5:         if Obs < 100 then
6:             Obs ← Obs + Constant * (100 − Obs)^2
7:         end if
8:     end for
9:     return Obs
10: end procedure
```

POMCP is an extension of MCTS to partially observable environments, using a sampling method to move from the initial belief state to a next belief state in an online fashion. Each node of the search tree represents a belief state, which contains a history of the actions and observations. In node h, the Q-value $Q(h,d)$ is calculated by averaging over the a set of simulated trajectories that start with taking action d at node h. We then select an action $d(ucb1)^*$ using the Upper Confidence Bounds (UCB1) as follows:

$$d(ucb1)^* \leftarrow \underset{h \in node}{\arg\max}(Q(h,d) + k\sqrt{\frac{\log Nb(h)}{Nb(hd)}}), \tag{3.5}$$

where $Nb(hd)$ represents the number of simulations at node hb with action d selected, and $Nb(h)$ represents the number of visits to node h.

The P-UAV algorithm computes the approximate Q-value $Q(h,d)$ by employing

$$Q(h,d) \approx cost(b,d). \tag{3.6}$$

The P-UAV algorithm is shown in detail in Algorithms 2 and 3. We implemented the method by applying a parallel technique with Matlab on a multicore processor. The algorithm typically begins by processing the root node $h = (x,d,cost(\hat{b},d),Q(h,d),Nb(h),Nb(hd))$ of the search tree, where x indicates the states of the sensors, the targets, and the tracker, d indicates the actions,

$cost(\hat{b},d)$ indicates the cost function of all child nodes, $Q(h,d)$ indicates the Q-value using the UCT method, $Nb(h)$ indicates the visit count of h, and $Nb(hd)$ indicates the visit count of h with action d.

3.3 The Parallel-Computing Strategy with a Heuristic Random-Sampling Process

Traditional UAVs designed by using a fixed-wing aircraft control its direction by adjusting bank angle and its speed by changing forward acceleration. In general, a principle of aerial vehicle motion includes travelling in a straight line and changing its direction, or turning. Aerial vehicle travelling in a straight line can be represented as a normal state with a zero angle of bank angle for its direction. When the turn has been completed vehicle must roll back to a normal state in order to resume a straight movement. Therefore, the straight movement is commonly utilized in vehicle motion control as a normal state. This heuristic of vehicle direction control is applied to our P-UAV algorithm by setting a zero angle of bank angle with a random-sampling approach, called a heuristic random-sampling process.

The zero angle of bank angle of UAV in our heuristic random-sampling technique is necessary for finding the shortest distance from UAV toward target after vehicle turning, as shown in Figure 3.1-3.6. Our algorithm can create the minimum distance of trajectories leading to save time and fuel. Reviewing the MCTS papers in autonomous vehicle, our heuristic random-sampling process have not been found in any proposals.

The control of the UAV direction uses three groups of angles with positive, zero, and negative. Our heuristic process inserts the zero angle of bank angle to every level of a time horizon H from the root node of MCTS tree search to a node of the maximum depth of tree. In simulation stage, our P-UAV method uses this heuristic knowledge with three groups of angles (positive, zero, and negative) to process the rollout technique for one level of a time horizon in the limited number of loop. After that, the maximun value is seeked for updating value from branch node in the level of maximum depth to the root node.

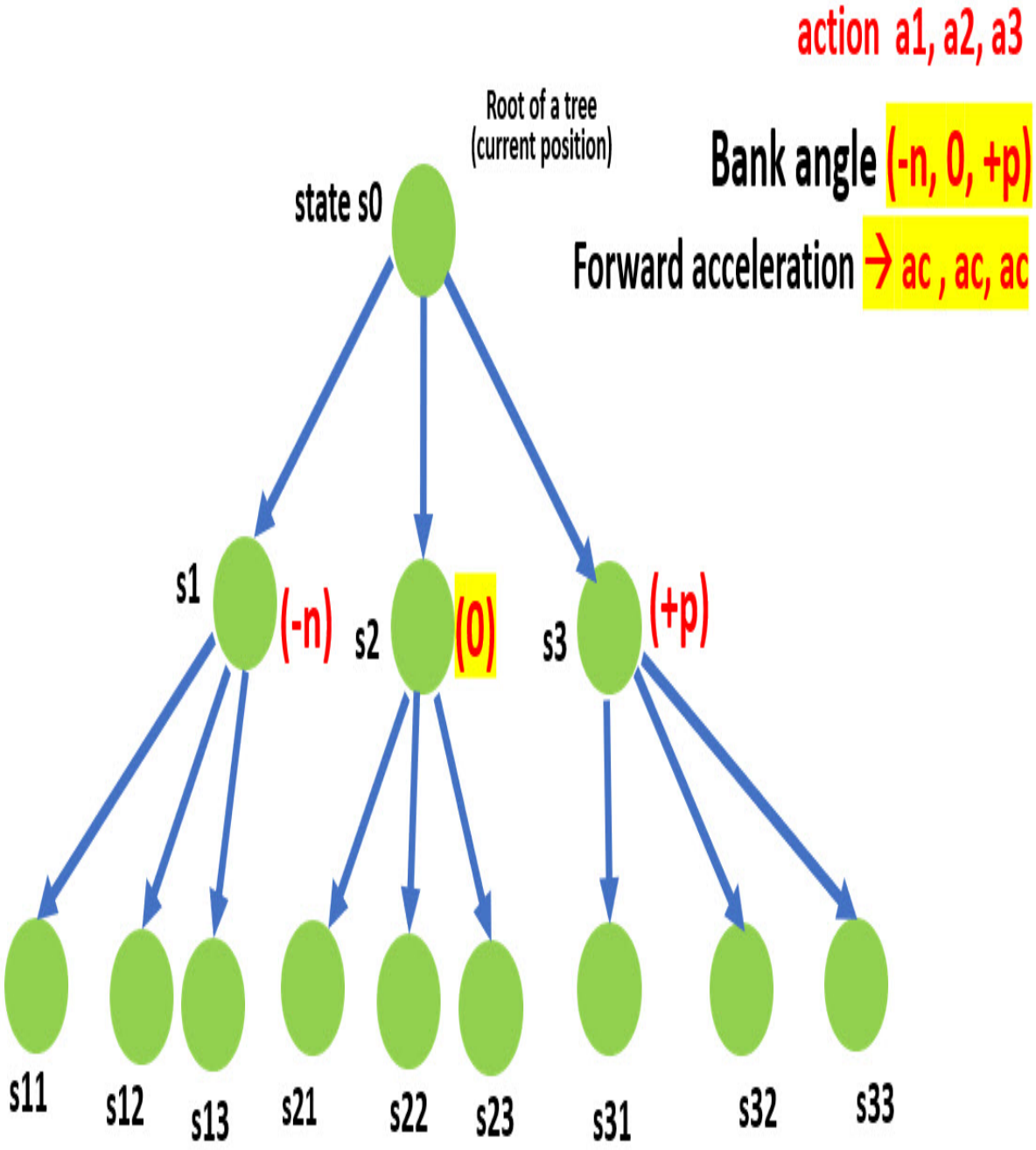

Figure 3.1: The heuristic random-sampling process (depth level=1)

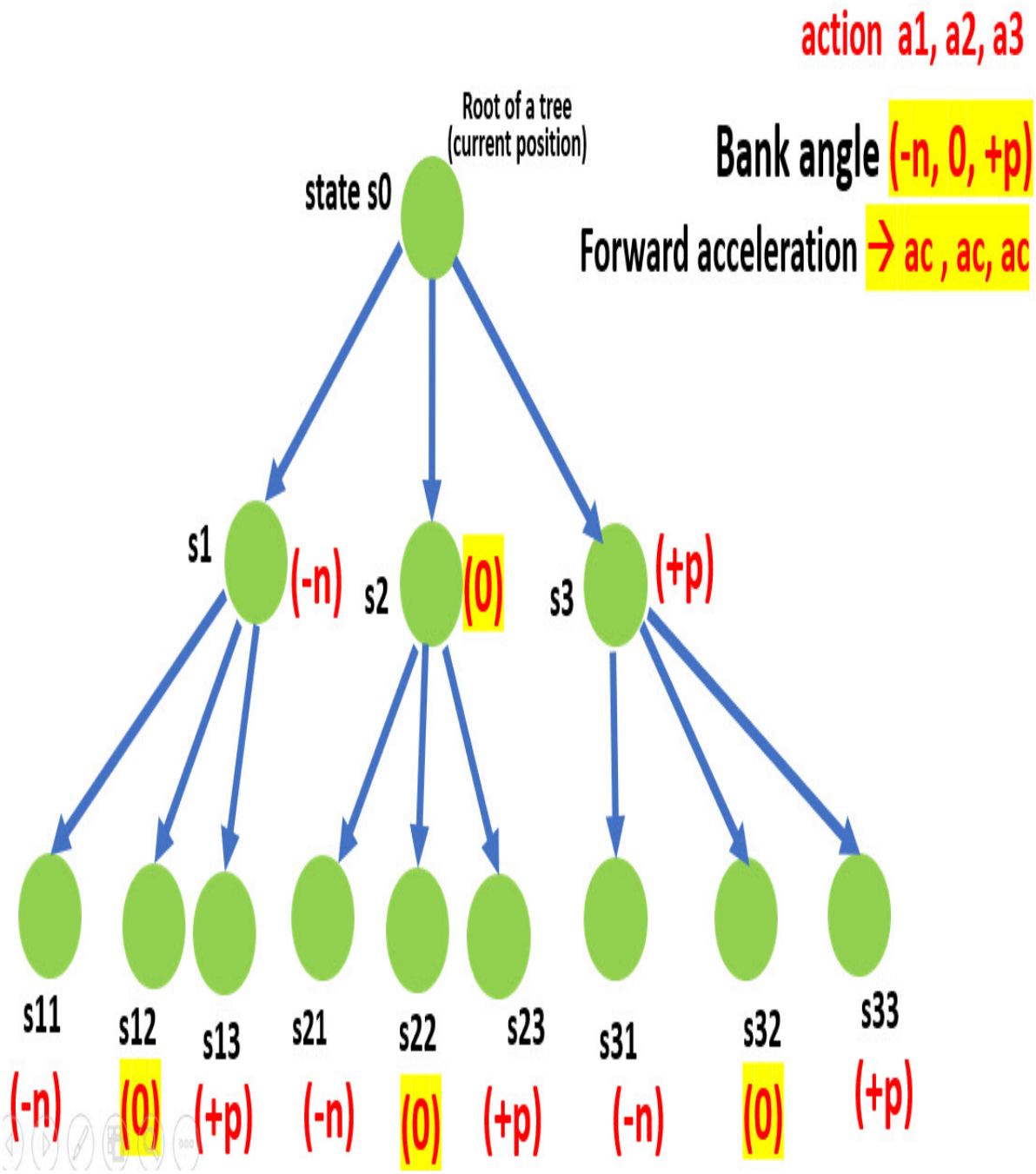

Figure 3.2: The heuristic random-sampling process (depth level=1, 2)

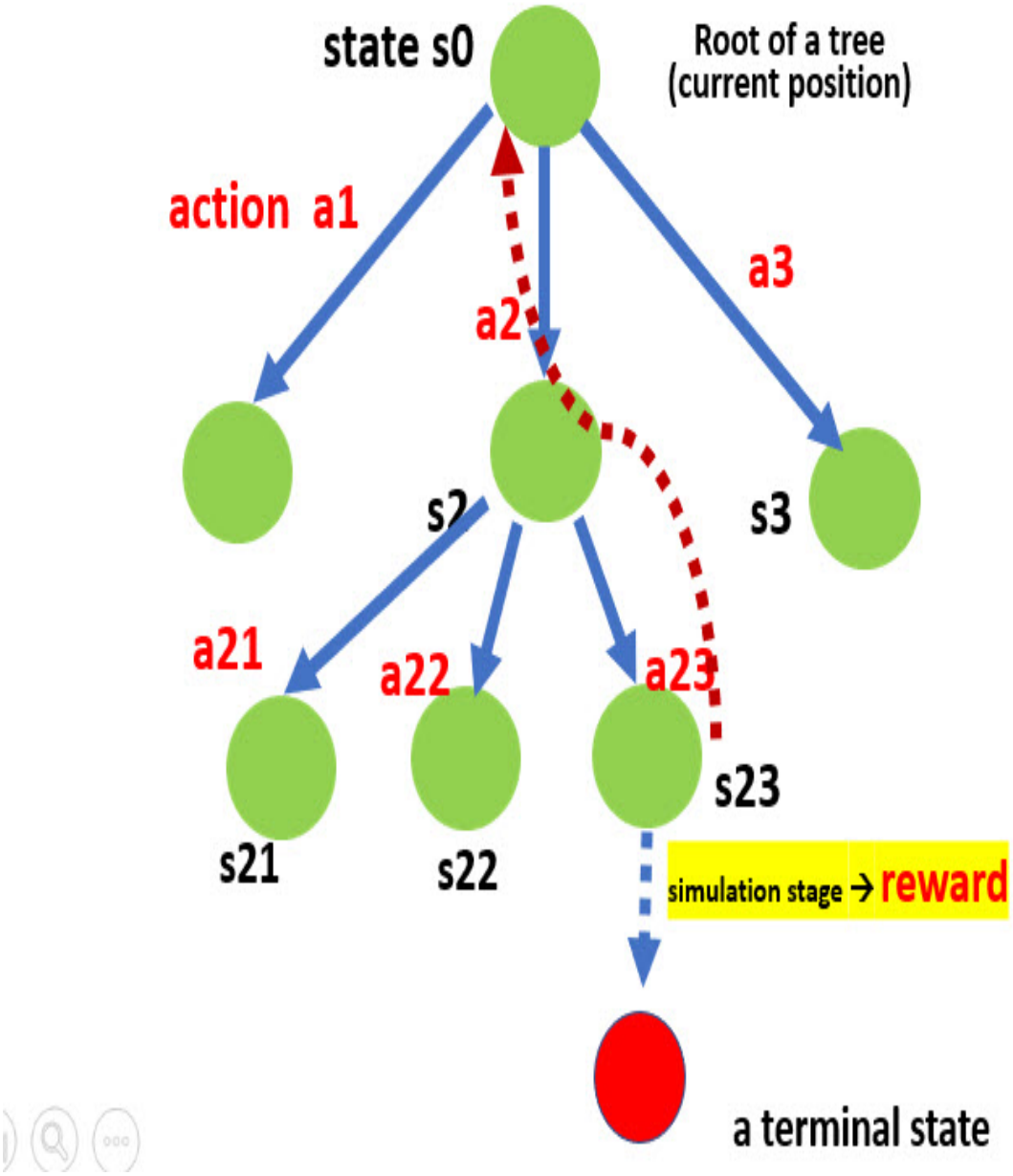

Figure 3.3: The heuristic random-sampling process: simulation stage with update

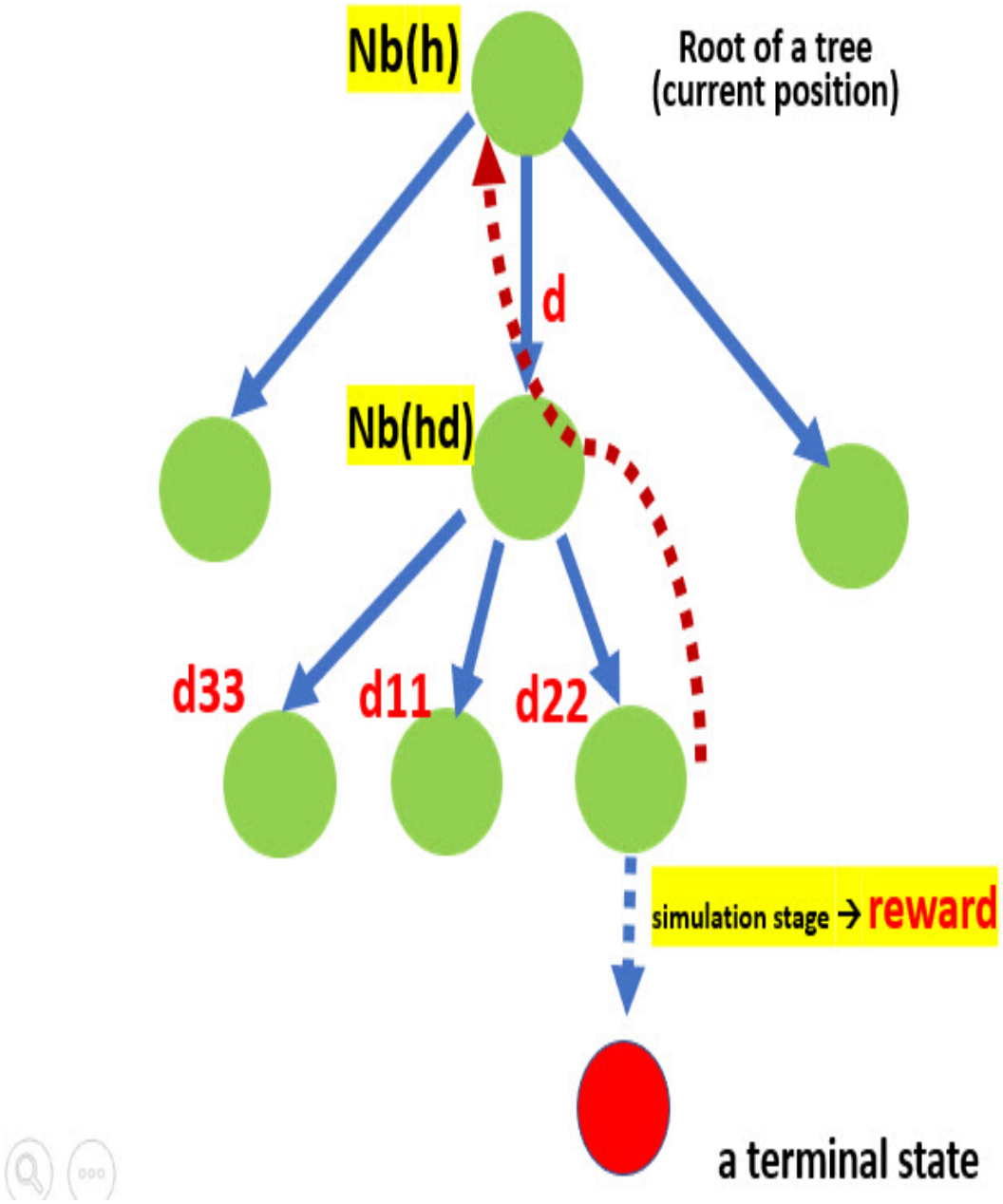

Figure 3.4: The heuristic random-sampling process: simulation stage with the value updated

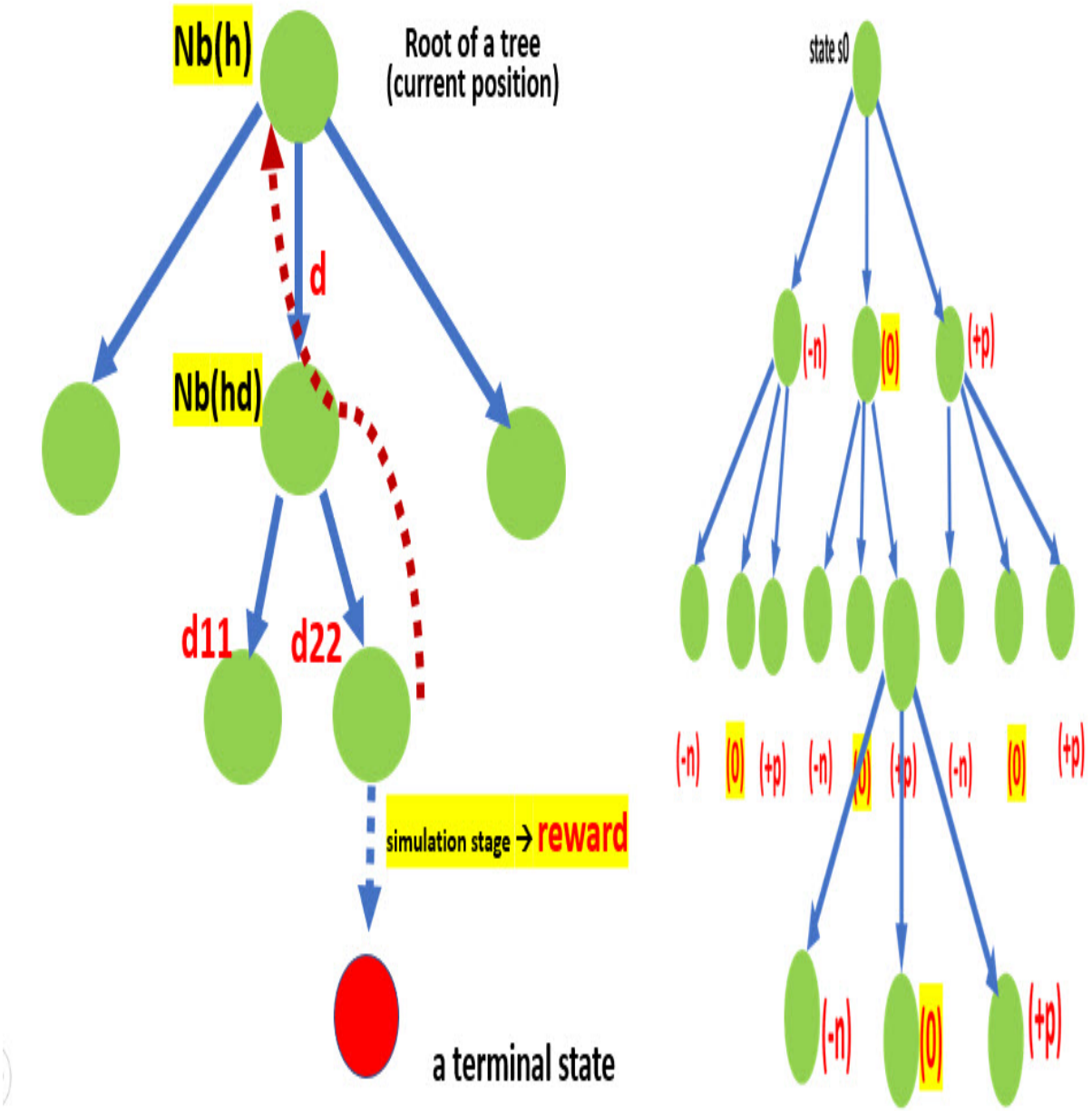

Figure 3.5: The heuristic random-sampling process: simulation stage with the values of bank angle (1)

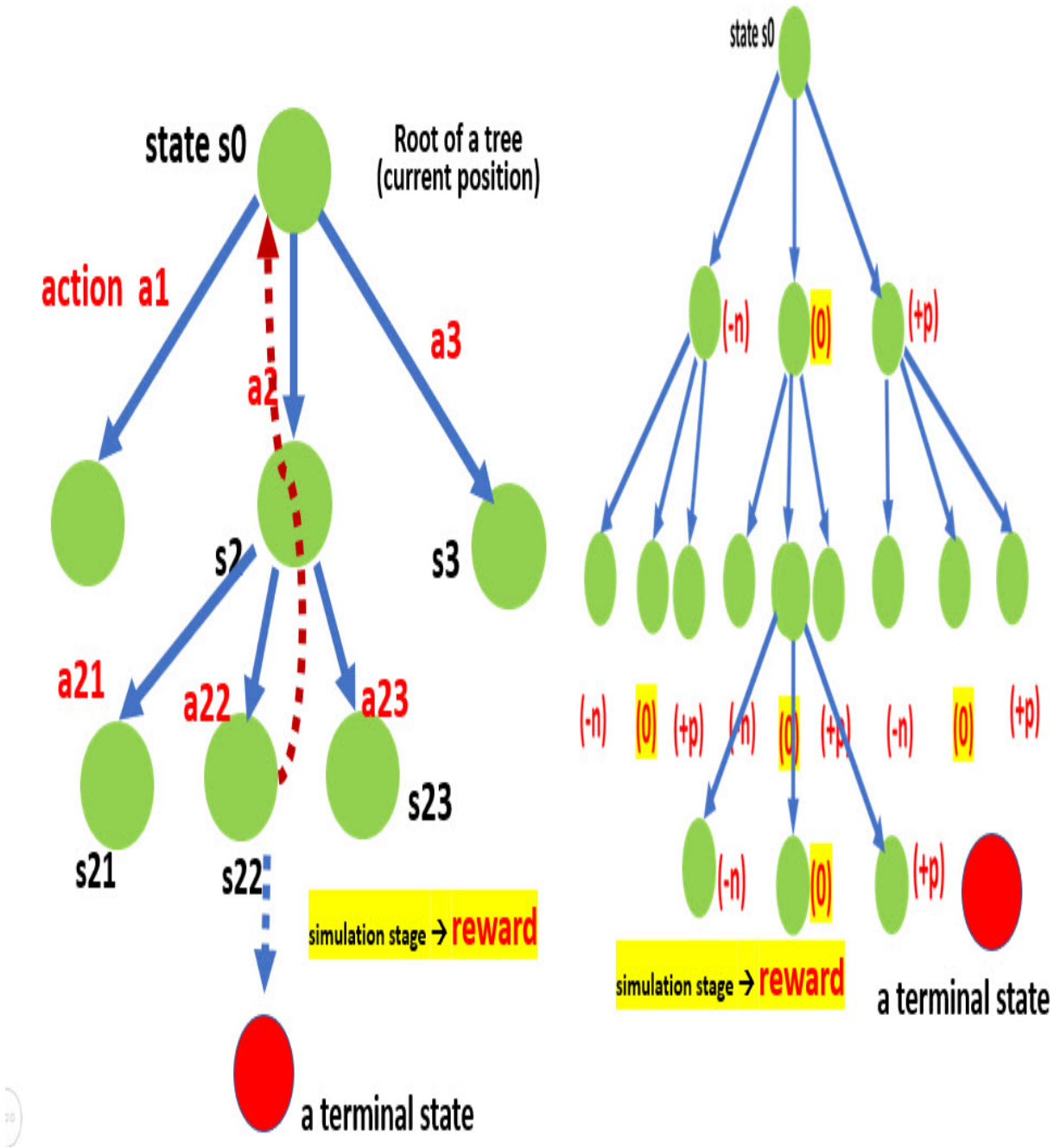

Figure 3.6: The heuristic random-sampling process : simulation stage with the values of bank angle (2)

3.4 Wind Effects on UAVs

Wind can affect UAVs's stability causing aerial accidents. There are several type of wind such as wind shear, propeller vortex, constant wind, turbulent flow. This research focuses on the P-UAV

method with a constant wind-reciprocation technique based on the POMCP technique. The wind state indicated as the velocity in x- and y- axis is entered the P-UAV algorithm for computing wind reciprocation to achieve UAVs's mission.

