Declaration

I, Menzi Mthwecu (1831125), am a student registered for the degree of Masters of Science (MSc) by dissertation in the academic year 2019.

I hereby declare the following:

I am aware that plagiarism (the use of someone else's work without their permission and/or without acknowledging the original source) is wrong.

I confirm that the work submitted for assessment for the above degree is my own unaided work except where explicitly indicated otherwise.

I have not submitted this work before for any other degree or examination at this or any other University.

I have followed the required conventions in referencing the thoughts and ideas of others.

I understand that the University of the Witwatersrand may take disciplinary action against me if there is a belief that this is not my own unaided work or that I have failed to acknowledge the source of the ideas or words in my writing.

Menzi Mthwecu

Signed on the 21 day of May 2019

Efficient search and tracking for non-stationary targets

Menzi Mthwecu

A dissertation presented in fulfillment of the
requirements for the degree of:
Master of Science

School of Computer Science and Applied Mathematics
University of the Witwatersrand

Abstract

Target search and tracking are fields which stand to benefit from guidance by intelligent, autonomous robots. We consider the problem of building an algorithm to guide a target search and tracking algorithm to autonomously search for, and track, a moving target. The thesis is motivated using the example of a drone which is tasked with finding and tracking a target in a restricted area. The drone is constrained by sparse signal information, time, and its limited knowledge of how the target moves. Most of the existing methods used to guide such robots, such as Bayesian Optimization [6][25] and Kinematic Motion Models [4], do not consider both the search and tracking elements of the problem. The remaining methods in use, such as the Dirichlet-Multinomial pseudo-count [32][26], are unsuitable because they require a long training period before they are effective. This thesis proposes a solution which uses elements of a Wonham Filter [33][12] to combine a proven search algorithm, Bayesian Optimization, with a ubiquitous tracking technique, Kinematic Motion Models. Secondarily, the solution considers a novel way of optimizing the search by using unsuccessful searches. The results show that the primary proposed solution may be more efficient at finding and tracking a target than existing methods, in certain circumstances.

Contents

Chapter 1

Introduction

A growing body of research is concerned with the development of intelligent, autonomous systems [1]. Particularly, target detection and tracking are important applications which would benefit from the development of intelligent, self-guided systems. Such systems would have a broad set of applications, including security surveillance and environmental monitoring [10][11][19][25].

For example, if an intruder is believed to be inside a restricted area, an autonomous security system should be able to find the intruder and follow it until support arrives. This could be done by sending a drone to search for the intruder. The drone would fly to a location in the restricted area, and then lower itself and deploy sensors to search the location. The drone may be unable to search while it flies, as its sensors have to be deployed while the drone is stationary, once the drone has arrived at a location. If the intruder was not detected, it would fly to another location and search again. The drone would keep doing this until it detected the intruder. After detecting the intruder, the drone would closely follow it. Although drone technology has been developed which allows drones to search as they fly, we use this example as motivation for a more general research problem in target search and tracking.

Clearly, if the automated drone was reliable, it would add value by allowing the security personnel to focus on other tasks. A successful drone would be able to find and track the intruder in a short space of time, with minimal human intervention.

An ideal drone would be effective given very limited signal information. Such sensors may only be able to detect signals such as temperature, sound, light, or vibrations [3][5]. As another example, if the intruder is moving in an opaque environment, then it may be optimal for the drone to use noise sensors because cameras would be ineffective. Alternatively the drone could use a heat sensor to detect the presence of the intruder. In both of these cases, the drone's sensor would only indicate the presence of a nearby intruder. However, the sensor would not indicate exactly where the intruder is. We refer to such signals and sensors as *sparse*.

Target tracking with sparse sensor readings has drawn interest from researchers due to its wide number of applications [3][25][10][11]. For example, assume that the area of interest is equipped with a Wireless Sensor Network (WSN) which is used to find and track the target, rather than a drone. The

WSN may consist of sparse sensors, each of which only indicates if the target is nearby or not. The WSN may reduce its energy requirements by keeping sensors in sleep mode when they are not in use [11] (see Figure 1.1 for an illustration of a potential sensor network layout in the aforementioned domain). Alternatively if the WSN has limited energy capacity, it may only be able to activate a single sensor at a time [5]. In such a case, the target search and tracking problem is very similar to that of the drone described above, because at each time point the WSN activates a single sensor to search a location for the target, and if the target is not found then the WSN would deactivate the active sensor and then activate a different sensor in search of the target. WSNs with sparse sensors have been of interest to researchers in the field of target detection and tracking [5]. It is common for WSNs to be used for target detection and tracking when only equipped with sparse sensors as described above, which can only detect signals such as ground vibrations or noise signals [5].