Let sw_{x-k} and sw_{y-k} be the wind velocity in x- and y- axis for applying to the autoregressive equation $sw_{x-k} = \sum_j^p K_j sw_{x-k+1} + e_{k+1} j$, $sw_{y-k} = \sum_j^p K_j sw_{y-k+1} + e_{k+1} j$, where $K_1, ..., K_p$ indicates the autoregressive coefficients and $e_{k+1} \sim N(0, Q)$. The wind velocity changes according to time

We eliminate the small effect of wind and apply the approximate wind velocity to The P-UAV algorithm. The UAV position is updated with the wind effection by $m^i_{j+1} = m^i_j + V^i_j T \cos(\omega^i_j) + sw_{x-k}$ and $n^i_{j+1} = n^i_j + V^i_j T \sin(\omega^i_j) + sw_{y-k}$. The P-UAV path planning can generate optimal trajectories of UAVs with a wind effect. Our path planning use a look-ahead concept, so it can predict the future right location of UAVs in the wind situation.

The approximate-wind velocity is calculated by using the Kalman filter to track the wind-velocity vector. Aerial vehicle can detect the true airspeeds by utilizing a pitot tube, vehicle velocity with direction is measured by Global Positioning System (GPS). So, the wind velocity and direction can be calculated.

Parallel computing techniques can improve computation times and are often exploited in Monte Carlo approaches. We use the Parfor-loop in Matlab to process sequential statements in the algorithm's main loop. We control the UAV direction in terms of three groups of angles (positive, zero, and negative). This strategy helps to reduce the computation time. The control action is denoted as $d_j = (f_j, \alpha_j)$, where $f_j \in [-3, 3]$, $\alpha_j = \{[-\alpha, 0), 0, (0, \alpha]\}$, and $\alpha = \frac{\pi}{12}$.The tracker state $(e_j, \mathbf{B}_j)$ is computed by a Kalman filter, giving the cost function $cost(\hat{b}_j, d_j)$ by incorporating the penalty $\sum_{k=1}^{N_{\text{sen}}} Obs^k_{j+1}$ associated with obstacle avoidance, key to our approach. We then calculate $Q(h,d)' \leftarrow cost(\hat{b}_j, d_j)$. The Q-value $Q(h,d)$ is updated using $Q(h,d) \leftarrow Q(h,d) + \frac{Q(h,d)' - Q(h,d)}{Nb(hd)}$.

3.5 Threat and UAV Avoidance in Mobile Target Tracking

Avoiding and monitoring threat. One of the most notable concerns about UAV applications prevent accidents, keep safety in flight, and monitor their environment.

Threats movement is indicated as the location in x-direction and y-direction, and a velocity with direction, according to $Thr_j = [Loc_j, V_j, \varepsilon_j]$. The state of threat is represented as $Thr_{j+1} = \Phi_j Thr_j + n_j, \quad n_j \sim N(0, Q_j^t)$,where Q_j^t is a matrix of process noise covariance. We apply a heuristic method for the Kalman filter to find the threat state by solving normally distributed random variables with known means and standard deviations. The value of $(\hat{e}_j^i, \hat{\mathbf{B}}_j^i)$ from the Kalman filter equations : $\hat{b}_j^{tg^i} = \mathscr{N}(\hat{e}_j^i, \hat{\mathbf{B}}_\mathbf{j}^\mathbf{i})$, $\hat{e}_{j|j-1}^i = \Phi_j \hat{e}_{j-1}^i$, $\hat{\mathbf{B}}_{j|j-1}^i = \Phi_j \hat{\mathbf{B}}_{j-1}^i \Phi_j^{\mathrm{T}} + \mathbf{Q}_j$

We obtain the belief state from the previous step and calculate a distance between UAVs and threats as shown in Algorithm 2.

$$cost(\hat{b}_j, d_j) = \sum_{i=1}^{N_{\text{tar}}} \mathrm{Tr}\hat{\mathbf{B}}_{j+1}^i + \sum_{k=1}^{N_{\text{sen}}} Obs_{j+1}^k + \sum_{k=1}^{N_{\text{sen}}} Threat_{j+1}^k, \tag{3.7}$$

Algorithm 2 Threat-penalty calculation for threat collision avoidance

```
1:  procedure THREAT PENALTY(the estimated current threat location )
2:      Threatlocation ← HeuristicKalman(EstimatedCurrentThreatlocation)
3:      Threat ← 0
4:      for y ← 1,n do/*n indicates the number of UAVs
5:          Threats ← min (distance of UAV_n and all threats)
6:          if Threat < 100 then
7:              Threats ← Threats + Constant ∗ (100 − Threats)^2
8:          end if
9:      end for
10:     return Threats
11: end procedure
```

Algorithm **??** includes four procedures. The SelectPhase procedure implements the selection phase by selecting a child node for the expansion phase. The HeuSampGennode procedure applies

a best-first strategy according to the UCT equation. The UpdateC procedure computes the Q-value. The SimulPhase procedure implements the simulation phase by randomly choosing control actions in accordance with the HeuSampGennode procedure. In the loop iterations, we use parallel computing to compute $Q_{max,o}$ in each loop and finally find the highest value over all loops.

Algorithm 3 P-UAV algorithm with obstacle and threats avoidance

procedure MAIN P-UAV ALGORITHM($Deepness(tree)$, x)
  $h \leftarrow x$, d, $cost(\hat{b},d)$, $Q(h,d)$, and statistical values
  $h^{'},d \leftarrow$ HeuSampGennode(h), $Pdep \leftarrow 1$
  for $L \leftarrow 1,p$ **do** /*Parallel loops
    for $W \leftarrow 1,o$ **do**
      while $Pdep \neq Deepness(tree)$ **do**
        $d^{*} \leftarrow$ SelectPhase($h^{'},d$)
        if $h^{'} \neq$ a child node of T **then**
          $h^{''} \leftarrow$ HeuSampGennode($h^{'}(d^{*})$)
        end if
        $h^{'} \leftarrow h^{''}$, Pdep=Pdep+1
      end while
      $cost(b,d) \leftarrow$ SimulPhase(h_{Pdep+1}), $Pdep \leftarrow 1$
      $Q(h,d) \leftarrow$ UpdateC($cost(\hat{b},d)_{h(0,Pdep)}$)
    end for
    $Q_{max,o} \leftarrow \max\{Q(h,d_1),Q(h,d_2),Q(h,d_3)\}$
  end for
  $d_{max} \leftarrow \max\{Q_{max,o,1}(h,d_j),..,Q_{max,o,p}(h,d_j)\}$
  return d_{max};
end procedure

Algorithm 4 Subroutines used in P-UAV with obstacle and threat avoidance

procedure HEUSAMPGENNODE(h)
  $d=(f,\alpha)$, $f\in[-3,3]$, $\alpha=\{[-\alpha_q,0),0,(0,\alpha_q],\}$,
  $\alpha_q=\frac{\pi}{12}, f=rand[-3,3], \alpha(1)=rand[-\alpha,0)$,
  $\alpha(2)=0, \alpha(3)=rand(0,\alpha]$
  $h'(d)\leftarrow$ generate new history nodes to node h
  $(e,\mathbf{B})\leftarrow$ Kalmanfilter($h'(d)$) /*generate tracker
  $cost(\hat{b},d)=\sum_{i=1}^{N_{\text{tar}}}\text{Tr}\hat{\mathbf{B}}+\sum_{k=1}^{N_{\text{sen}}}Obs_{j+1}^{k}$,
  $h'(d)\leftarrow cost(\hat{b},d)$
  return $h'(d)$;
end procedure
procedure UPDATEC($c(\hat{b},d)_{h(0,depth)}$)
  for $h_0 \quad to \quad h_{depth}$ **do**
    $Q(h,d)'\leftarrow cost(\hat{b},d)$
    $Nb(h)\leftarrow Nb(h)+1, Nb(hd)\leftarrow Nb(hd)+1$
    $Q(h,d)\leftarrow Q(h,d)+\frac{Q(h,d)'-Q(h,d)}{Nb(hd)}$
  end for
end procedure
procedure SELECTPHASE(h,d)
  $d(h')\leftarrow \arg\max_{h\in node}(-\mathbf{Q}(h,d)+\sqrt{\frac{\log Nb(h)}{Nb(hd)}})$
  return $d(h')$;
end procedure
procedure SIMULPHASE(h)
  $d^*\leftarrow 0$
  while $loop<kt$ **do**
    $h^{'},d\leftarrow$ HeuSampGennode(h)
    $dl\leftarrow$ SelectPhase($h^{'},d$)
    if $dl>d^*$ **then** $d^*\leftarrow dl$
    end if
      $h\leftarrow h^{'}$
  end while;
  return $cost(\hat{b},d^*)$;
end procedure

Chapter 4

Design of Autonomous UAV Guidance System Using Monte Carlo Tree Search

4.1 Overview

In this paper, we propose an online path-planning algorithm using a variation of Monte Carlo tree search (MCTS) for navigating intelligently unmanned aerial vehicles (UAVs) to track mobile ground targets, called the P-UAV algorithm. The proposed algorithm employs a non-myopic method applied to a partially observable Markov decision process (POMDP) model, accounting for long-term decision making. Our algorithm integrates a heuristic technique to efficiently generate paths. The algorithm yields to parallel processing methods to significantly enhance its computational performance, making it suitable for realtime implementation. Simulation experiments show that our path-planning algorithm is efficient and achieves good exploration-exploitation tradeoff in finding a near-optimal solution despite the very large search space.

keywords:

Path planning, Partially observable Markov decision processes, Monte Carlo tree search, An online path planning algorithm, Unmanned aerial vehicles (UAVs)

4.2 Introduction

Unmanned aerial vehicles (UAVs), also known as drones, are widely used in defense and commercial applications due to their ease of use, low maintenance cost, and high mobility performance. Information processing in UAVs use embedded onboard systems, and can be exploited for autonomous positioning and tracking. These systems incorporate the global positioning system (GPS), communication capabilities, autonomous control, image processing, and artificial intelligence (AI). Nonetheless, interest persists in more intelligent UAVs to support future applications,

simultaneouly reducing human effort and achieving higher performance goals. A major challenge in intelligent UAV development is path planning in dynamic environments with constraint, differing significantly in some aspects from traditional robot path planning. For example, a typical UAV has a relatively large turning radius and must maintain a minimum speed.

To aid in path planning, UAVs can be equipped with a variety of sensors and onboard computing capabilities. However, onboard path planning exploiting sensor measurements is computationally expensive. The UAV path-planning problem is often posed as an optimization problem with a very large set of feasible candidate solutions. The problem is exacerbated by the need to compute paths continuously in real-time in response to dynamically changing conditions, especially when tracking mobile ground targets. Partially observable Markov decision processes (POMDPs) provide a natural framework to capture the desired features of the path-planning problem.

A POMDP is a generalization of a Markov decision process (MDP), combining a MDP to model system dynamics and a hidden Markov model for the observations of unobservable states. Finding the optimal solution to a POMDP problem is generally intractable [6], necessitating approximation methods. These include heuristics, policy rollout, hindsight optimization, foresight optimization, and nominal belief-state optimization (NBO) [8]. In particular, the NBO method is an ADP approach developed for UAV path planning [8], [9], [10] based on a POMDP formulation. In other related work, [11] solves an object search problem for a space confined area, formulated as a POMDP, and uses the online DESPOT system [12] to find the shape of the area for planning.

In a POMDP, observations follow an observation law (conditional distribution) that depends on the current state and control action. It turns out that a POMDP can be reduced to a standard MDP in which the state space is the posterior distribution (in the Bayesian sense) of the underlying unobservable state given the history of observations. This posterior distribution is called the *belief state*. The set of possible belief states is a continuous state space; when discretized, the number of possible states is still often extremely large. Monte Carlo tree search (MCTS) is a sampling-based technique [7] to find approximate MDP solutions. Partially observable Monte Carlo planning

(POMCP) [7] aims to solve large POMDPs based on MCTS [8]-[10], typically using particle filters for belief-state representation in the search tree.

This paper proposes an online path-planning algorithm, called P-UAV, based on POMCP to guide UAVs for tracking mobile ground targets. P-UAV differs from traditional POMCPs because it uses a Kalman filter instead of using a particle filter to update the belief-state in the search tree. Our P-UAV algorithm specifically addresses extends the POMCP framework to the target-tracking problem formulated as a POMDP. It integrates a heuristic technique to control the UAV acceleration and bank angle subject to constraints to produce an efficient non-myopic path planning method. Moreover, we incorporate parallel processing into P-UAV to improve its computational performance, making it suitable for real-time applications.

Monte Carlo tree search (MCTS) was applied to the well-known Go-playing program by Coulom [13] in 2006. MCTS searches for near-optimal solutions by combining a best-first approach with an adaptive sampling technique using the upper-confidence-bound (UCB) approach. The method aims to achieve a good tradeoff in exploration versus exploitation in finding near-optimal solutions with large state-spaces. This is achieved by focusing the search on the most promising candidate solutions, selected based on objective-function evaluations of feasible solutions. MCTS is particularly suited to problems with expensive function evaluations. MCTS has been used in a variety of problems, ranging from aerial glider control for collecting information to object recognition in image processing for online decision making.

POMCP [7] was introduced by Silver and Veness in 2010. The method extends MCTS to the case of partial observations by associating action and observation histories to a node of the search tree. A node of the search tree contains a history and a numerical value for that history. A history is a sequence of actions and observations $h_t = \{a_1, o_1, ..., a_t, o_t\}$, and the numerical value of the history is computed using Monte-Carlo simulations for states sampled from the current belief state. Several efforts have used POMCPs for mobile robots and drones using the POMDP framework. Parameterized Action Partially Observable Monte-Carlo Planning (PA-POMCP) was used in [15] for target object search. A partially observable stochastic shortest path (PO-SSP) planning problem

for autonomous urban navigation of UAVs was solved in [14] by using an adapted POMCP method utilizing the traditional UCB algorithm as the action-selection strategy.

In the next section (Section 4.3), we describe how our path-planning problem can be posed in the POMDP framework. Section 4.4 presents our POMDP solution technique and its algorithmic realization, P-UAV. In Section 4.5 we show experimental results to illustrate and evaluate the performance of our approach, comparing it with the existing method of NBO. The paper concludes in Section 4.6.

4.3 Problem Statement

The problem of path planning for tracking ground targets is specified by the various components of the system:

Motion of targets and UAVs. For simplicity, we consider two-dimensional motion where UAVs fly at a constant altitude over the ground, and targets move in a plane on the ground.

Sensors for measurement. On each UAV, a camera and an image-processing system provide visual measurements, assumed to have random errors (see below) but no false alarms and missed detections.

Spatially varying measurement error. The sensor-measurement errors are spatially varying, depending on the relative location of targets and UAVs.

Dynamic constraints. For simplicity, we assume that the motion of each UAV is constrained to have variable speed within a prespecified interval, with bounded lateral acceleration in the plane.

The goal is to compute UAV motion commands to keep track of the ground targets. We capture this as an optimization objective to minimize the mean-squared error between tracks and targets. Notionally, the UAV path-control algorithm is centralized in the sense that we do not address its distributed implementation. The algorithm takes all the sensor measurements, fuses them in the tracker, which then produces updated tracks. It then computes UAV motion commands to control the immediate UAV trajectories. This online process is executed at discrete times at a rate appropriate for the available computational capabilities.

4.3.1 The Partially Observable Markov Decision Process Formulation

We follow the POMDP formulation in [8]. For completeness, we specify the POMDP model below.

States. The state consists of three parts corresponding to the UAV sensors, the targets, and the tracker. The state at time j is denoted by $\mathbf{x}_j = (c_j, tg_j, e_j, \mathbf{B}_j)$, where c_j is the sensor state, tg_j is the target state, and $(e_j, \mathbf{B}_j)$ is the tracker state. The sensor state contains the locations and velocities of the UAVs. The target state contains the locations, velocities, and accelerations of the targets. In more detail, $tg_j = (tg_j^1, tg_j^2, ..., tg_j^{N_{\text{tar}}})$ is the target state, where tg_j^i is the state of the ith target, given by $tg_j^i = [x_j, y_j, v_j^x, v_j^y, f_j^x, f_j^y]^{\text{T}}$, containing the target location (x_j, y_j), target velocity (v_j^x, v_j^y), and target acceleration (f_j^x, f_j^y). The tracker is a regular Kalman filter, so the tracker state is the Kalman-filter state, where e_j represents the posterior mean vector and $\mathbf{B}_j$ represents the posterior covariance matrix.

Actions. The UAV control action consists of the forward acceleration and the bank angle. The action at time j is defined by $d_j = (f_j, \alpha_j)$, where f_j and α_j are vectors containing all of the accelerations and bank angles respectively.

Observation and Observation Law. We assume that the tracker and the sensor states are fully observable but the target states are not. The number of targets is denoted N_{tar}. The UAVs can measure only target locations, by receiving observation $z_j^i = H_j tg_j^i + w_j^i$ from the ith target, where H_j is a matrix (the same for every target) and w_j^i is additive zero-mean noise.

State-Transition Law. We represent the state transition laws for sensors, target, and trackers seperately. The sensor state is $c_{j+1} = f_c(c_j, d_j)$, where f_c is a function representing simple kinematic equations. The ith UAV state at time j is indicated by $c_j^i = \left(m_j^i, n_j^i, V_j^i, \omega_j^i\right)$, where (m_j^i, n_j^i) is the UAV location in two dimensions, and V_j^i and ω_j^i are the speed and the heading angle respectively. The acceleration f_j^i and the bank angle α_j^i of the UAV constitute the action applied to target i at time j as $d_j^i = (f_j^i, \alpha_j^i)$. The Kalman filter is applied with the joint probabilistic data association (JPDA) method to evaluate the tracker state. The UAV heading angle is given by $\omega_{j+1}^i = \omega_j^i + (GT \tan(\alpha_j^i)/V_j^i)$, where G is the gravitational constant, T is the time-step duration,

and V_j^i is the velocity. The updated velocity is calculated according to $V_{j+1}^i = [V_j^i + f_j^i T]_{V_{\min}}^{V_{\max}}$, where $[v]_{V_{\min}}^{V_{\max}} = \max\{V_{\min}, \min(V_{\max}, v)\}$, $V_{\min}$ is the minimum velocity of the UAVs, and $V_{\max}$ is the maximum of the UAVs. The UAV location is updated by $m_{j+1}^i = m_j^i + V_j^i T\cos(\omega_j^i)$ and $n_{j+1}^i = n_j^i + V_j^i T\sin(\omega_j^i)$. The target state follows $tg_{j+1} = f_t(tg_j) + v_j$. The target state dynamics with a linearized target motion model and zero-mean noise is given by $tg_{j+1}^i = \mathbf{F}_j tg_j^i + v_j^i$, $v_j^i \sim \mathcal{N}\left(0, \mathbf{Q}_j\right)$, $i \in \{1, \ldots, N_{\text{tar}}\}$, and $\mathbf{F}_j$ is the velocity transition matrix.

Cost Function. The mean-squared error of the tracks and the targets defines the cost function: $C(\mathbf{x}_j, d_j) = \mathrm{E}_{v_j, w_{j+1}}\left[||tg_{j+1} - e_{j+1}||^2 \,\middle|\, \mathbf{x}_j, d_j\right]$.

Belief State. The belief state is represented as $b_j = (b_j^c, b_j^{tg}, b_j^e, b_j^B)$. The fully observable sensor and tracker states have belief states given by $b_j^c = \delta(c - c_j)$, $b_j^e = \delta(e - e_j)$, and $b_j^{\mathbf{B}} = \delta(\mathbf{B} - \mathbf{B}_j)$ respectively. The partially observable target belief state is given by the normal distribution $b_j^{tg^i} = \mathcal{N}\left(e_j^i, \mathbf{B}_j^i\right)$.

4.3.2 Monte Carlo Tree Search Approach

The MCTS technique is a method to solve MDPs. It can be applied to a POMDP by treating the belief state as the state of an associated MDP (called the *belief MDP*). MCTS uses a decision tree where nodes represent states and directed edges represent state transitions.

The MCTS policy consists of two sub-policies: the tree policy and the default policy [13]. The tree policy is used to trade off the need to *explore* unvisited areas of the search space and the desire to *exploit* the previously visited areas. The default policy used in the simulation process to estimate the cost function values at the terminal nodes based on evaluating the cost of a random policy.

The MCTS procedure is divided into the following four phases: (1) Selection: We start at the root node and traverse down the tree by selecting child nodes based on a selection scheme until we reach a leaf node. (2) Expansion: If we reach a leaf node that is not a terminal node, then each possible child node (state) is visited according to the available actions until we reach a terminal node. (3) Simulation: If we reach a terminal node, then we evaluate its value via simulation. (4)

Backpropagation: For each node evaluated, we update its value and increment the number of its visits. After finishing this phase, we repeat selection phase.

4.4 POMDP Solution Technique

4.4.1 Preliminaries

A POMDP is a stochastic control problem. But it is difficult to solve exactly because the problem has dynamic constraints on distributions over the state space. A POMDP is an extension of an MDP used for planning under partial observations. Parametric and nonparametric algorithms are commonly used for solving finite Markov decision problem using Bellman's principle; these include Monte Carlo methods. However, a belief state in a POMDP is a probability distribution, giving ries to an infinite state space. At best, the states can be approximated by finite-dimensional objects.

An approximately optimal POMDP solution can be found using several approximation approaches. Specifically, we approximate the Q-value $Q(b,d)$, where b is a belief state and d is an action. Some approaches approximate the Q-value by approximating the cost-to-go value $J^*(b)$, where b is a belief state. For example, the parametric approximation method of Q-learning computes the Q-value using $Q(b,d) \approx \hat{Q}(b,d,\theta)$, while the policy rollout method uses $Q(b,d) \approx c(b,d) + E[J^{\pi_{\text{base}}}(b') \,|\, b]$. The hindsight optimization method and the Nominal Belief-State Optimization (NBO) method [1] approximately compute the cost-to-go value $J^*(b)$ for the approximate Q-value $Q(b,d)$ using $J^*(b) \approx E[\min_{(d_j)} \sum_j c(b_j,d_j) \,|\, b]$ and $J^*(b) \approx \min_{(d_j)} \sum_j c(\hat{b}_j,d_j)$ respectively.

The POMDP optimization problem is to find actions over a time horizon H. So, the expected cumulative cost of problem is minimized over the actions sequence $d_0, d_1, ..., d_{H-1}$, can be written by $J_H = \mathrm{E}\left[\sum_{j=0}^{H-1} C(x_j,d_j)\right]$. The process of maintaining the belief state is Markovian in that the next belief state depends only on the current belief state with the current action and observation. Therefore, the action chosen at time j naturally accounts for the history of actions and observations at time $j-1$. The objective function J_H is represented in terms of the belief states by

$$J_H = \mathrm{E}\left[\sum_{j=0}^{H-1} c(b_j, d_j) \,\middle|\, b_0\right], \tag{4.1}$$

where b_0 is the initial belief state. The cost function assocaited with belief states is related to the original cost function C by $c(b_j, d_j) = \int C(x, d_j) b_j(x)\,\mathrm{d}x$. An optimal policy $\pi_j^* : \mathscr{B} \to \mathscr{D}$ specifies the optimal action $d_j = \pi_j^*(b_j)$, where $\mathscr{B}$ is the set of belief states and $\mathscr{D}$ is the set of actions. The minimal value of the objective function (4.1) follows Bellman's principle of optimality:

$$J_H^*(b_0) = \min_u \left\{ c(b_0, d) + \mathrm{E}\left[J_{H-1}^*(b_1) \,\middle|\, b_0, d \right] \right\}, \tag{4.2}$$

where $\mathrm{E}[\cdot|b_0, d]$ is the conditional expectation with the given current belief state b_0 and action d (at time $j = 0$), b_1 is the random next belief state, and $J_{H-1}^*(b_1)$ is the optimal cumulative cost with belief state b_1 over the time horizon $j = 1, 2, \ldots, H-1$ with length $H-1$. The Q-value with action d taken at belief state b_0 is then

$$Q_H(b_0, d) = c(b_0, d) + \mathrm{E}\left[J_{H-1}^*(b_1) \,\middle|\, b_0, d \right]. \tag{4.3}$$

The NBO method computes the approximate Q-value by defining a belief-state sequence $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{H-1}$ and action sequence $d_1, d_2, \ldots, d_{H-1}$ according to the tracking model, the data association, and Gaussian statistics. The belief state sequence of the ith target is defined in terms of the track $(\hat{e}_j^i, \hat{\mathbf{B}}_j^i)$ from the standard Kalman filter without noise: $\hat{b}_j^{tg^i} = \mathscr{N}(\hat{e}_j^i, \hat{\mathbf{B}}_\mathbf{j}^\mathbf{i})$, $\hat{e}_{j+1}^i = \mathbf{F}_j \hat{e}_j^i$, and

$$\hat{\mathbf{B}}_{j+1}^i = \begin{cases} \left[[\hat{\mathbf{B}}_{j+1|j}^i]^{-1} + \mathbf{C}_{j+1}^i \right]^{-1}, & \text{if measurement available,} \\ \hat{\mathbf{B}}_{j+1|j}^i, & \text{otherwise,} \end{cases} \tag{4.4}$$

where

$$\hat{\mathbf{B}}^i_{j+1|j} = \mathbf{F}_j \hat{\mathbf{B}}^i_j \mathbf{F}_j^{\mathrm{T}} + \mathbf{Q}_j,$$
$$\mathbf{C}^i_{j+1} = \mathbf{H}^{\mathrm{T}}_{j+1} \left[\mathbf{R}_{j+1}\left(\hat{e}^i_{j+1}, c_{j+1}\right)\right]^{-1} \mathbf{H}_{j+1},$$

and $c_{j+1} = f_c(c_j, d_j)$. The cost function, the mean-squared error between the tracks and the targets, is given by

$$c(\hat{b}_j, d_j) = \sum_{i=1}^{N_{\mathrm{tar}}} \mathrm{Tr}\, \hat{\mathbf{B}}^i_{j+1}. \tag{4.5}$$

In 4.2, it is generally difficult to calculate the second term. Recall that the belief states are distributions over the state space. Often, belief states must be represented in terms of finite objects to make them practically computable.

POMCP builds on MCTS for partially observable environments. POMCP moves from the initial belief state to a next state by using a sampling method. A tree of history (belief) nodes is created with each node representing a history of the actions and observations. The Q-value $Q(h,d)$ for a node h and action d is calculated by averaging the values from all trajectories starting from node h and taking action d at that node. The UCT method uses the Upper Confidence Bounds (UCB1) for choosing action d:

$$d(ucb1)^* \leftarrow \underset{h \in node}{\arg\max} (Q(h,d) - k\sqrt{\frac{\log N(h)}{N(hd)}}), \tag{4.6}$$

where $N(h)$ is the number of times of a visited node and $N(hd)$ is the number of all simulations with action d chosen at node h.

4.4.2 The P-UAV Algorithm

In the P-UAV algorithm, we calculate the approximate Q-value $Q(h,d)$ using

$$Q(h,d) \approx Q_H(b_0,d) \approx c(b_0,d). \tag{4.7}$$

The P-UAV algorithm is described in detail in Algorithm 5. The process starts by generating a root node of the search tree T. A node h in the search tree represents a history of actions and states and is given by $h = (x, d, c(\hat{b}, d), Q(h,d), N(h), N(hd))$, where x represents the states of the sensors, the targets, and the tracker, d represents the actions, $c(\hat{b}, d)$ is the cost function of all child nodes updated by taking an observation, $Q(h,d)$ is the Q-value used for the UCB calculation, $N(h)$ is the visit count of h, and $N(hd)$ is the visit count of h with action d chosen.

We implement the P-UAV algorithm using parallel computing with Matlab on multicore processors. Parallel computing can reduce computation time and is often exploited in Monte Carlo methods. We use the Parfor-loop in Matlab to execute a series of statements in a loop. The UAV control action is specified using the forward acceleration and the bank angle to control the UAV direction with three types of angles: positive, zero, and negative. The action is represented as $d_j = (f_j, \alpha_j)$, where $f_j \in [-3, 3]$, $\alpha_j = \{[-\alpha, 0), 0, (0, \alpha]\}$, and $\alpha = \frac{\pi}{12}$.The tracker state $(e_j, \mathbf{B}_j)$ is generated by a Kalman filter and enables us to obtain the cost value $c(\hat{b}_j, d_j)$, which is key to P-UAV. After the cost function $c(\hat{b}_j, d_j)$ is calculated, the sum of the cost function $Q(h,d)$ is updated by $Q(h,d)' \leftarrow c(\hat{b}_j, d_j)$, and $Q(h,d) \leftarrow Q(h,d) + \frac{Q(h,d)' - Q(h,d)}{N(hd)}$.

In the selection process, the first child node is chosen for the expansion process using a best-first rule. A sample is then drawn from the belief state. Using the sample, the belief and cost function are updated for each observation along the path from the root to the chosen node. In the expansion process, the P-UAV algorithm uses the HeuristicSamplingNode function for generating the new tree nodes added to a chosen node by using a heuristic technique for UAV control. The simulation process selects actions randomly according to the HeuristicSamplingNode function. The maximum value of action is chosen for calculating the cost function $c(\hat{b}, d^*)$.

The final cost value is stored at the new node, and the cost value is then propagated from the expanded node to the root node. The maximun value of the sum of cost values $Q_{max,z}$ is calculated for each parallel loop. When finishing the process, the maximun value over all actions is evaluated again.

Algorithm 5 The P-UAV algorithm

procedure P-UAV($Depth$, x)
  $h \leftarrow x = (c, \hat{b}(tg), e, \mathbf{B})$ /* the history node h contains
  $h^{'}, d \leftarrow$ HeuristicSamplingGenNewnode(h), $dep \leftarrow 1$
  for $y \leftarrow 1, n$ **do** /*Matlab Parfor loop parallel
    for $z \leftarrow 1, m$ **do**
      while $dep \neq Depth$ **do**
        $d^{*} \leftarrow$ SelectionProcess($h^{'}, d$)
        if $h^{'}$ is not member of T **then**
          $h^{''} \leftarrow$ HeuristicSamplingNode($h^{'}(d^{*})$)
        end if
        $h^{'} \leftarrow h^{''}$, dep=dep+1
      end while
      $c(b,d) \leftarrow$ Simulation($h_{depth+1}$), $dep \leftarrow 1$
      $Q(h,d) \leftarrow$ UpdateCost($c(\hat{b},d)_{h(0,depth)}$)
    end for
    $Q_{max,m} \leftarrow \max\{Q(h,d_1), Q(h,d_2), Q(h,d_3)\}$
  end for
  $d_{maximum} \leftarrow \max\{Q_{max,m,1}(h,d_j), .., Q_{max,m,n}(h,d_j)\}$
end procedure
procedure HEURISTICSAMPLINGNODE(h)
  $d = (f, \alpha)$, $f \in [-3,3]$, $\alpha = \{[-\alpha, 0), 0, (0, \alpha],\}$,
  $\alpha = \frac{\pi}{12}, f = random[-3,3], \alpha(1) = random[-\alpha, 0)$
  $\alpha(2) = 0, \alpha(3) = random(0, \alpha]$
  $(e, \mathbf{B}) \leftarrow$ Kalmanfunction($h'(d)$) /* Create tracker
  $c(\hat{b},d) = \sum_{i=1}^{N_{\text{tar}}} \text{Tr}\,\hat{\mathbf{B}}$, $h'(d) \leftarrow c(\hat{b},d)$
end procedure
procedure UPDATECOST($c(\hat{b},d)_{h(0,depth)}$)
  for $h_0 \quad to \quad h_{depth}$ **do**
    $Q(h,d)' \leftarrow c(\hat{b},d)$
    $N(h) \leftarrow N(h) + 1$, $N(hd) \leftarrow N(hd) + 1$
    $Q(h,d) \leftarrow Q(h,d) + \frac{Q(h,d)' - Q(h,d)}{N(hd)}$
  end for
end procedure
procedure SELECTIONPROCESS(h,d)
  $d(h') \leftarrow \arg\max_{h \in node}(\mathbf{1}/\mathbf{Q}(h,d) + \sqrt{\frac{\log N(h)}{N(hd)}})$
end procedure
procedure SIMULATION(h)
  while $loop < constant$ **do**
    $h^{'}, d \leftarrow$ HeuristicSamplingNewnode(h)
    $dl \leftarrow$ SelectionProcess($h^{'}, d$)
    if $dl > d^{*}$ **then** $d^{*} \leftarrow dl$
    end if; $h \leftarrow h^{'}$
  end while; **return** $c(\hat{b}, d^{*})$;
end procedure

4.5 Simulation Results

We implemented the P-UAV algorithm in Matlab and ran it on a machine with an Intel I7 CPU (2.2GHz, 6 cores total), 2GB RAM, and 6 parallel workers. Table 4.1 summarizes our results comparing P-UAV with NBO in terms of the track error for track1 (t1) and track2 (t2). P-UAV incorporates an exploration-exploitation tradeoff through the use of UCT. For t1, P-UAV is superior to NBO, but for t2, NBO is superior. Figures 4.1 and 4.2 show the errors over time for P-UAV and NBO. These figures clearly demonstrate that the NBO errors often exceed those of P-UAV, although their average track error values (in Table 4.1) are similar. Figure 4.3 illustrates some possible trajectories of the UAVs resulting from P-UAV and NBO for the two tracks t1 and t2.

Table 4.1: P-UAV vs NBO track error (meter) results

NBO-t1-error	NBO-t2-error	P-UAV–t1-error	P-UAV-t2-error
2.7130	1.8741	1.9599	1.8421
2.4406	2.3560	2.3139	2.7125
1.7391	2.7433	2.4025	3.7572
2.2975 (Avg)	2.3244 (Avg)	2.2254 (Avg)	2.7706(Avg)

4.6 Conclusions

In this work, we introduced the P-UAV algorithm for UAV-based target tracking. With an appropriate implementation, the method can be used in real time for realistic practical scenarios. The method uses a Kalman filter to update the belief state and incorporates a heuristic approach to select UAV control actions. P-UAV combines a best-first selection scheme and an adaptive Monte Carlo sampling technique using the UCB approach. It incorporates a way to manage the exploration-exploitation tradeoff. Parallel processing can enhance the performance of P-UAV by increasing the number of CPU cores. P-UAV potentially outperforms NBO, especially with sufficient number of CPU cores, In our future work, we plan to explore P-UAV in more complex dynamic environments for online processing and robustness testing.

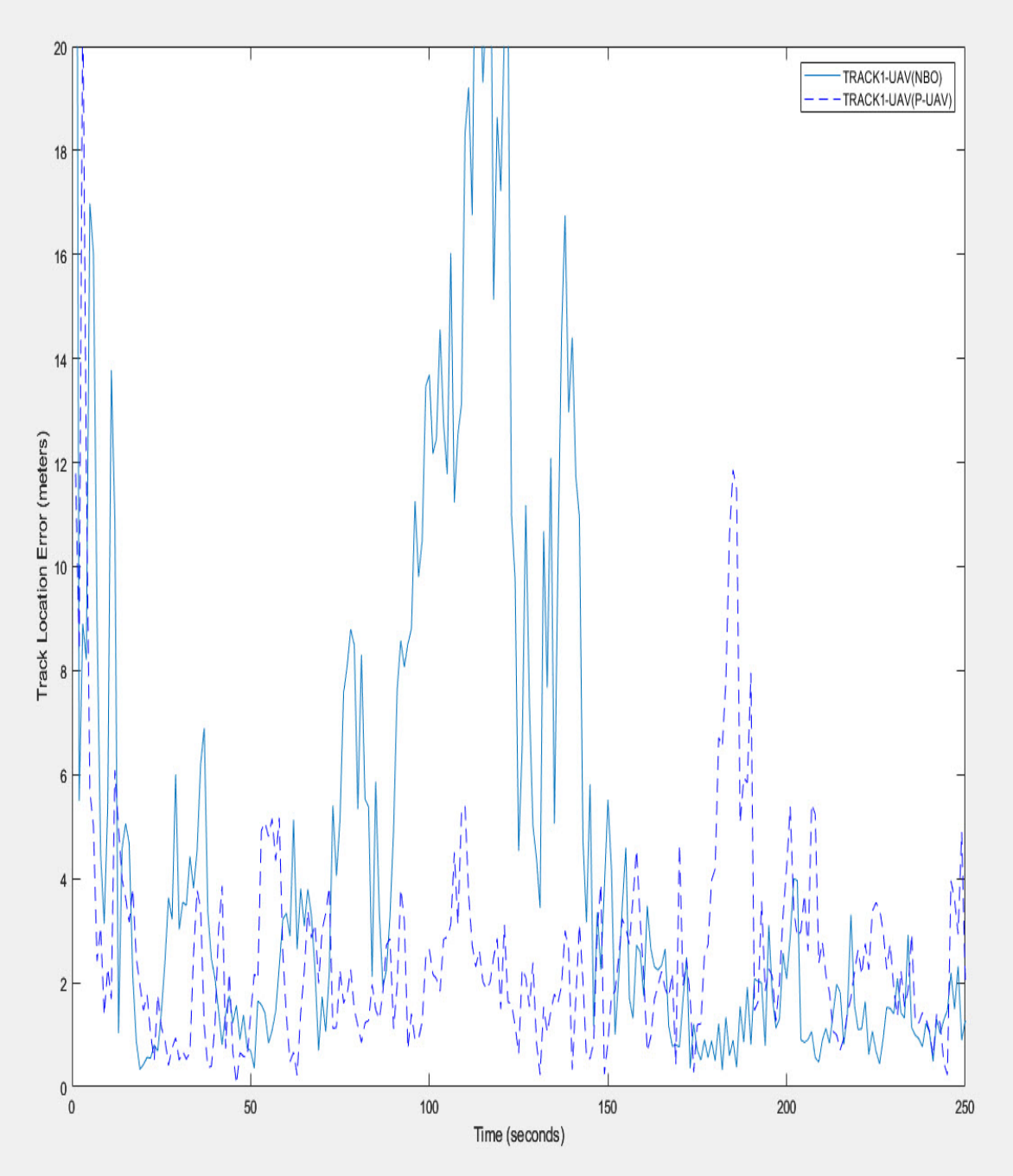

Figure 4.1: P-UAV vs NBO track 1 error

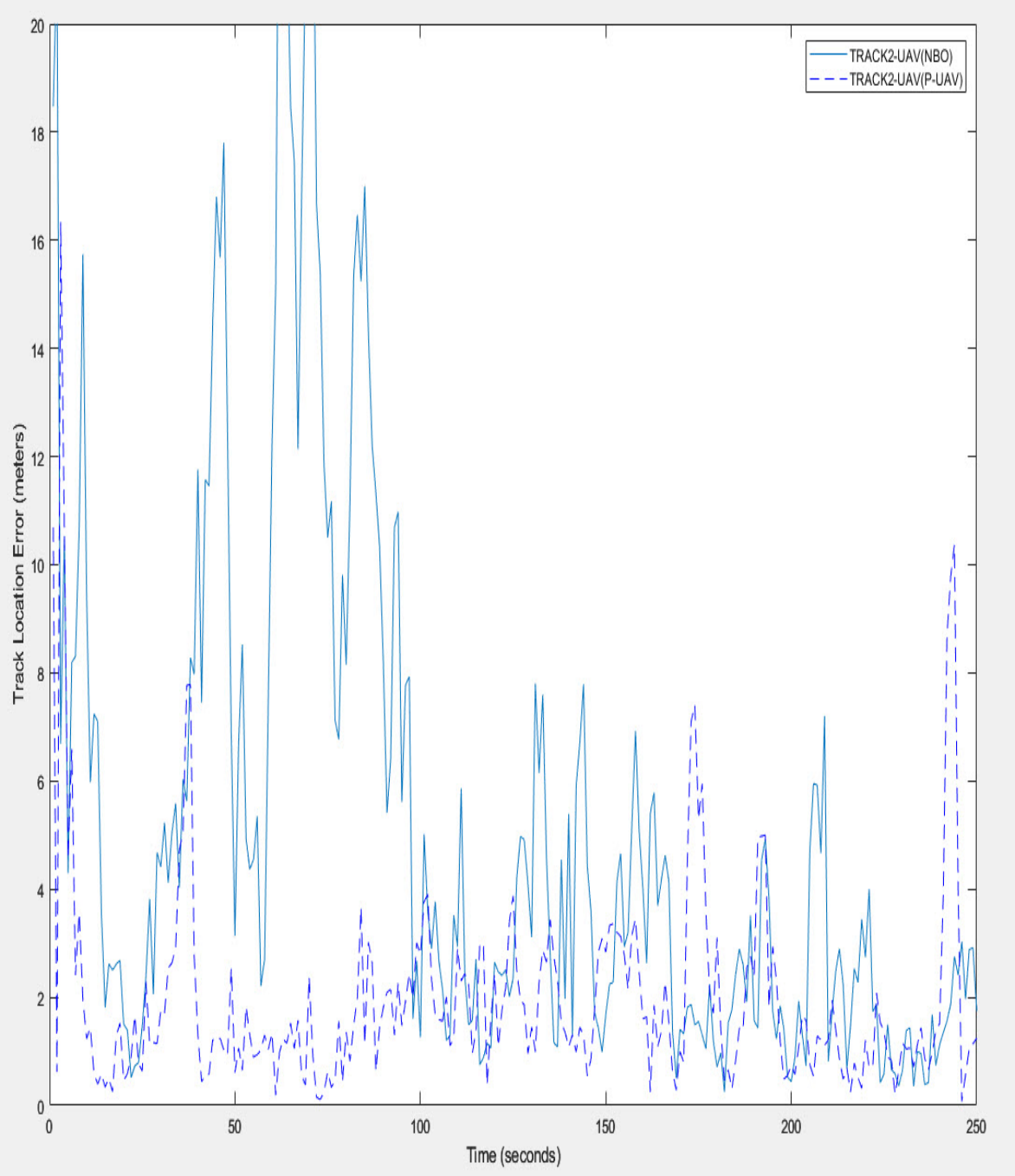

Figure 4.2: P-UAV vs NBO track 2 error

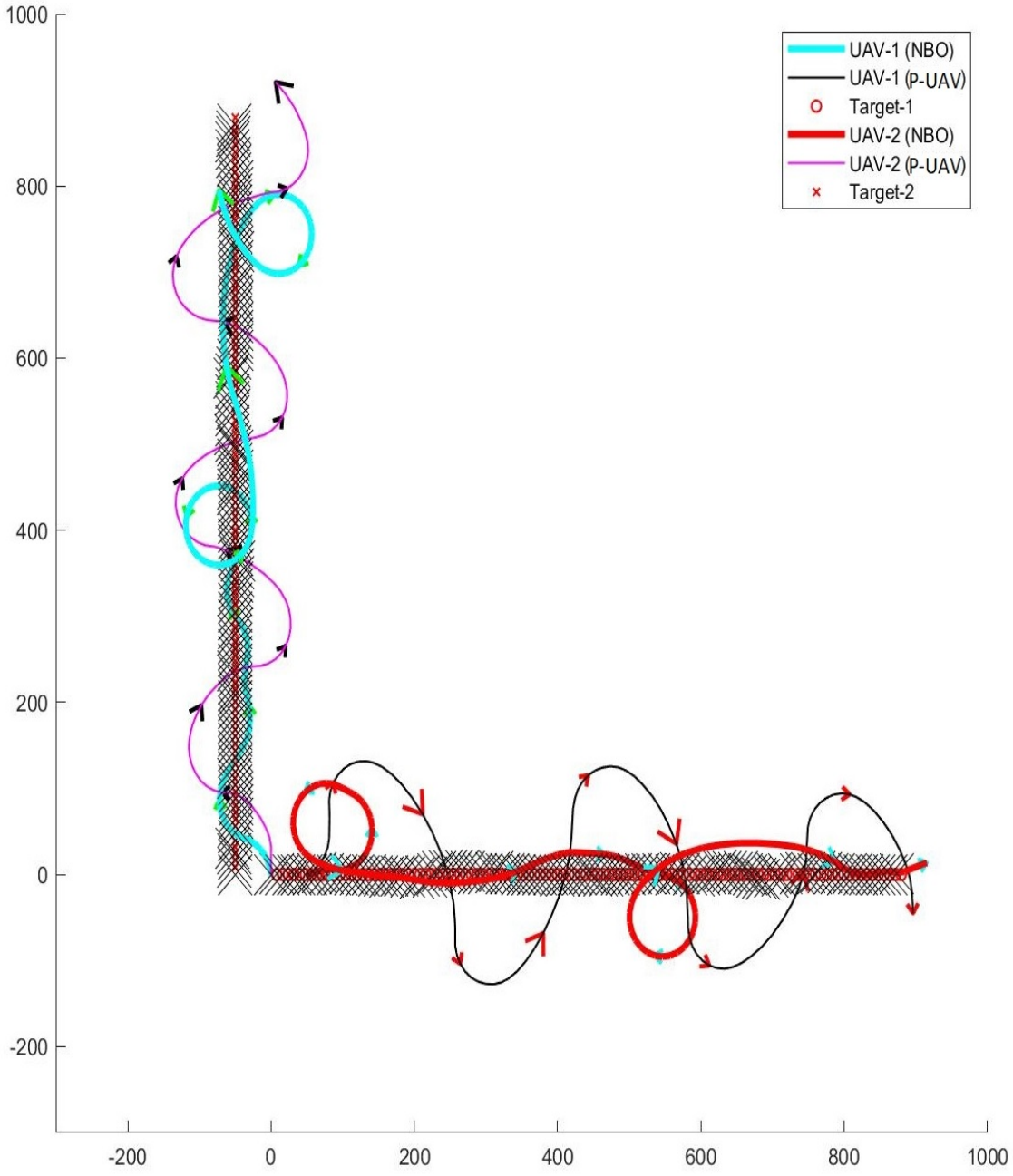

Figure 4.3: P-UAV vs NBO track 1, and track 2

Chapter 5

An Autonomous UAV Path-Planning Algorithm for Mobile Target Tracking in Complex Environments

5.1 Overview

Unmanned Aerial Vehicles (UAVs) are now part of many different domains, including civilian and defense industries. Recent advances in UAV technology have mainly emerged from artificial intelligence (AI). UAV path planning is a major challenge, typically NP-hard, particularly in complex environments with real-time obstacle avoidance, exposing opportunities for AI in autonomous UAV systems. In this paper, we propose an autonomous path-planning algorithm to guide UAVs for tracking mobile ground targets in complex environments. We build on the framework of partially observable Markov decision process (POMDP) by integrating a non-myopic method with long-term decision making. Our algorithm is based on partially observable Monte Carlo planning (POMCP) using Monte Carlo tree search (MCTS), originally developed for Markov decision process (MDP) problems. Combining a heuristic method with parallel computing, the algorithm is applicable to real-time applications. Our experimental results demonstrate that our autonomous path-planning algorithm is able to achieve near-optimal performance in large complex environments by appropriately tuning the exploration-exploitation tradeoff.

Mathematics Subject Classification: Mathematical modeling, Probability and statistics

Keywords: Unmanned aerial vehicles (UAVs), Monte Carlo tree search (MCTS), Markov decision processes (MDP), Online path planning, Partially observable Markov decision processes (POMDP), Artificial intelligence (AI), NP-hard

5.2 Introduction

Path planning is a critical component of guidance, navigation, and control of unmanned aerial vehicles (UAVs), also known as drones. The objective of path planning is to find an optimal trajectory from an initial point to a terminal point and avoid a variety of obstacles in real-world environments. UAVs operate in constant flight, which differs significantly in some aspects from traditional ground robot path planning. A conventional UAV is controlled by both forward acceleration and bank angle subject to constraints in constructing paths with a relatively large turning radius and a minimum velocity to maintain UAV paths in the air.

The UAV path-planning problem is challenging in complex scenarios with large numbers of obstacles and multiple moving targets. Typical formulations of the problem are NP-hard, with a very large set of feasible candidate solutions. It is computationally demanding to find an optimal solution subject to various constraints. Modern UAVs are often equipped with sensors and onboard computing capabilities to observe the environment and make decisions autonomously. However, it is generally computationally expensive to obtain complete sensor measurements and calculate paths continuously while tracking mobile ground targets in real-time in response to dynamic conditions.

We formulate and solve the path-planning problem here by employing partially observable Markov decision processes (POMDPs). A POMDP is generally used to model decision problems under uncertainty. Applications of POMDPs appear in several fields such as machine vision, search and tracking, robot navigation, autonomous control, human-robot interaction, and machine maintenance. A POMDP is generalization of a Markov decision process (MDP), employing a MDP to model system dynamics and a hidden Markov model for the observations of unobservable states. It is generally intractable to compute an optimal solution in a POMDP problem [6]. Normally, approximations are needed in obtaining POMDP solutions, such as approximate dynamic programming, heuristics, hindsight optimization, partially observable Monte-Carlo planning (POMCP) [7], and nominal belief-state optimization (NBO) [8]. Specifically, the NBO technique is designed for

UAV path planning [8], [9], [10] using a POMDP framework. In other related work, [11] formulates an object search problem as a POMDP and applies a DESPOT system [12] for planning.

A POMDP can be converted to a conventional MDP in which the state space is the posterior distribution (called the *belief state*) of the underlying unobservable state computed from the observations history. The observation law (conditional distribution) depends on the present state and control action. The set of belief states are continuous and extremely large even if discretized. Monte Carlo tree search (MCTS) is used in calculating approximate MDP solutions, based on sampling [13]. POMCP [7] is a variant of MCTS that employs particle filters for belief-state representation in the search tree for large POMDPs [50] – [51].

Monte Carlo tree search (MCTS) was employed in AlphaGo [13] in 2006. It seeks near-optimal solutions of decision problems by combining a best-first strategy with an adaptive sampling method using the upper-confidence-bound (UCB). Particularly, MCTS applies the method of upper confidence bound for trees (UCT) to achieve a tradeoff for exploration and exploitation in order to seek near-optimal solutions in large state-spaces. The MCTS algorithm combines the generality of Monte Carlo sampling with the precision of tree search. MCTS is particularly suited to problems with expensive function evaluations. MCTS has been widely used in a variety of areas such as planning, scheduling, image processing, and gaming. In the UAV area, the applications of MCTS range from data gathering to image recognition for online decision making.

POMCP [7] involves applying MCTS to POMDPs, presented by Silver and Veness in 2010. The method accounts for partial observations by connecting actions with a history of observations to a node in the search tree. Each node of the search tree represents a state of the problem in the terms of a belief state, containing a history with its a statistical value. The history consists of actions and observations $h_t = \{a_1, o_1, ..., a_t, o_t\}$, and the statistical value of the history is calculated using a Monte-Carlo technique for states considered from the present belief state. POMCPs for POMDPs have been used for drones and robots. For example, [14] used an adapted POMCP technique with the traditional UCB algorithm for an autonomous urban UAV navigation problem. A target object search problem was solved using POMCP in [15].

Our present work develops an autonomous path-planning algorithm based on a method called *P-UAV*. We developed P-UAV in [4] for multitarget tracking problems. P-UAV was designed using a POMCP approach to sensibly navigate UAVs for tracking mobile ground targets in complex environments in real time. Here, we use a Kalman filter technique instead of a particle filter to update the belief-state in the search tree. Our current modified P-UAV algorithm employs a heuristic technique with the UAV acceleration and bank angle under constraints to produce an efficient non-myopic path planning method. In real-time applications, we integrate parallel computing into P-UAV to improve its computational performance.

In Section 5.3, we introduce our POMDP framework for the path-planning problem. Section 5.4 presents our POMDP methodology and its implementation, our P-UAV algorithm with obstacle collision avoidance. In Section 5.5, we show experimental results to evaluate of the performance for our algorithm, comparing it with the NBO algorithm. We conclude in Section 5.6.

5.3 Problem Definition

The UAV path-planning problem for mobile ground targets tracking is defined by the numerous features of the system.

The targets move on the ground in two dimensions, and UAVs fly at a constant altitude over the ground with a simple UAV motion model. Each UAV is controlled by applying the control action of forward acceleration and bank angle using position coordinates in two dimensions. A camera is mounted on the UAV with a machine-vision system for visual measurements, assumed to have random errors without false alarms and missed detections. The measurement error covariance of a target depends on the relative position of UAVs and targets. Obsctacles are modelled as occlusions with a variety of shapes such as rectangles and circles, which block the UAV paths in the operation area, as shown in Figures 5.1-5.2. UAV velocities are constrained within a prespecified interval, with bounded lateral acceleration in the plane.

The goal of P-UAV is to minimize the mean-squared error between tracks and targets and generate control actions for UAVs to keep track of the ground targets. In centralized control,

the P-UAV algorithm collects data from all the sensors and fuses them in the tracker for updating tracks. The algorithm then calculates UAV control action commands to supervise the incremental UAV trajectories. We assume a discrete-time model with a suitable rate.

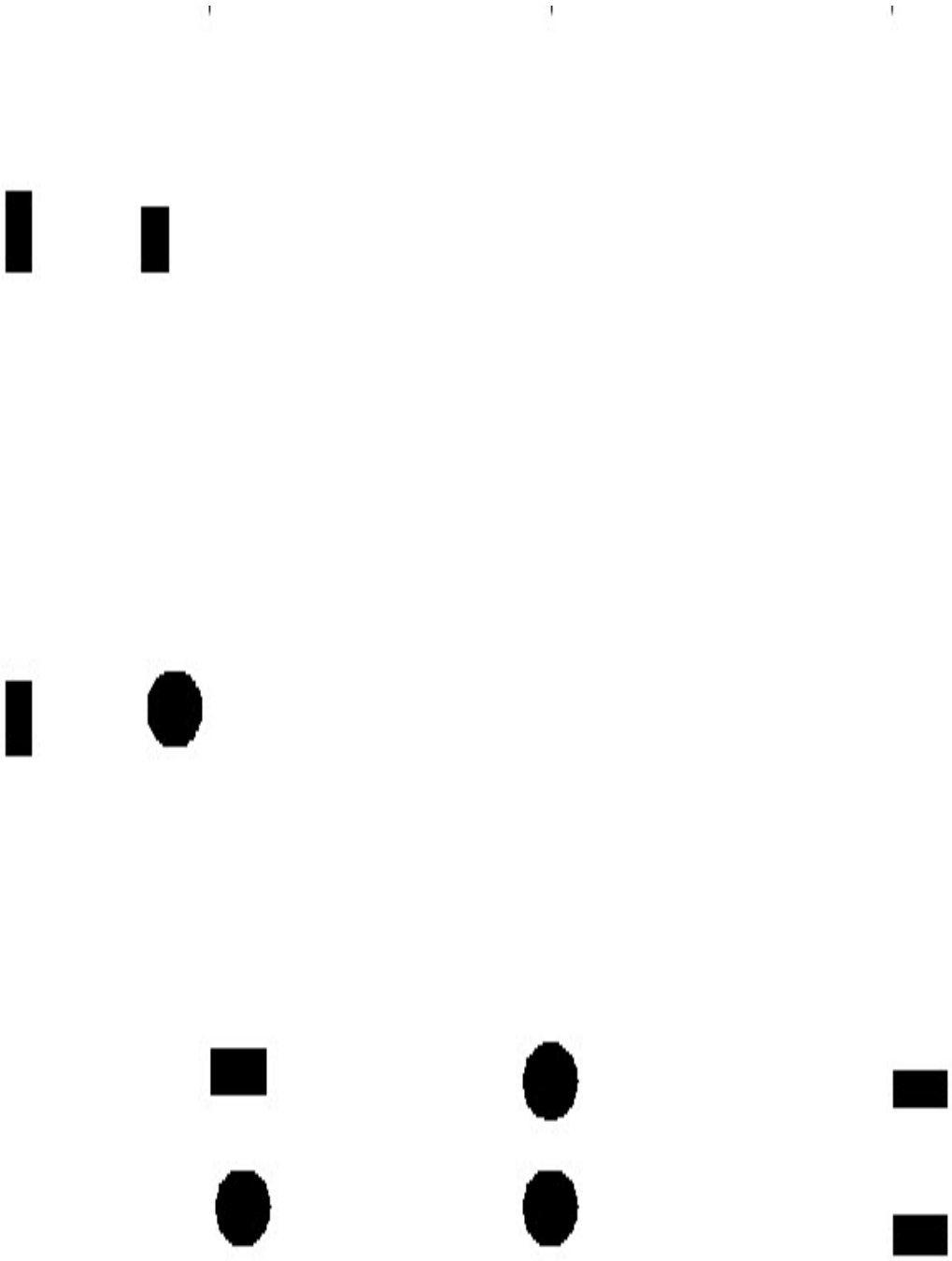

Figure 5.1: Obstacles in surveillance area

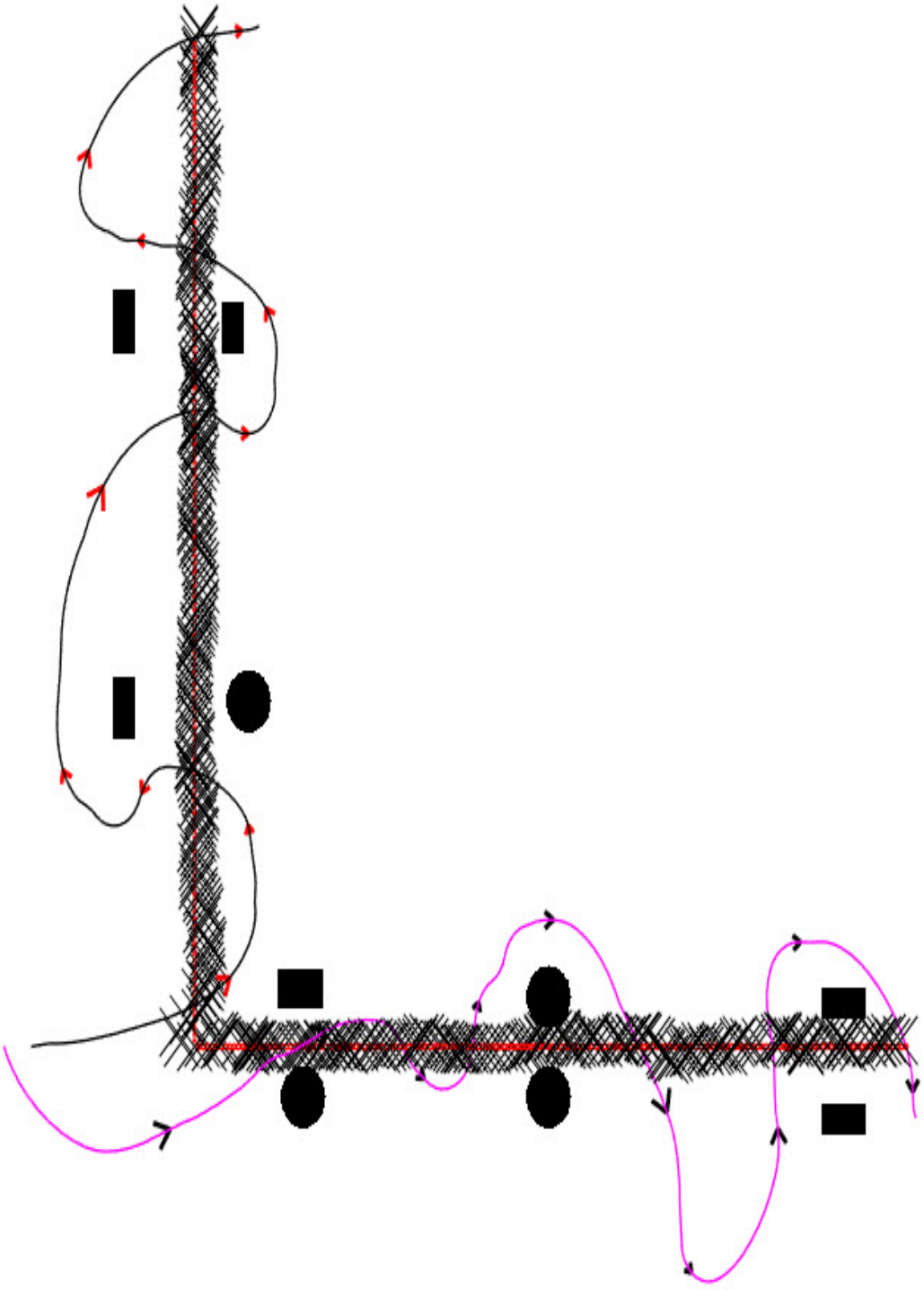

Figure 5.2: UAVs following targets through obstacles

5.3.1 Partially Observable Markov Decision Process Approach

The original POMDP formulation in [8] is used for defining the UAV path-planning problem, as descrbed below.

States. The POMDP state used in the P-UAV algorithm includes three segments according to the UAV sensors, the targets, and the tracker. The state at time j is given by $\mathbf{x}_j = (c_j, tg_j, e_j, \mathbf{B}_j)$, where c_j denotes the sensor state, tg_j denotes the target state, and $(e_j, \mathbf{B}_j)$ denotes the tracker state. The sensor state includes the UAV positions and velocities. The target state includes the target positions, velocities, and accelerations. The tracker utilizes a simple Kalman filter, and therefore the tracker state is defined as the Kalman-filter state, where e_j indicates the posterior mean vector and $\mathbf{B}_j$ indicates the posterior covariance matrix.

Actions. The P-UAV algorithm generates each control action for guiding the UAVs as forward acceleration and bank angle. The action at time j is given by $d_j = (f_j, \alpha_j)$, where f_j is the acceleration vector and α_j is the bank-angle vector.

Observation and Observation Law. Our P-UAV algorithm assumes the tracker and the sensor states to be fully observable, but the target states are partially observable. The observation of the position of the ith target at time j is $z_j^i = H_j tg_j^i + w_j^i$, where matrix H_j applies to every target, and w_j^i represents additive zero-mean Gaussian noise.

State-Transition Law. The state-transition law in the P-UAV algorithm is divided into separate laws for sensors, targets, and trackers. The transition law of each sensor is represented as $c_{j+1} = f_c(c_j, d_j)$, where a function f_c represents a simple kinematic function, defined later. The control action at time j for target i is $d_j^i = (f_j^i, \alpha_j^i)$, where the acceleration is represented as f_j^i and the bank angle is represented as α_j^i. The joint probabilistic data association (JPDA) approach is utilized in the Kalman filter for the tracker-state evaluation.

The UAV direction is forced by a heading angle $\omega_{j+1}^i = \omega_j^i + (GT\tan(\alpha_j^i)/V_j^i)$, where V_j^i indicates the speed, G indicates the gravitational constant, and T indicates the time-step duration. The UAV speed is updated by $V_{j+1}^i = [V_j^i + f_j^i T]_{V_{\min}}^{V_{\max}}$, where $[v]_{V_{\min}}^{V_{\max}} = \max\{V_{\min}, \min(V_{\max}, v)\}$, $V_{\max}$ in-

dicates the UAV maximum speed, and $V_{\min}$ indicates the UAV minimum speed. The UAV position is updated by $m^i_{j+1} = m^i_j + V^i_j T\cos(\omega^i_j)$ and $n^i_{j+1} = n^i_j + V^i_j T\sin(\omega^i_j)$.

The updated target state is defined by $tg_{j+1} = f_t(tg_j) + v_j$, where the target dynamics is based on linear motion with zero-mean noise: $tg^i_{j+1} = \mathbf{F}_j tg^i_j + v^i_j$, $v^i_j \sim \mathcal{N}\left(0, \mathbf{Q}_j\right)$, $i \in \{1,\ldots,N_{\text{tar}}\}$, where $\mathbf{F}_j$ is the speed transition matrix.

Cost Function. The P-UAV algorithm aims to minimize the mean-squared error of the tracks and the targets according to $C(\mathbf{x}_j, d_j) = \mathrm{E}_{v_j, w_{j+1}}\left[||tg_{j+1} - e_{j+1}||^2 \,\middle|\, \mathbf{x}_j, d_j\right]$.

Belief State. The P-UAV algorithm employs the belief state $b_j = (b^c_j, b^{tg}_j, b^e_j, b^B_j)$. The belief state of the fully observable sensor and tracker states is given by $b^c_j = \delta(c - c_j)$, $b^e_j = \delta(e - e_j)$, and $b^{\mathbf{B}}_j = \delta(\mathbf{B} - \mathbf{B}_j)$ sequentially. The belief state of the partially observable target state is computed from the normal distribution $b^{tg^i}_j = \mathcal{N}\left(e^i_j, \mathbf{B}^i_j\right)$.

5.3.2 Monte Carlo Tree Search

The MCTS process is basically an approach to solve a MDP. MCTS can be applied to a POMDP by using the belief state as the state of a MDP, called the *belief MDP*. A decision tree in a MCTS process is utilized in which states correspond to nodes and state transitions correspond to directed edges.

The two policies used in the MCTS approach are the *tree* policy and the *default* policy [13]. Finding an optimal solution involves an exploration-exploitation tradeoff in the tree policy. *Exploration* involves trying out new actions to learn something new about their impact, whereas *exploitation* involves actions to optimize the rewards. The tradeoff corresponds to finding the right balance between trying new things and enjoying the rewards of actions taken. We use the upper confidence bound for trees (UCT) method to achieve this tradeoff. In the simulation process, the default policy is utilized to compute approximately the values of the cost function at the leaf nodes.

There are four stages in the MCTS method, which begins by creating the root node of the search tree, the current state, and actions leading to child nodes. After that, MCTS uses the following four stages to find optimal solutions.

Selection stage: This stage begins at the root node of the search tree and chooses a child node using the upper confidence bound for trees (UCT) method until arriving at a leaf node.

Expansion stage: If a leaf node chosen in the selection stage is not a terminal node, the expansion stage is used to generate new child nodes by using a random sampling method. In particular, the MCTS method applies the expansion stage according to the depth of a node.

Simulation stage: From the child node in the expansion stage, the simulation stage evaluates the value of the node by applying a random policy until a terminal node is reached.

Backpropagation stage: The backpropagation stage updates the numerical values of the node evaluated by the previous simulation stage along the path back to the root node. After this stage is finished, the MCTS method returns to the selection stage.

5.4 POMDP Methodology

5.4.1 Preliminaries

POMDPs are used for planning problems with partial observations. A POMDP is a stochastic optimal control problem and is hard to solve exactly. Formulated as an equivalent MDP, the states in a POMDP (belief state), are distributions, leading to an uncountably infinite state space. The usual approach is to apply certain approximations. First, the states can be approximated in terms of finite-dimensional objects. Second, we can directly approximate the Q-value $Q(b,d)$, where b represents a belief state and d represents an action. Alternatively, we can indirectly approximate the Q-value by estimating the cost-to-go value $J^*(b)$, where b represents a belief state.

For example, in the Nominal Belief-State Optimization (NBO) method [8], the Q-value $Q(b,d)$ is approximated by approximating the cost-to-go value $J^*(b)$ as $J^*(b) \approx \min_{(d_j)} \sum_j cost(\hat{b}_j, d_j)$. Building on this, in [15], the Q-value with action d taken at belief state b_0 is computed according to

$$Q_H(b_0,d) = cost(b_0,d) + \mathrm{E}\left[J^*_{H-1}(b_1) \,\middle|\, b_0,d\right]. \tag{5.1}$$

The approximate Q-value in the NBO technique is calculated from a belief-state sequence $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{H-1}$ and action sequence $d_1, d_2, \ldots, d_{H-1}$ corresponding to the tracking model, Gaussian distribution, and the data association method.

In the tracking problem, the belief state sequence of the ith target is approximated in terms of the track $(\hat{e}^i_j, \hat{\mathbf{B}}^i_j)$ from the Kalman filter equations without noise: $\hat{b}^{tg^i}_j = \mathcal{N}(\hat{e}^i_j, \hat{\mathbf{B}}^{\mathbf{i}}_{\mathbf{j}})$, $\hat{e}^i_{j+1} = \mathbf{F}_j \hat{e}^i_j$, and

$$\hat{\mathbf{B}}^i_{j+1} = \begin{cases} \left[[\hat{\mathbf{B}}^i_{j+1|j}]^{-1} + \mathbf{C}^i_{j+1} \right]^{-1}, & \text{the observation available,} \\ \hat{\mathbf{B}}^i_{j+1|j}, & \text{in other ways,} \end{cases} \tag{5.2}$$

where

$$\begin{aligned} \hat{\mathbf{B}}^i_{j+1|j} &= \mathbf{F}_j \hat{\mathbf{B}}^i_j \mathbf{F}^{\mathrm{T}}_j + \mathbf{Q}_j, \\ \mathbf{C}^i_{j+1} &= \mathbf{H}^{\mathrm{T}}_{j+1} \left[\mathbf{R}_{j+1} \left(\hat{e}^i_{j+1}, c_{j+1} \right) \right]^{-1} \mathbf{H}_{j+1}, \end{aligned}$$

and $c_{j+1} = f_c(c_j, d_j)$. The cost function is given by the mean-squared error between the tracks and the targets, calculated using

$$cost(\hat{b}_j, d_j) = \sum_{i=1}^{N_{\text{tar}}} \mathrm{Tr}\, \hat{\mathbf{B}}^i_{j+1}. \tag{5.3}$$

In equation (5.1), the second term is generally difficult to calculate because the belief states are probability distributions over the state space. Accordingly, belief states are approximated as finite objects.

5.4.2 The P-UAV Algorithm with Obstacle Collision Avoidance

To incorporate obstacle collision avoidance, we assume that each UAV flies at an altitude that is lower than the height of the obstacles while following the targets that move on the ground in the presence of these obstacles. We use a penalty term in the cost function in [7], where the value of this term increases as a UAV flies closer to an obstacle within a distance of 100 meters, as shown

in detail in Algorithm 6. The minimum distance between the nth UAV, UAV_n, and any obstacle is calculated in the first step in line 4. The penalty is applied only at distance of less than 100 meters. If the distance to an obstacle is smaller than 100 meters, the square difference of this distance from 100 meters is multiplied by a constant and added to the distance to obtain the penalty value (in line 6). The cost function is then calculated as

$$cost(\hat{b}_j, d_j) = \sum_{i=1}^{N_{\text{tar}}} \text{Tr}\hat{\mathbf{B}}^i_{j+1} + \sum_{k=1}^{N_{\text{sen}}} Obs^k_{j+1}, \tag{5.4}$$

where $\sum_{k=1}^{N_{\text{sen}}} Obs^k_{j+1}$ is the penalty term described above.

Algorithm 6 Penalty calculation for obstacle collision avoidance

```
1:  procedure PENALTY
2:      Obs ← 0
3:      for y ← 1,n do/*n indicates the number of UAVs
4:          Obs ← min (distance of UAV_n and all obstacles)
5:          if Obs < 100 then
6:              Obs ← Obs + Constant * (100 − Obs)^2
7:          end if
8:      end for
9:      return Obs
10: end procedure
```

POMCP is an extension of MCTS to partially observable environments, using a sampling method to move from the initial belief state to a next belief state in an online fashion. Each node of the search tree represents a belief state, which contains a history of the actions and observations. In node h, the Q-value $Q(h,d)$ is calculated by averaging over the a set of simulated trajectories that start with taking action d at node h. We then select an action $d(ucb1)^*$ using the Upper Confidence Bounds (UCB1) as follows:

$$d(ucb1)^* \leftarrow \underset{h \in node}{\arg\max}(Q(h,d) + k\sqrt{\frac{\log Nb(h)}{Nb(hd)}}), \tag{5.5}$$

where $Nb(hd)$ represents the number of simulations at node hb with action d selected, and $Nb(h)$ represents the number of visits to node h.

The P-UAV algorithm computes the approximate Q-value $Q(h,d)$ by employing

$$Q(h,d) \approx cost(b,d). \tag{5.6}$$

The P-UAV algorithm is shown in detail in Algorithms 2 and 3. We implemented the method by applying a parallel technique with Matlab on a multicore processor. The algorithm typically begins by processing the root node $h = (x,d,cost(\hat{b},d),Q(h,d),Nb(h),Nb(hd))$ of the search tree, where x indicates the states of the sensors, the targets, and the tracker, d indicates the actions, $cost(\hat{b},d)$ indicates the cost function of all child nodes, $Q(h,d)$ indicates the Q-value using the UCT method, $Nb(h)$ indicates the visit count of h, and $Nb(hd)$ indicates the visit count of h with action d.

Parallel computing techniques can improve computation times and are often exploited in Monte Carlo approaches. We use the Parfor-loop in Matlab to process sequential statements in the algorithm's main loop. We control the UAV direction in terms of three groups of angles (positive, zero, and negative). This strategy helps to reduce the computation time. The control action is denoted as $d_j = (f_j, \alpha_j)$, where $f_j \in [-3,3]$, $\alpha_j = \{[-\alpha,0),0,(0,\alpha]\}$, and $\alpha = \frac{\pi}{12}$.The tracker state $(e_j, \mathbf{B}_j)$ is computed by a Kalman filter, giving the cost function $cost(\hat{b}_j, d_j)$ by incorporating the penalty $\sum_{k=1}^{N_{\text{sen}}} Obs_{j+1}^{k}$ associated with obstacle avoidance, key to our approach. We then calculate $Q(h,d)' \leftarrow cost(\hat{b}_j, d_j)$. The Q-value $Q(h,d)$ is updated using $Q(h,d) \leftarrow Q(h,d) + \frac{Q(h,d)'-Q(h,d)}{Nb(hd)}$.

Algorithm 3 includes four procedures. The SelectPhase procedure implements the selection phase by selecting a child node for the expansion phase. The HeuSampGennode procedure applies a best-first strategy according to the UCT equation. The UpdateC procedure computes the Q-value. The SimulPhase procedure implements the simulation phase by randomly choosing control actions in accordance with the HeuSampGennode procedure. In the loop iterations, we use parallel computing to compute $Q_{max,o}$ in each loop and finally find the highest value over all loops.

Algorithm 7 P-UAV algorithm with obstacle avoidance

procedure MAIN P-UAV ALGORITHM($Deepness(tree)$, x)
  $h \leftarrow x$, d, $cost(\hat{b},d)$, $Q(h,d)$, and statistical values
  $h^{'}, d \leftarrow$ HeuSampGennode(h), $Pdep \leftarrow 1$
  for $L \leftarrow 1, p$ **do** /*Parallel loops
    for $W \leftarrow 1, o$ **do**
      while $Pdep \neq Deepness(tree)$ **do**
        $d^{*} \leftarrow$ SelectPhase($h^{'}, d$)
        if $h^{'} \neq$ a child node of T **then**
          $h^{''} \leftarrow$ HeuSampGennode($h^{'}(d^{*})$)
        end if
        $h^{'} \leftarrow h^{''}$, Pdep=Pdep+1
      end while
      $cost(b,d) \leftarrow$ SimulPhase(h_{Pdep+1}), $Pdep \leftarrow 1$
      $Q(h,d) \leftarrow$ UpdateC($cost(\hat{b},d)_{h(0,Pdep)}$)
    end for
    $Q_{max,o} \leftarrow \max\{Q(h,d_1), Q(h,d_2), Q(h,d_3)\}$
  end for
  $d_{max} \leftarrow \max\{Q_{max,o,1}(h,d_j), .., Q_{max,o,p}(h,d_j)\}$
  return d_{max};
end procedure

5.5 Experimental Results

The P-UAV algorithm with obstacle collision avoidance was implemented in Matlab and ran on a computer with an Intel I7 CPU (2.2GHz, six cores total) and 2GB RAM. The track errors for P-UAV and NBO (for tracks I and II) are shown in Table I, which illustrates that P-UAV outperforms NBO. The errors over time of P-UAV and NBO are shown in Figures 5.3-5.5 which shows that the error in NBO is generally worse than in P-UAV.

Table 5.1: Track errors for P-UAV and NBO (meters)

Track I-error (NBO)	Track II-error (NBO)	Track I-error (P-UAV)	Track II-error (P-UAV)
3.174	2.791	2.881	2.572
3.378	2.830	2.925	2.437
Average= 3.276	2.811	2.903	2.505

Algorithm 8 Subroutines used in P-UAV with obstacle avoidance

procedure HEUSAMPGENNODE(h)
  $d = (f, \alpha)$, $f \in [-3,3]$, $\alpha = \{[-\alpha_q, 0), 0, (0, \alpha_q], \}$,
  $\alpha_q = \frac{\pi}{12}, f = rand[-3,3], \alpha(1) = rand[-\alpha, 0)$,
  $\alpha(2) = 0, \alpha(3) = rand(0, \alpha]$
  $h'(d) \leftarrow$ generate new history nodes to node h
  $(e, \mathbf{B}) \leftarrow$ Kalmanfilter($h'(d)$) /*generate tracker
  $cost(\hat{b}, d) = \sum_{i=1}^{N_{\text{tar}}} \text{Tr}\hat{\mathbf{B}} + \sum_{k=1}^{N_{\text{sen}}} Obs_{j+1}^{k}$,
  $h'(d) \leftarrow cost(\hat{b}, d)$
  return $h'(d)$;
end procedure
procedure UPDATEC($c(\hat{b}, d)_{h(0,depth)}$)
  for $h_0 \quad to \quad h_{depth}$ **do**
    $Q(h,d)' \leftarrow cost(\hat{b}, d)$
    $Nb(h) \leftarrow Nb(h) + 1, Nb(hd) \leftarrow Nb(hd) + 1$
    $Q(h,d) \leftarrow Q(h,d) + \frac{Q(h,d)' - Q(h,d)}{Nb(hd)}$
  end for
end procedure
procedure SELECTPHASE(h, d)
  $d(h') \leftarrow \arg\max_{h \in node}(-\mathbf{Q}(h,d) + \sqrt{\frac{\log Nb(h)}{Nb(hd)}})$
  return $d(h')$;
end procedure
procedure SIMULPHASE(h)
  $d^* \leftarrow 0$
  while $loop < kt$ **do**
    $h^{'}, d \leftarrow$ HeuSampGennode(h)
    $dl \leftarrow$ SelectPhase($h^{'}, d$)
    if $dl > d^*$ **then** $d^* \leftarrow dl$
    end if
      $h \leftarrow h^{'}$
  end while;
  return $cost(\hat{b}, d^*)$;
end procedure

5.6 Conclusions

We have presented our P-UAV algorithm with obstacle collision avoidance for mobile target tracking in complex environments. The algorithm can be applied in real time. A Kalman filter was employed to update the belief states. The computational performance was enhanced using

heuristics and parallel computing. The P-UAV algorithm appropriately captures the exploration-exploitation tradeoff. The computational performance can be improved by increasing the number of CPU cores. Our experimental results show that P-UAV generally outperforms NBO (from [8]). The P-UAV algorithm contributes to the development and real-time implementation of autonomous UAVs.

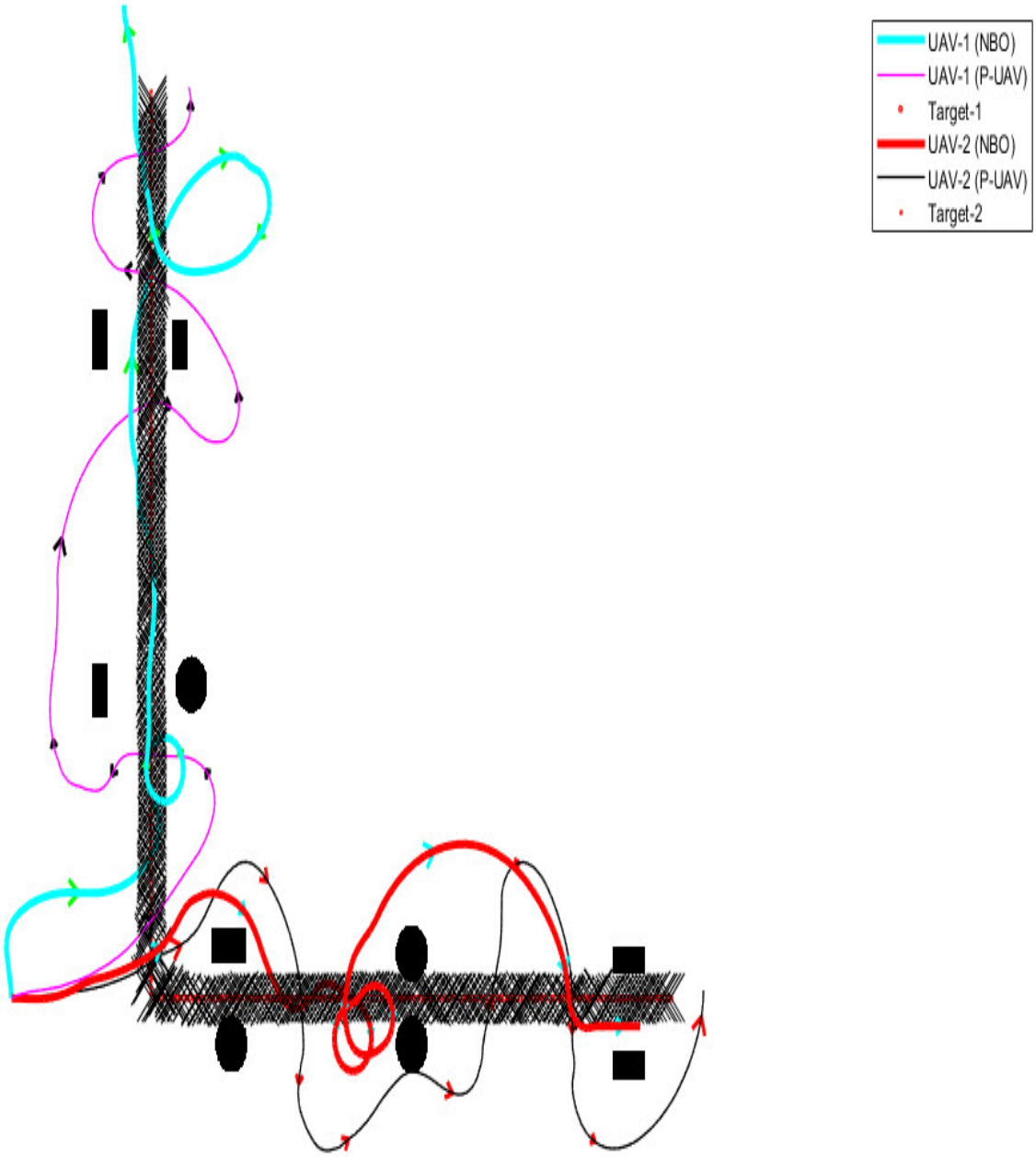

Figure 5.3: UAV trajectories for P-UAV and NBO pursuing mobile targets with obstacles

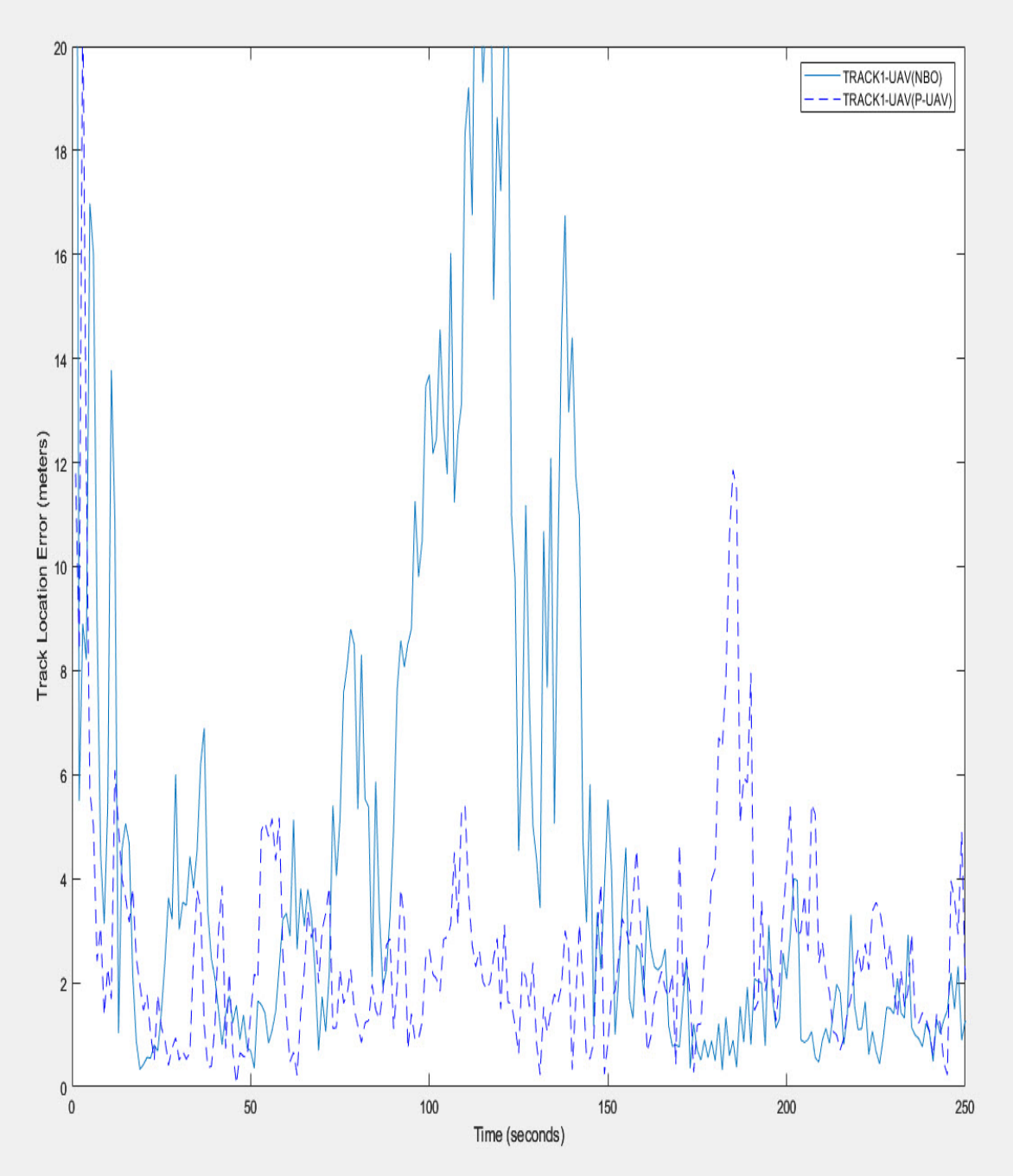

Figure 5.4: Track error for track I for P-UAV and NBO pursuing mobile targets with obstacles

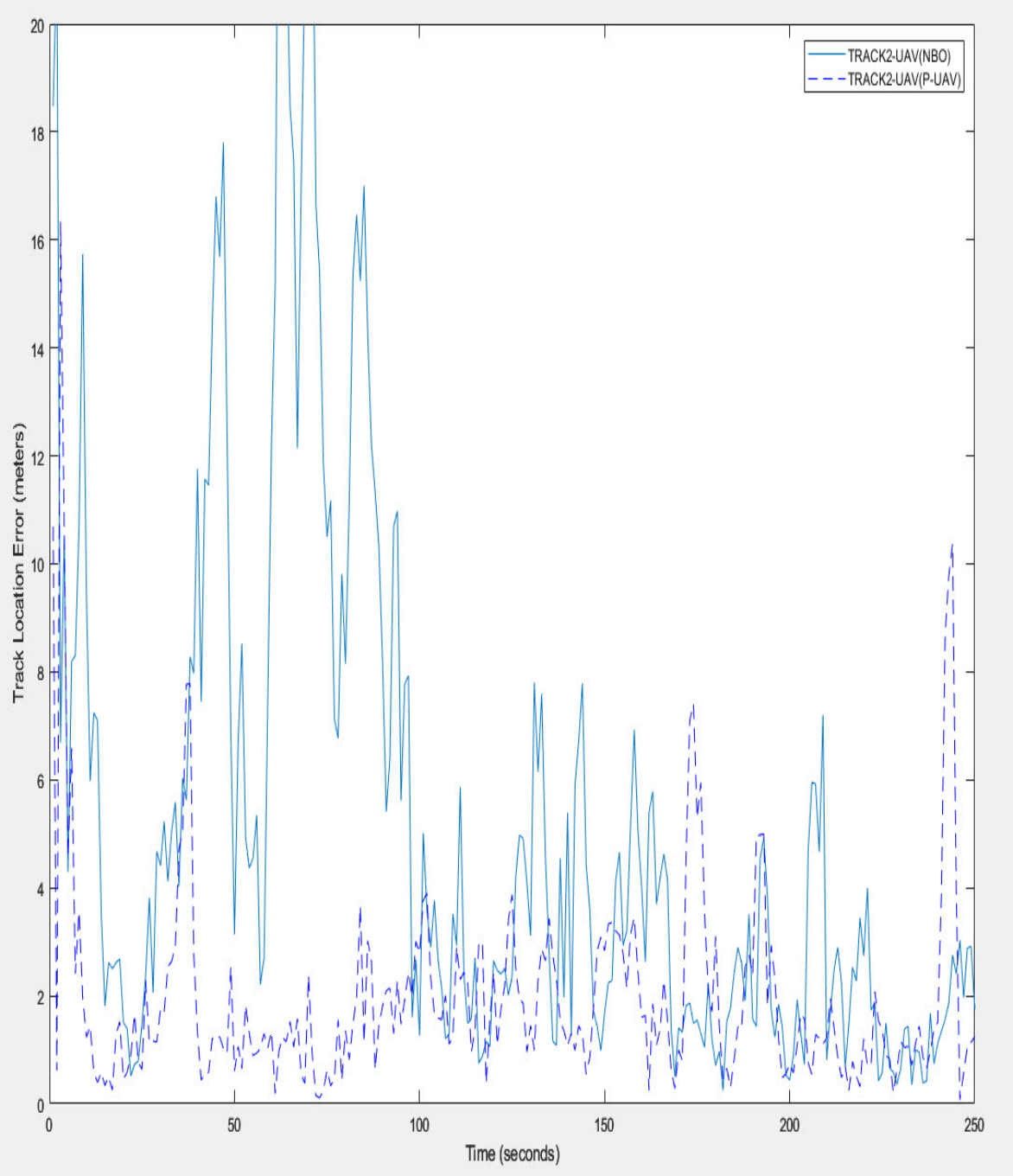

Figure 5.5: Track error for track II for P-UAV and NBO pursuing mobile targets with obstacles

Chapter 6

Experimental Results of P-UAV Algorithm for Path-Planning Problem in Complex Environments

The P-UAV algorithm with obstacle and threat collision avoidance, and wind compensation was developed from Chapter 5. This chapter shows the results of experiments for the P-UAV algorithm to track mobile targets in more complex environment such as a variety of obstacles and threat, and wind factor. The algorithm was implemented in Matlab and ran on a computer with an Intel I7 CPU (2.2GHz, six cores total) and 2GB RAM. The track errors for P-UAV and NBO (for tracks I and II) are shown in Table 6.1, which illustrates that P-UAV outperforms NBO. The errors over time of P-UAV and NBO are shown in Figures 6.1-6.3 which shows that the error in NBO is generally worse than in P-UAV.

Table 6.1: Track errors for P-UAV and NBO (meters)

Track I-error (NBO)	Track II-error (NBO)	Track I-error (P-UAV)	Track II-error (P-UAV)
3.174	2.791	2.881	2.572
3.378	2.830	2.925	2.437
Average= 3.276	2.811	2.903	2.505

6.1 The Heuristic Random Sampling Process

Our P-UAV method uses a heuristic random sampling process with three groups of angles (positive, zero, and negative) from the root node of search tree to the leaf node, all nodes uses a same forward accerelation. In the time horizon H=1, one of zero angle is inserted into the search tree, the time horizon H=2 with three of zero angle into, and the time horizon H=3 with nine of zero angle, as shown in Figure 3.1-3.2.

The process of simulation uses only one level of a search tree with three groups of angles with the limited number of loop (40 loops). After that, the maximun value is seeked for updating value from the left node to the root node of a search tree.

The zero angle of blank angle developed from a general vehicle heuristic with a initial state of vehicle or a stable state, is the most important to help find the shortest path from UAVs toward the targets, as shown in Figure 6.4-6.6. The P-UAV algorithm with a heuristic random-sampling process is superiors than the P-UAV algorithm with a uniform random-sampling process, as shown in Table 6.2.

Table 6.2: Track errors for P-UAV with a heuristic random-sampling process and P-UAV with a uniform random-sampling process (meters)

T1(P-UAV+Heu)	T1(P-UAV+Uni)	T2(P-UAV+Heu)	T2(P-UAV+Uni)
3.8484	3.9325	2.0635	5.447
3.5493	3.7175	2.8938	4.9553
Average= 3.6989	3.8250	2.4787	5.2012

6.2 Wind Compensation

We set the wind velocity to 10 m/s in the east direction , The P-UAV algorithm with the wind compensation outperforms the P-UAV algorithm without the wind compensation, as shown in Figure 6.7-6.9 and Table 6.3. We eliminate the small effect of wind and apply the approximate wind velocity to The P-UAV algorithm. The UAV position is updated with the wind compensation by $m^i_{j+1} = m^i_j + V^i_j T\cos(\omega^i_j) + sw_{x-k}$ and $n^i_{j+1} = n^i_j + V^i_j T\sin(\omega^i_j) + sw_{y-k}$, and then bring them to be calculated by the P-UAV algorithm.

Table 6.3: Track errors for P-UAV with a wind compensation and P-UAV without a wind compensation (meters)

T1(P-UAV+WindComp)	T1(P-UAV)	T2(P-UAV+WindComp)	T2(P-UAV)
4.7463	4.8165	4.2365	8.6454
3.2329	3.5275	3.9493	7.8126
Average= 3.9896	4.172	4.0929	8.229

6.3 Threat Avoidance

To avoid threats collision with UAV, the radar search for detecting threat with the location , velocity, and direction. This data is submitted into the heuristic Kalman filter process for predicting the next location of threat as shown in Algorithm 2. Our Threat-penalty algorithm computes the distance of each UAVs to all threats, and chooses the closest distance for each UAV to process for penalty called Threat-penalty. This penalty is calculated into the cost of equation 3.7. The P-UAV algorithm with threat avoidance can escape threats as shown in Figure 6.10-6.12 and Table 6.4.

Table 6.4: Track errors for P-UAV with threat avoidance (meters)

T1-(P-UAV+threat avoid)	T1-(Threats)	T2-(P-UAV+threat avoid)	T2-(Threats)
7.4261	5.1005	4.8137	5.1005

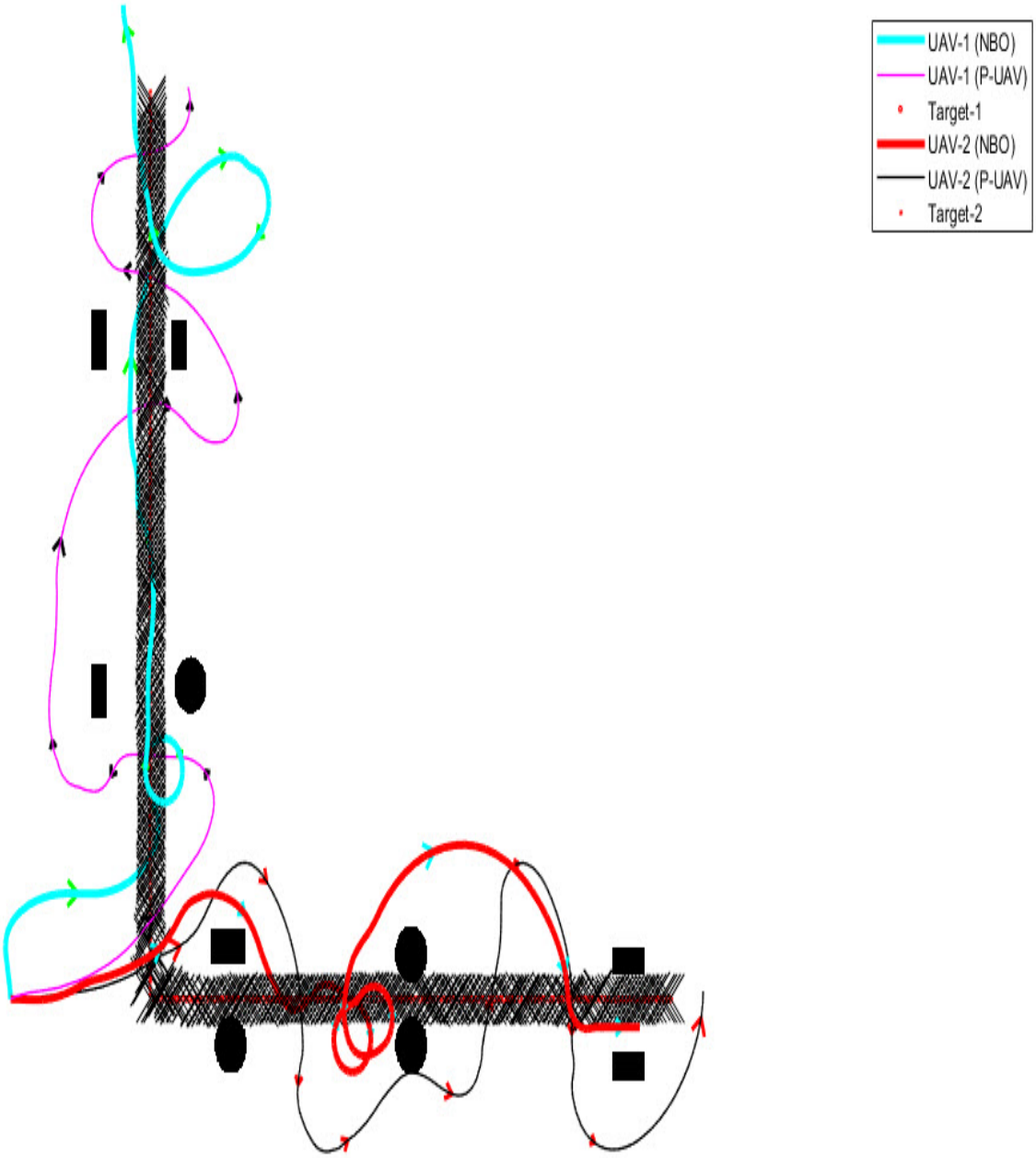

Figure 6.1: UAV trajectories for P-UAV and NBO pursuing mobile targets with obstacles

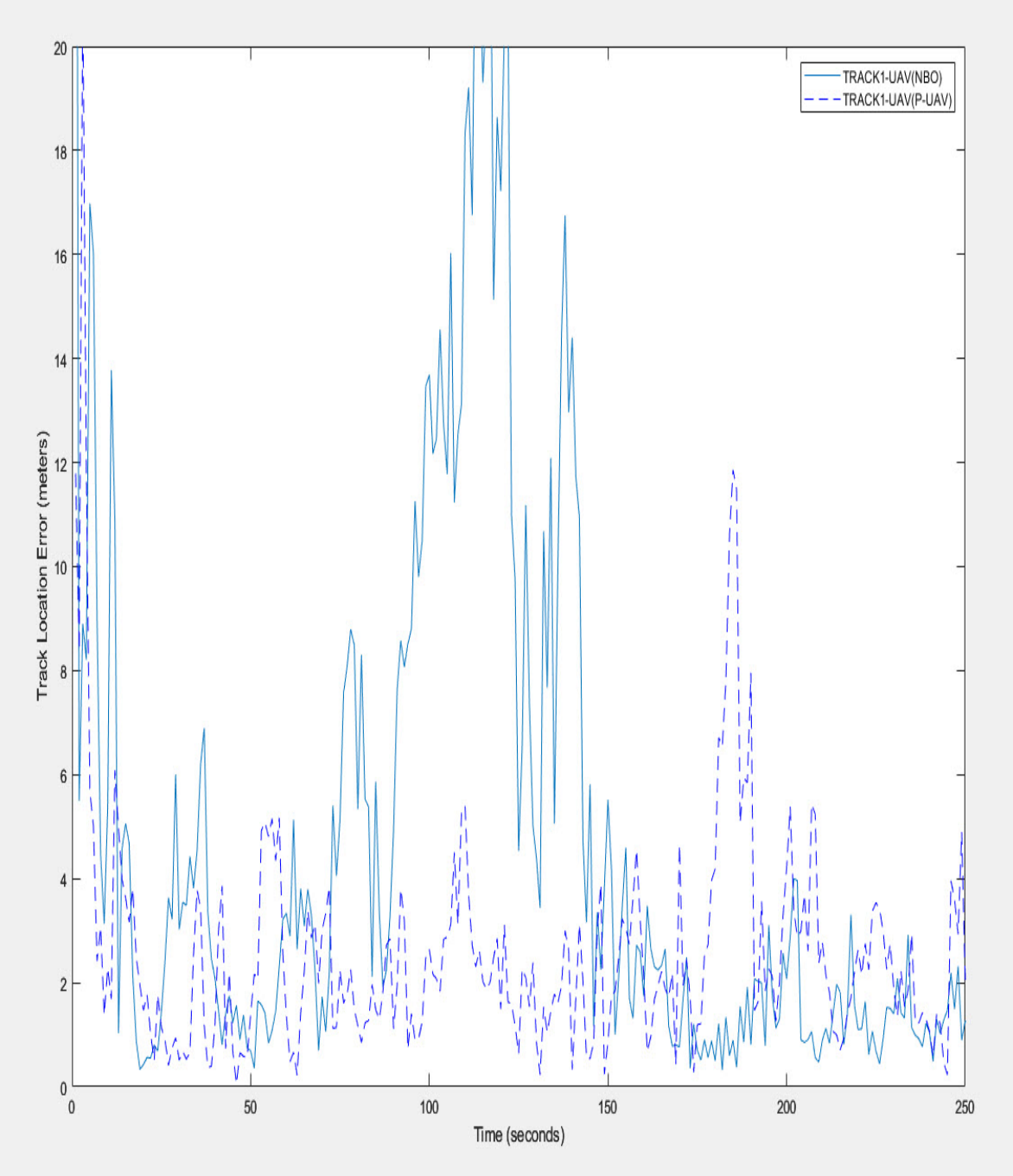

Figure 6.2: Track I error for P-UAV and NBO pursuing mobile targets with obstacles

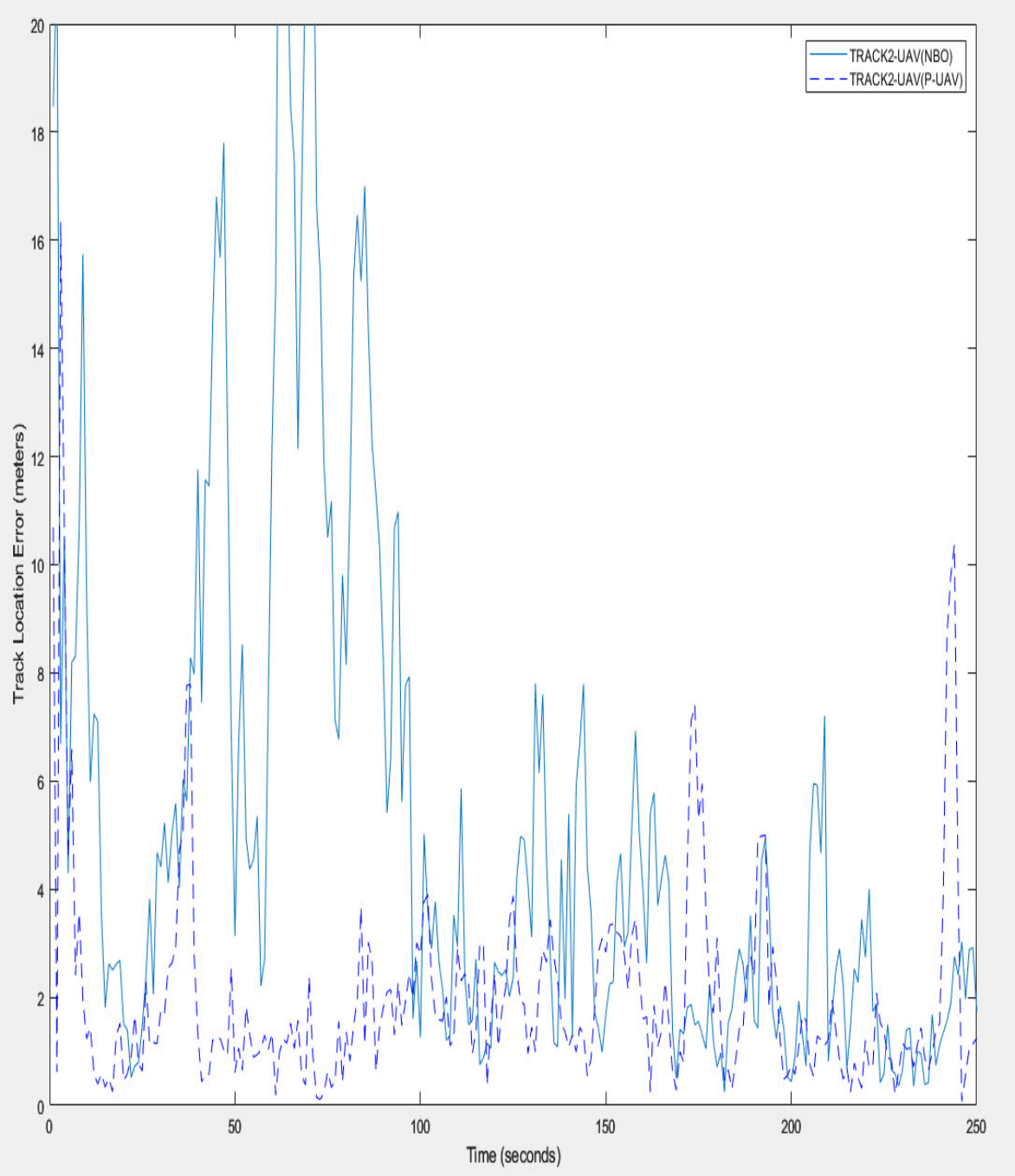

Figure 6.3: Track II error for P-UAV and NBO pursuing mobile targets with obstacles

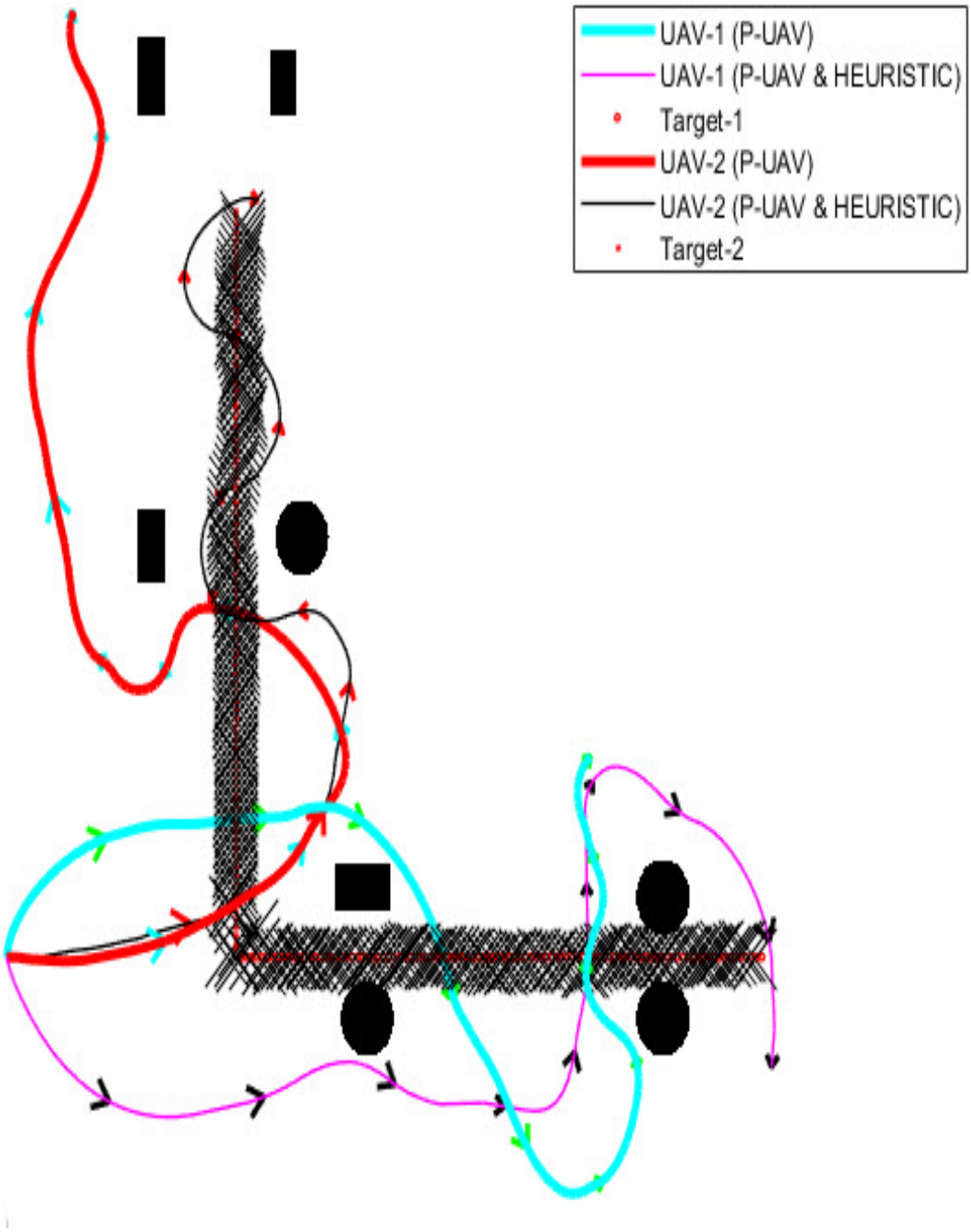

Figure 6.4: UAV trajectories for P-UAV with a heuristic random-sampling process and P-UAV with a uniform random- sampling process

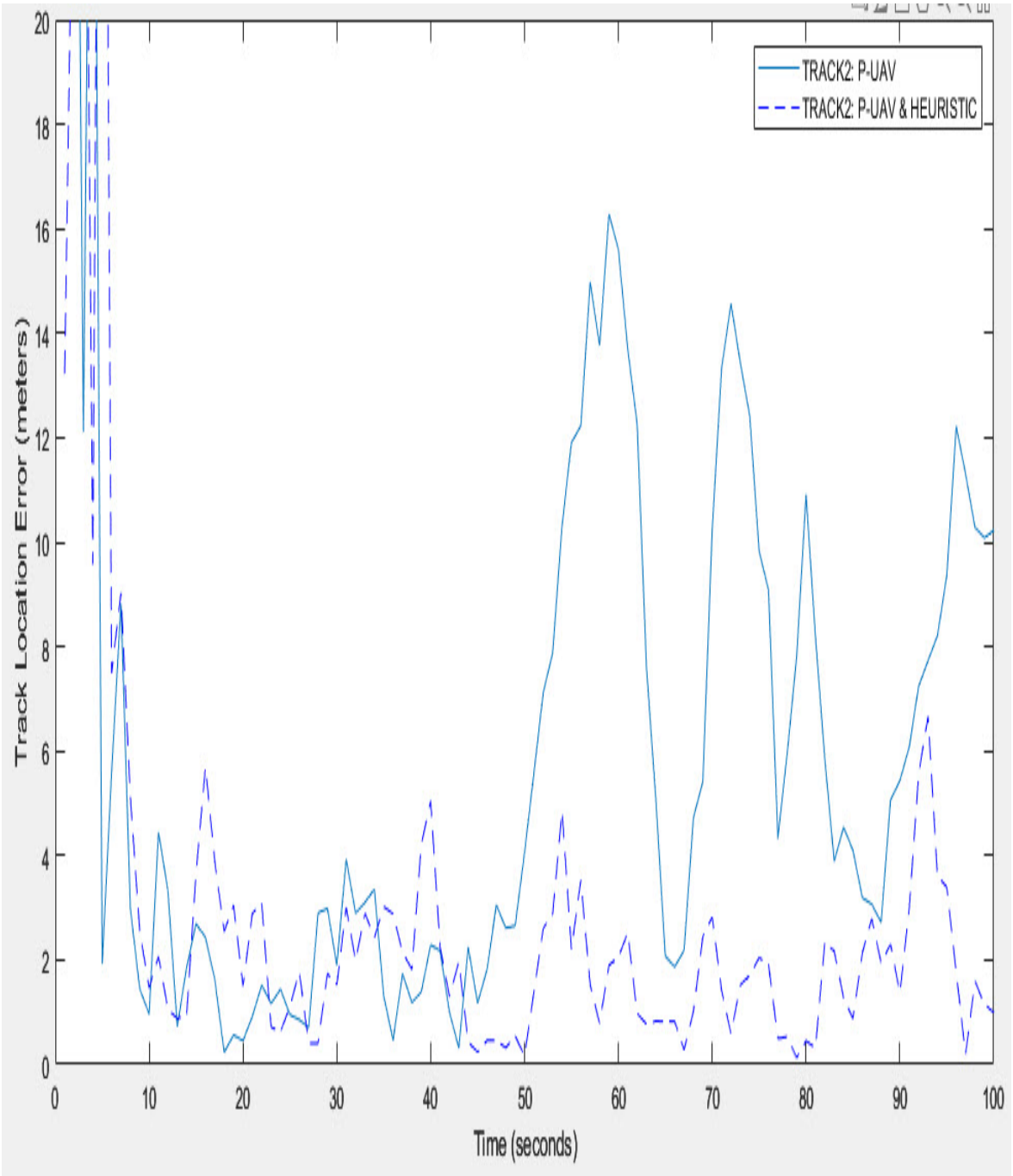

Figure 6.5: Track I error for P-UAV with a heuristic random-sampling process and P-UAV with a uniform random- sampling process

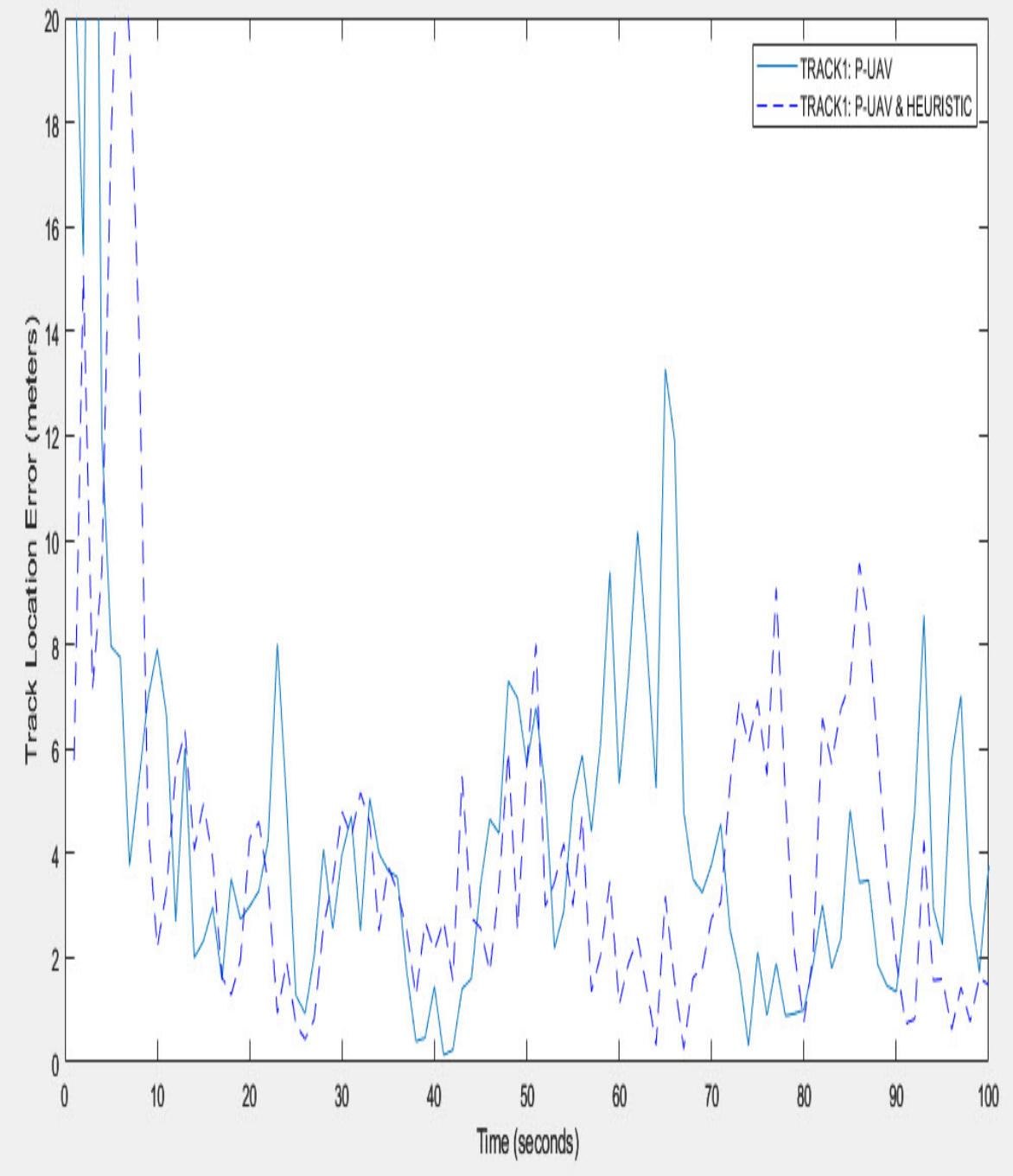

Figure 6.6: Track II error for P-UAV with a heuristic random-sampling process and P-UAV with a uniform random- sampling process

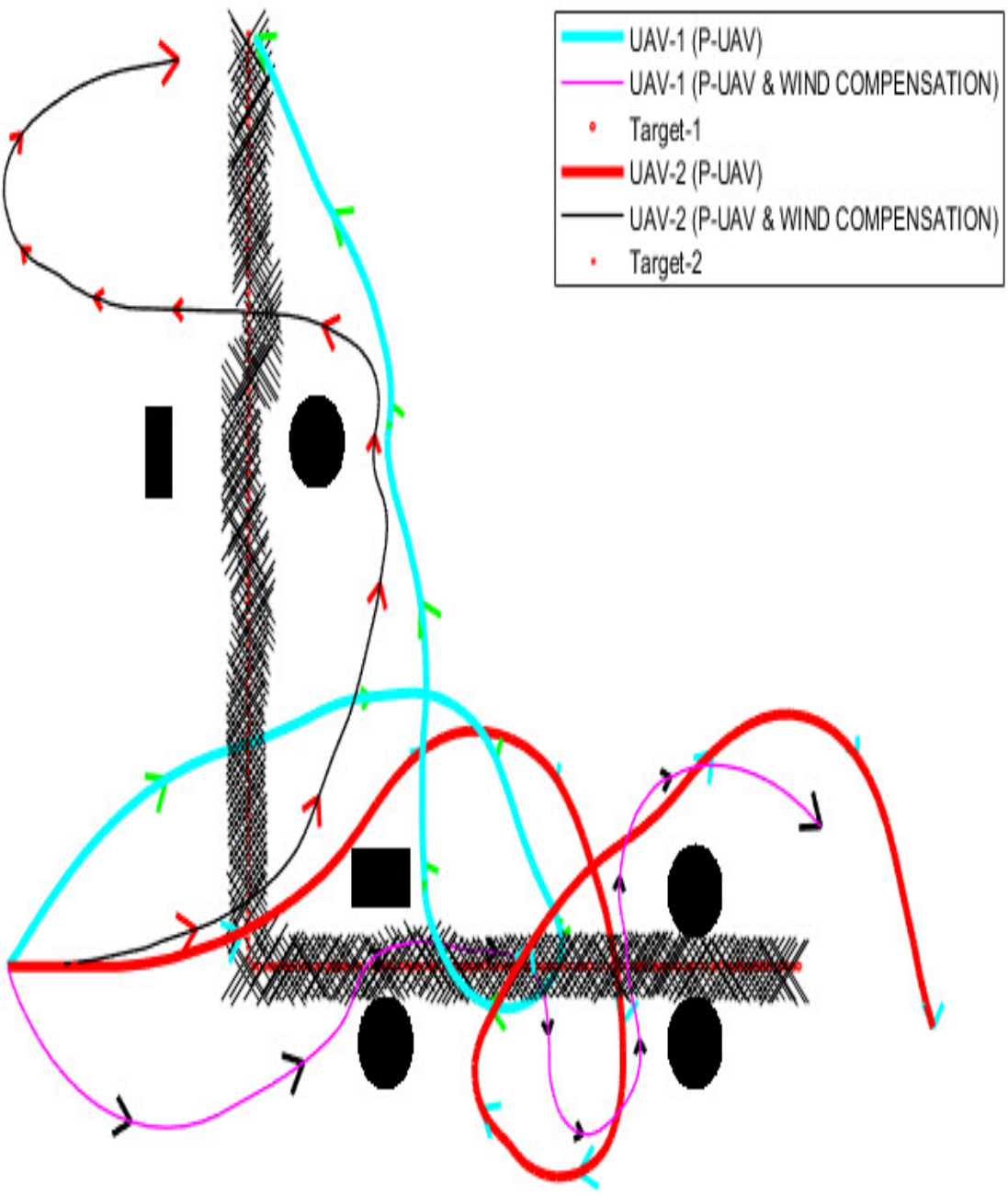

Figure 6.7: UAV trajectories for P-UAV with a wind compensation and P-UAV without a wind compensation

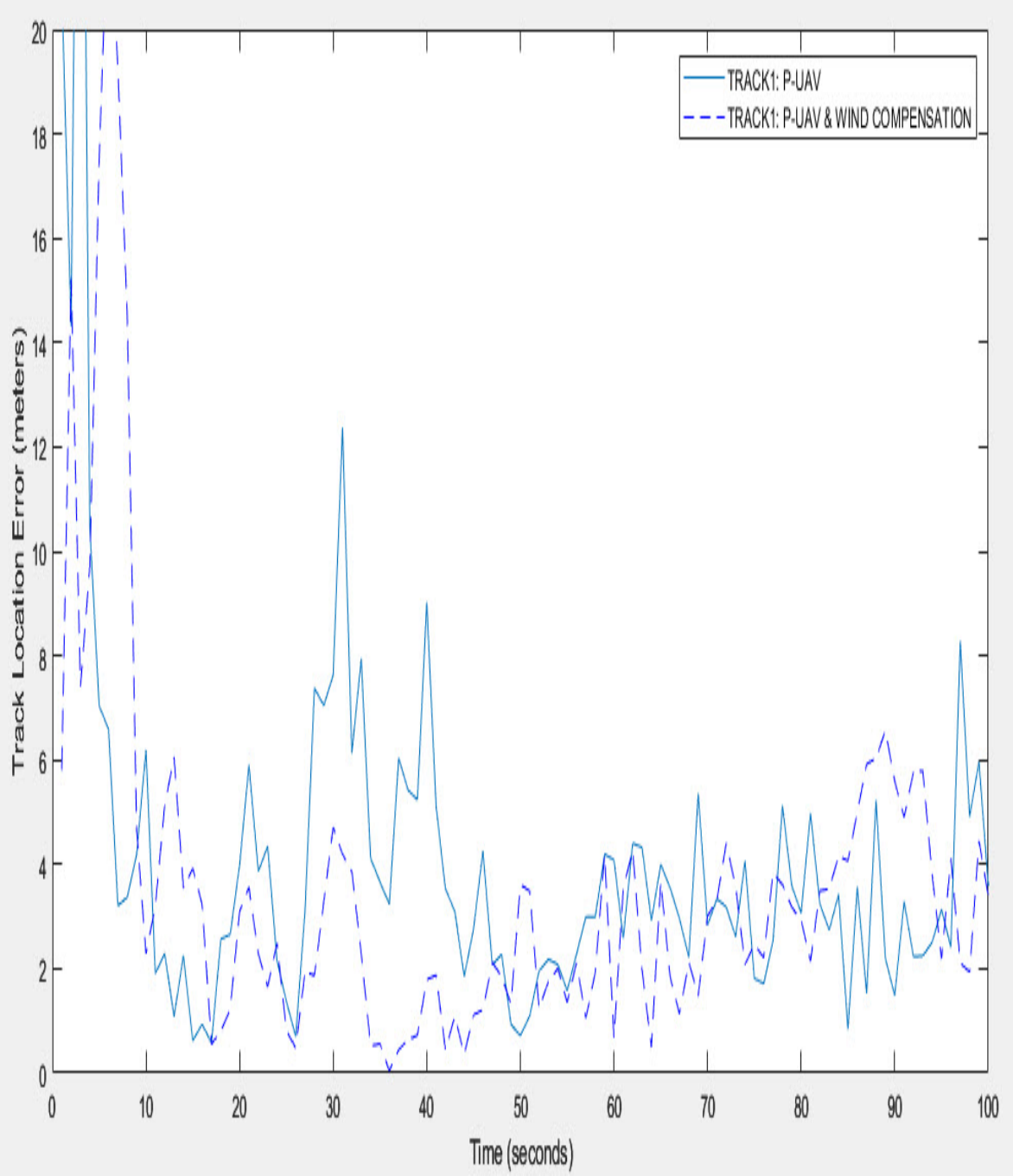

Figure 6.8: Track I error for P-UAV with a wind compensation and P-UAV without a wind compensation

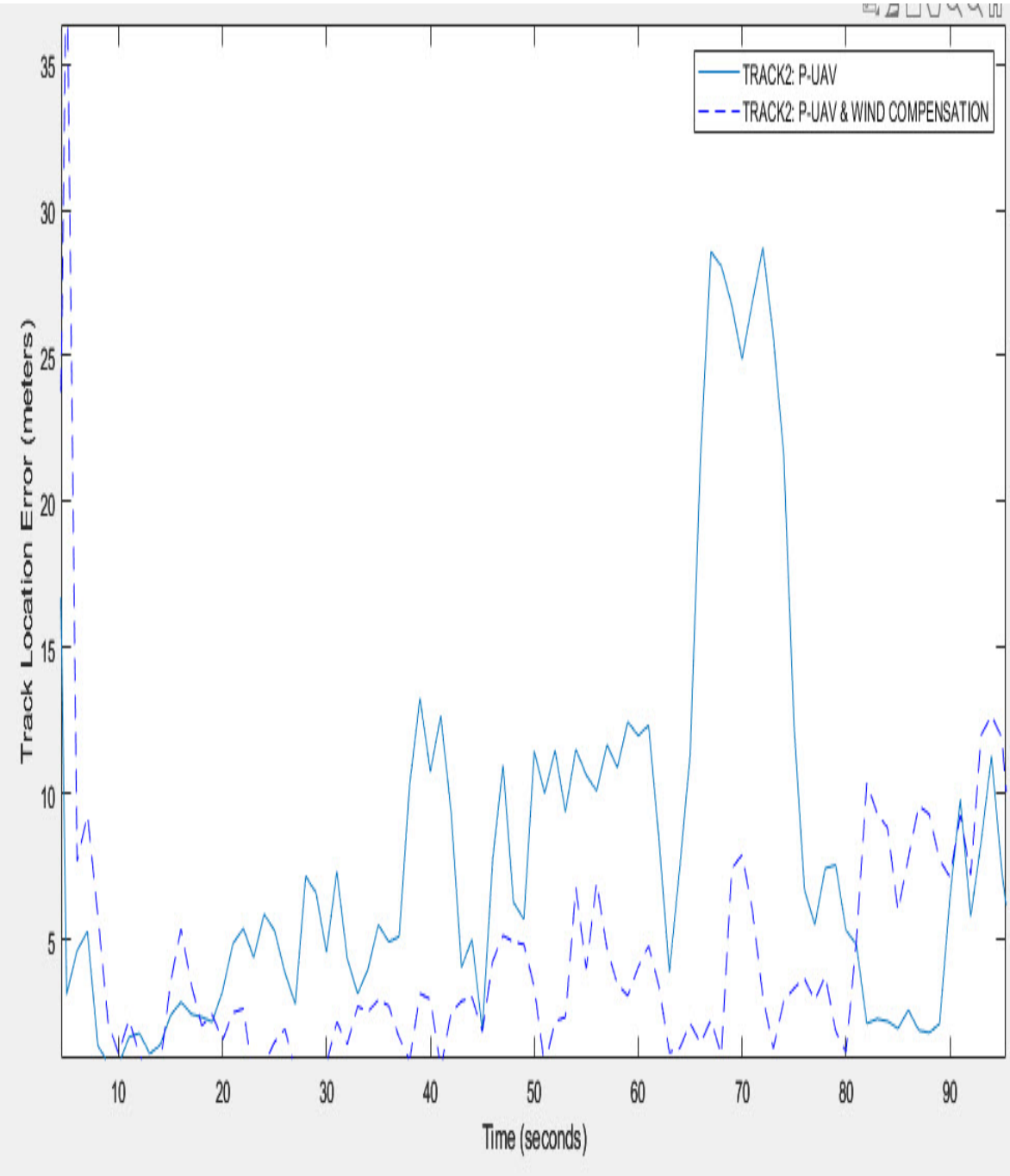

Figure 6.9: Track II error for P-UAV with a wind compensation and P-UAV without a wind compensation

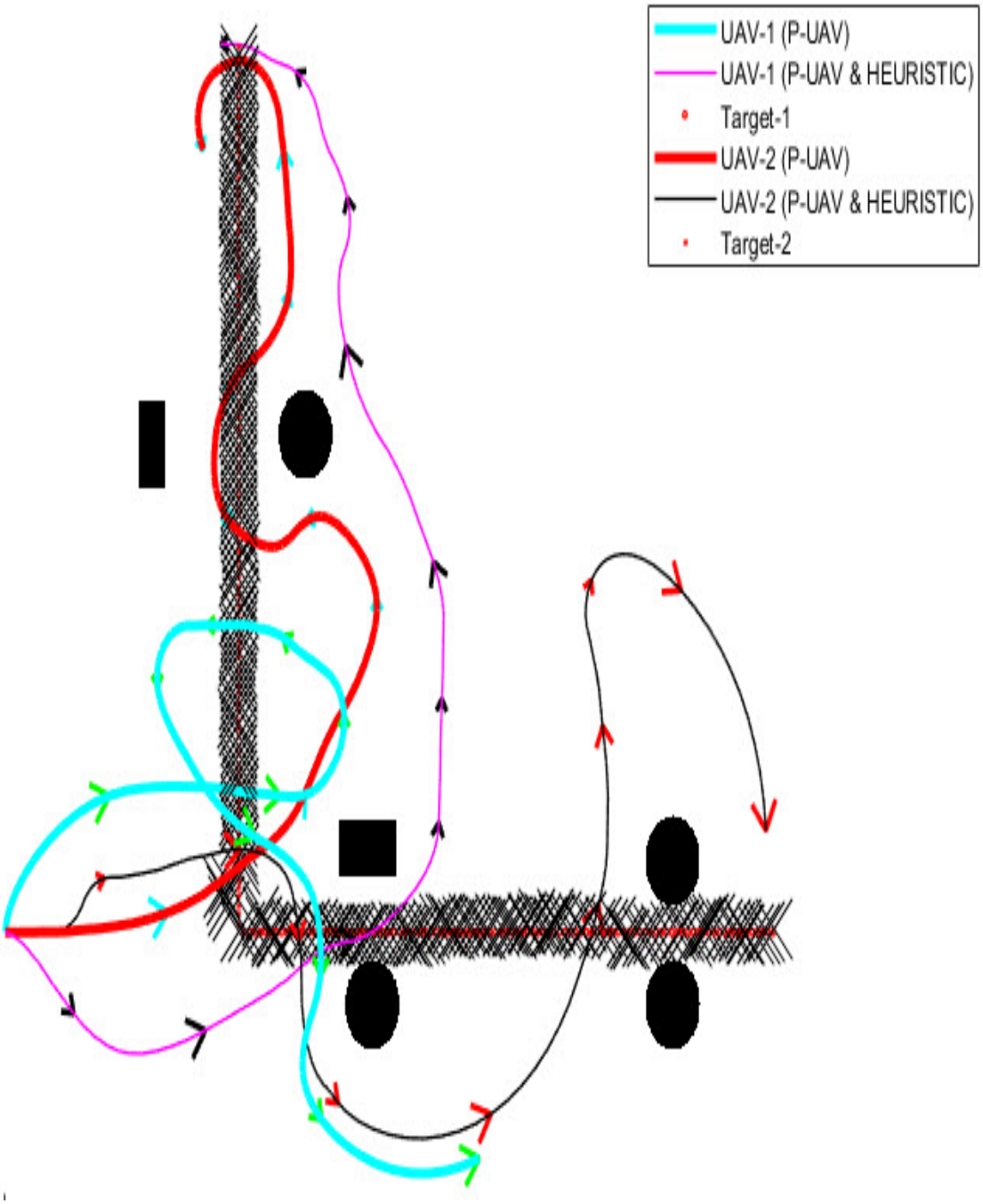

Figure 6.10: UAV trajectories for P-UAV with threat avoidance

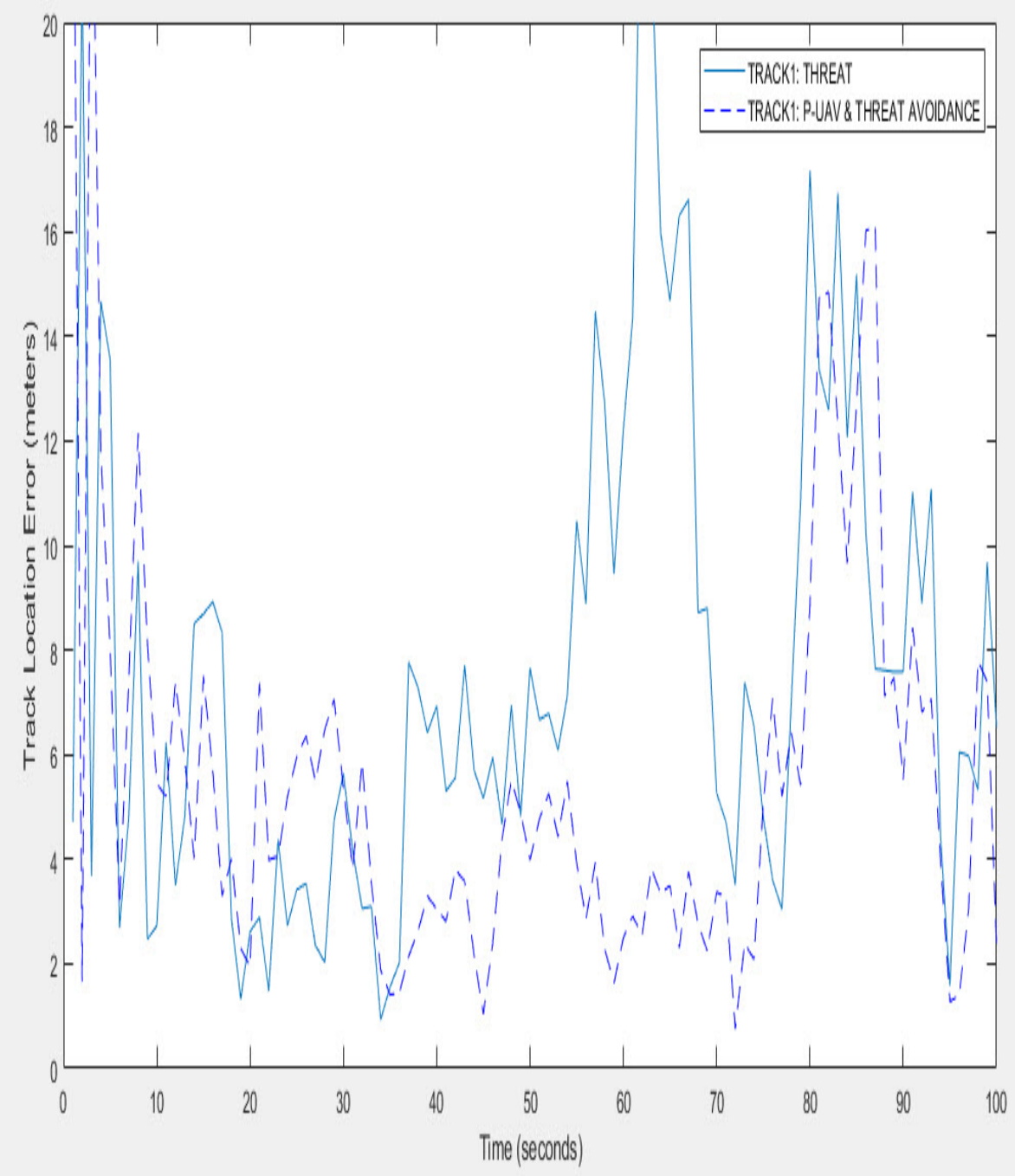

Figure 6.11: Track I error for P-UAV with threat avoidance

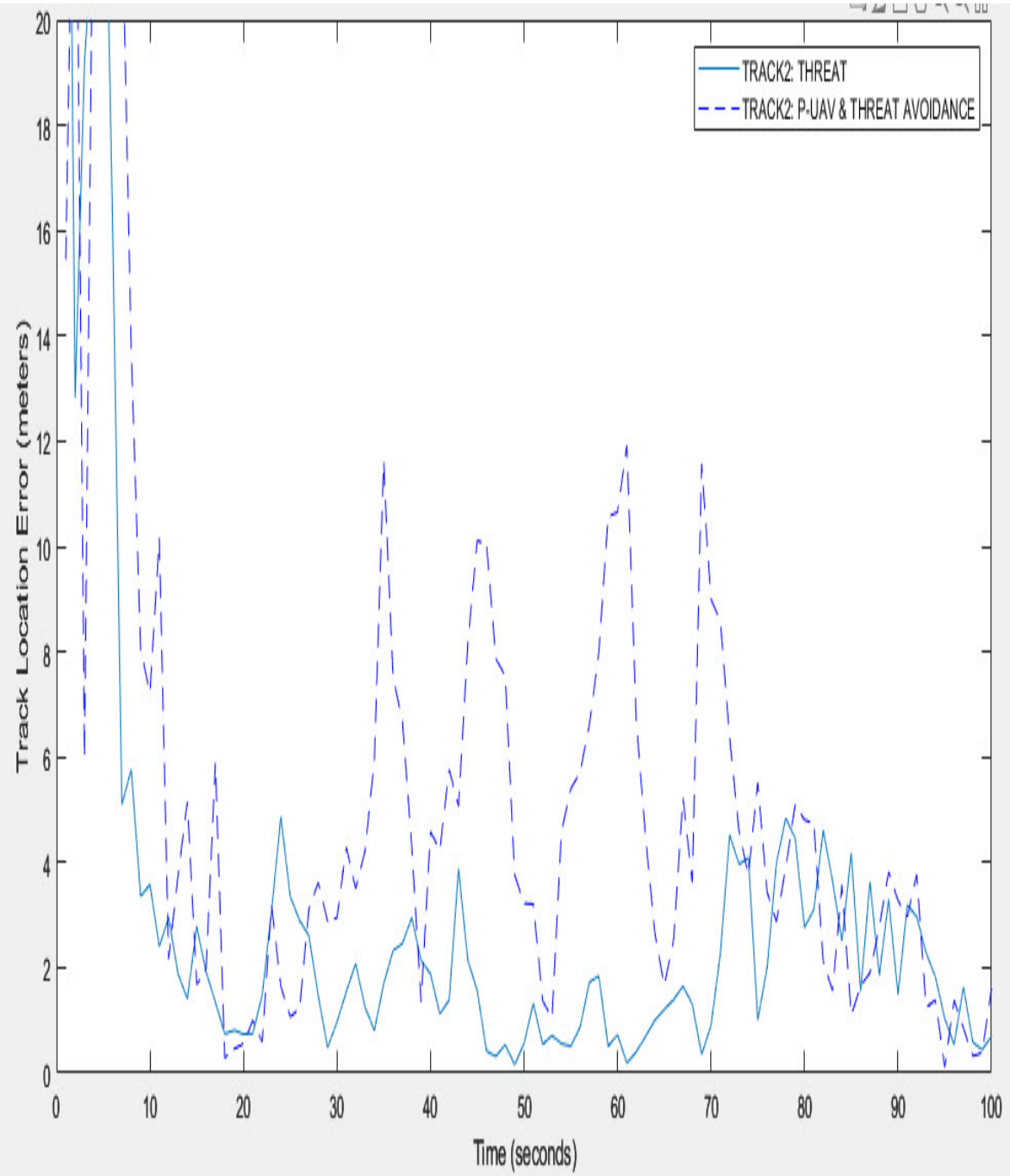

Figure 6.12: Track II error for P-UAV with threat avoidance

Chapter 7

Conclusions Summary

We have presented our P-UAV algorithm in complex environments with obstacle and threat collision avoidance, and wind factor, to track ground mobile targets. The algorithm can be applied in real time. A Kalman filter was employed to update the belief states. The computational performance was enhanced using heuristics and parallel computing. The P-UAV algorithm appropriately captures the exploration-exploitation tradeoff. The computational performance can be improved by increasing the number of CPU cores. Our experimental results show that P-UAV generally outperforms NBO from [8].

Our heuristic random-sampling process with zero angle for bank angle is the most important for a vehicle control heuristic to path-planning problem, reducing significantly the computation time. This heuristic can find the solution of problem by converge directly to a optimal solution. Positive and negative angles of bank angle support in computing the optimal value by balancing actions between negative and positive angle, designed conceptually from a practical vehicle control. This heuristic random-sampling process enhance our P-UAV algorithm to operate practically in real-time application. The P-UAV algorithm contributes to the development and real-time implementation of autonomous UAVs.

In hardware implementation, graphics processing unit (GPU) planned to be utilized by our P-UAV algorithm in future research is devised initially to enhance processing of computer graphics in computer. GPU parallel computing is widely used in many areas including graphics rendering, real-time processing. GPU is powerful for computer execution because it has a huge number of multicores inside with parallel processing architecture. GPU is currently developed for general and real-time applications, such as autonomous vehicle control . Program is executed by GPU by utilizing a parallel concept with many a large number of processor cores. Our P-UAV algorithm is designed theoretically and practically for a real-time computing on GPU, making suitable for control devices with a convenient size in daily life devices.

Bibliography

[1] S. Aggarwal and N. Kumar. Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges. *Computer communications*, 149:270–299, 2020.

[2] N. J. Nilsson. *Principles of artificial intelligence*. Springer Science & Business Media, 1982.

[3] A. Lazarowska. A discrete artificial potential field for ship trajectory planning. *The Journal of Navigation*, 73(1):233–251, 2020.

[4] A. Vasutapituks and E. K. P. Chong. Design of Autonomous UAV Guidance System Using Monte Carlo Tree Search. *in Proceedings of 2022 7th International Conference on Business and Industrial Research (ICBIR)*, pages 677–682.

[5] A. Vasutapituks and E. K. P. Chong. An autonomous UAV path-planning algorithm for mobile target tracking in complex environment. *in Proceedings of AMM2023 and ICNA2023*, pages 105–115, 2023.

[6] E. K. P. Chong, C. Kreucher, and O. Hero. Partially observable Markov decision process approximations for adaptive sensing. *Discrete Event Dynamic Systems*, 19:377–422.

[7] D. Silver and J. a Veness. Monte-Carlo planning in large POMDPs. *In Advances in Neural Information Processing Systems 23*, page 2164–12172.

[8] S. A. Miller, Z. A. Harris, and E. K. P. Chong. *EURASIP Journal on Applied Signal Processing, special issue on Signal Processing Advances in Robots and Autonomy*.

[9] S. Ragi and E. K. P. Chong. Act to see and see to act: POMDP planning for objects search in clutter. *IEEE Trans. Aerosp. Electron. Syst.*, 49(4):2397–2412, 2013.

[10] S. Ragi and E. K. P. Chong. Dynamic UAV path planning for multitargte tracking. *in Proc. 2012 American Control Conference*, page 3845–3850.

[11] J. K. Li, D. Hsu, and W. S. Lee. Act to see and see to act: POMDP planning for objects search in clutter. *in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 5701–5707.

[12] A. Somani, N. Ye, D. Hsu, and W. S. Lee. Act to see and see to act: POMDP planning for objects search in clutter. *in Advances in Neural Information Processing Systems*, 26:1772–1780.

[13] R. Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. *in Proc. 5th Int. Conf. Comput. Games*, page 72–83.

[14] A. R. Carmo, J.-A. Delamer, Y. Watanabe, R. Ventura1, and C. P. C. Chanel. *in 9th International Conference on Prestigious Applications of Intelligent Systems*.

[15] Y. Xiao, S. Katt, T. Pas, S. Chen, and C. Amato. Online planning for target object search in clutter under partial observability. *In Proceedings of the International Conference on Robotics and Automation*, page 8241–8247.

[16] T. Lozano-Pérez and M.A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles . *Commun. ACM 22 (10)*, page 560–570, 1979.

[17] L. E. Kavraki, P. Svestka, J-C Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[18] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[19] E. Masehian and D. Sedighizadeh. Multi-objective robot motion planning using a particle swarm optimization model. *Journal of Zhejiang University SCIENCE C*, 11:607–619, 2010.

[20] N. Mansard, A. DelPrete, M. Geisert, S. Tonneau, and O. Stasse. Using a memory of motion to efficiently warm-start a nonlinear predictive controller. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2986–2993. IEEE, 2018.

[21] D. Hsu, J.-C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. In *Robotics Research: Results of the 12th International Symposium ISRR*, pages 83–97. Springer, 2007.

[22] J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. In *Robotics Research: The Seventh International Symposium*, pages 249–264. Springer, 1996.

[23] C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent and Robotic Systems*, 57:65–100, 2010.

[24] J. Yang, P. Dymond, and M. Jenkin. Practicality-based probabilistic roadmaps method. In *2011 Canadian conference on computer and robot vision*, pages 102–108. IEEE, 2011.

[25] Z. Lee and X. Chen. Path planning approach based on probabilistic roadmap for sensor based car-like robot in unknown environments. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, volume 3, pages 2907–2912. IEEE, 2004.

[26] M. U. Farooq, Z. Ziyang, and M. Ejaz. Quadrotor uavs flying formation reconfiguration with collision avoidance using probabilistic roadmap algorithm. In *2017 International Conference on Computer Systems, Electronics and Control (ICCSEC)*, pages 866–870. IEEE, 2017.

[27] S. LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

[28] D. Devaurs, T. Siméon, and J. Cortés. Optimal path planning in complex cost spaces with sampling-based algorithms. *IEEE Transactions on Automation Science and Engineering*, 13(2):415–424, 2015.

[29] D. Zhang, Y. Xu, and X. Yao. An improved path planning algorithm for unmanned aerial vehicle based on rrt-connect. In *2018 37th Chinese control conference (CCC)*, pages 4854–4858. IEEE, 2018.

[30] N. Wen, L. Zhao, X. Su, and P. Ma. Uav online path planning algorithm in a low altitude dangerous environment. *IEEE/CAA Journal of Automatica Sinica*, 2(2):173–185, 2015.

[31] A. Budiyanto, A. Cahyadi, T. B. Adji, and O. Wahyunggoro. Uav obstacle avoidance using potential field under dynamic environment. In *2015 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, pages 187–192. IEEE, 2015.

[32] M. M. Trujillo, M. Darrah, K. Speransky, B. DeRoos, and M. Wathen. Optimized flight path for 3d mapping of an area with structures using a multirotor. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 905–910. IEEE, 2016.

[33] R. M. C. Santiago, A. L. De Ocampo, A. T. Ubando, A. A. Bandala, and E. P Dadios. Path planning for mobile robots using genetic algorithm and probabilistic roadmap. In *2017IEEE 9th international conference on humanoid, nanotechnology, information technology, communication and control, environment and management (HNICEM)*, pages 1–5. IEEE, 2017.

[34] J. Tao, C. Zhong, L. Gao, and H. Deng. A study on path planning of unmanned aerial vehicle based on improved genetic algorithm. In *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, volume 2, pages 392–395. IEEE, 2016.

[35] T. R. Schäfle, S. Mohamed, N. Uchiyama, and O. Sawodny. Coverage path planning for mobile robots using genetic algorithm with energy optimization. In *2016 International Electronics Symposium (IES)*, pages 99–104. IEEE, 2016.

[36] C. Huang, Y. Lan, Y. Liu, W. Zhou, H. Pei, Lo. Yang, Y. Cheng, Y. Hao, and Y. Peng. A new dynamic path planning approach for unmanned aerial vehicles. *Complexity*, 2018:1–17, 2018.

[37] Z. Zhou, Y. Nie, and G. Min. Enhanced ant colony optimization algorithm for global path planning of mobile robots. In *2013 International Conference on Computational and Information Sciences*, pages 698–701. IEEE, 2013.

[38] W. Hao and X. Xu. Immune ant colony optimization network algorithm for multi-robot path planning. In *2014 IEEE 5th International Conference on Software Engineering and Service Science*, pages 1118–1121. IEEE, 2014.

[39] R. Rashid, N. Perumal, I. Elamvazuthi, M. K. Tageldeen, M. A. Khan, and S. Parasuraman. Mobile robot path planning using ant colony optimization. In *2016 2nd IEEE international symposium on robotics and manufacturing automation (ROMA)*, pages 1–6. IEEE, 2016.

[40] X. Wu, W. Bai, Y. Xie, X. Sun, C. Deng, and H. Cui. A hybrid algorithm of particle swarm optimization, metropolis criterion and rts smoother for path planning of uavs. *Applied Soft Computing*, 73:735–747, 2018.

[41] H. S. Dewang, P. K. Mohanty, and S. Kundu. A robust path planning for mobile robot using smart particle swarm optimization. *Procedia computer science*, 133:290–297, 2018.

[42] Y. K. Ever. Using simplified swarm optimization on path planning for intelligent mobile robot. *Procedia computer science*, 120:83–90, 2017.

[43] S. Shao, Y. Peng, C. He, and Y. Du. Efficient path planning for uav formation via comprehensively improved particle swarm optimization. *ISA transactions*, 97:415–430, 2020.

[44] M. M. Kurdi, A. K. Dadykin, I. Elzein, and I. S. Ahmad. Proposed system of artificial neural network for positioning and navigation of uav-ugv. In *2018 Electric Electronics, Computer Science, Biomedical Engineerings' Meeting (EBBT)*, pages 1–6. IEEE, 2018.

[45] Y. Zhang, Y. Zhang, Z. Liu, Z. Yu, and Y. Qu. Line-of-sight path following control on uav with sideslip estimation and compensation. In *2018 37th Chinese Control Conference (CCC)*, pages 4711–4716. IEEE, 2018.

[46] S.-Y. Park, C. S. Shin, D. Jeong, and H. Lee. Dronenetx: Network reconstruction through connectivity probing and relay deployment by multiple uavs in ad hoc networks. *IEEE Transactions on Vehicular Technology*, 67(11):11192–11207, 2018.

[47] Y. Pan, S. Li, X. Zhang, J. Liu, Z. Huang, and T. Zhu. Directional monitoring of multiple moving targets by multiple unmanned aerial vehicles. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.

[48] L. E. Parker. Cooperative robotics for multi-target observation. *Intelligent Automation & Soft Computing*, 5(1):5–19, 1999.

[49] A. Kolling and S. Carpin. Multirobot cooperation for surveillance of multiple moving targets-a new behavioral approach. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1311–1316. IEEE, 2006.

[50] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *in IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[51] Z. N. Sunberg and M. J. Kochenderfer. Online algorithms for POMDPs with continuous state, action, and observation spaces. *in Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS)*, page 259–263.