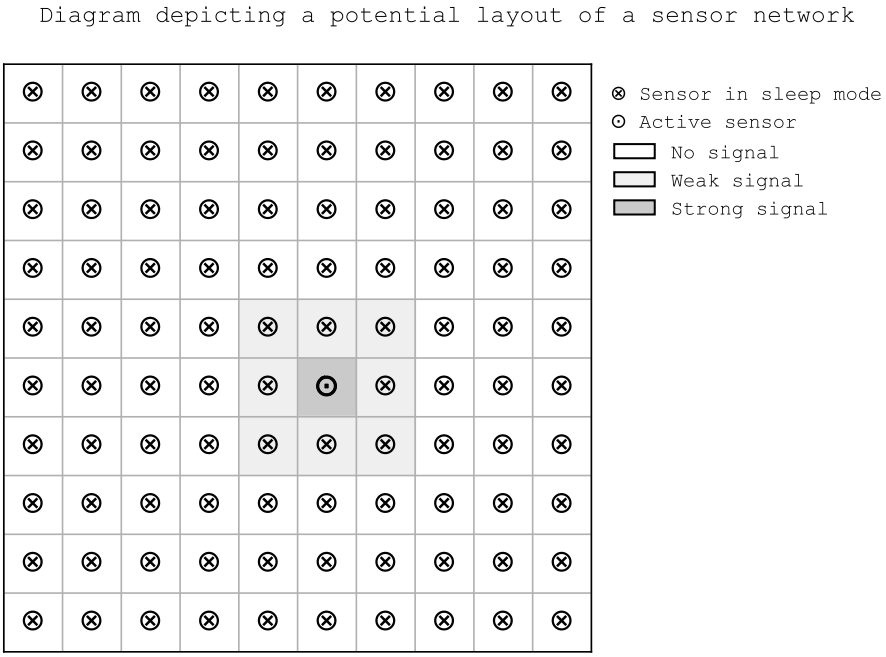


Figure 1.1: The active sensor receives a strong signal of any target which is very close to it (within the same grid cell). When a target is further away, but still within the nearby surrounds, then the active sensor receives a weak signal indicating that a target is in the nearby surrounds. Note that the signal received by the sensor only indicates that a target is nearby, but does not give the target's exact coordinates. In this 10×10 grid, the WSN could activate all 100 sensors and surely find the target. However, given an efficient search algorithm, the WSN could optimize its energy consumption by turning on a single sensor at a time to search for the target.

We use these examples as a motivation for the general problem considered, namely, searching for and tracking a target using sparse signals.

An additional constraint is that the search and tracking agent, which we assume to be an algorithm guiding a drone throughout this thesis, should be able to find the intruder given very limited knowledge about how the intruder moves. For example, before beginning its search, the algorithm guiding the drone may believe that the intruder moves at a relatively constant speed. However, the drone would not know the *exact* speed and direction of the intruder before beginning its search. Given this limited information about how the intruder moves, an ideal drone would still be able to find and track the intruder.

To motivate for the above assumption, we note the applications of such a target search and tracking algorithm. In [20] the authors develop an algorithm to guide a robot to search for and track concentrations of pollution in a similar problem setup as in this thesis. In such a setting, it is plausible that the algorithm has some knowledge of how quickly the pollutant moves through the environment. For example, the robot may be used to monitor pollutants in a lake used as a reservoir [20]. Based on historical observations, one may be able to build a model for how the pollutants move through the water over time. In [25] an algorithm is developed to guide a drone to search for a poacher in a similar problem setup. If the drone's sole purpose is to find poachers, then it is plausible that the algorithm guiding the drone would have some prior knowledge about how the poachers move. This may come from previous experience, or may be based on the types of poachers one would expect. For example, if the restricted area is an elephant reserve, then one may expect the poacher to move relatively slowly as their movement would be slowed when hauling such a large animal. In [10] they discuss applications of WSNs for target tracking, which include military applications, pollution monitoring and wildlife monitoring. If a drone or WSN is used to monitor wildlife, it may be possible for the algorithm to have some knowledge of how the target moves. For example, if searching for and tracking an animal in an environmental reserve, one may have prior knowledge about how quickly the animal moves. Furthermore, one may also know how the target is able to accelerate and decelerate, based on the historical findings of environmental researchers who study the particular animal.

Moreover, a successful drone would be able to find the intruder given a very limited time to search the area. Hence, the drone should have an efficient search technique. This requirement is critical because the intruder is moving. If the intruder was not moving, the drone could just search every single location in the restricted area, until it found the intruder. Put differently, the drone would not have a time-constrained search. However, if the intruder is moving, the drone must have a more efficient way of searching.

To make the search process quick and efficient, it would be ideal if the drone used what this thesis terms *negative information*. If the drone searches a location, but does *not* sense the intruder, or target, it has observed negative information. If the drone is somehow able to use this negative information, then its search can become faster and more efficient. Naturally if the drone searches a location and *does* get a signal of the target, it will have observed *positive information*.

There is existing research which considers the aforementioned target search and tracking problem, but under different assumptions. The first body of research [25] considers how to search for a target which is not moving. The search is carried out using a drone which has the same kind of sparse sensor described

above. The drone is guided by a popular search technique known as Bayesian Optimization [6]. The authors show that this technique is efficient, and optimal for the stationary target problem. However, the technique is ineffective if the target is moving.

There are also well-established techniques to follow a moving target after it has already been found [4]. These techniques work by observing how the target moves, and then predicting where it will go next. The techniques have been shown to be optimal methods for tracking a target which has already been found [4]. However, these methods do not consider how to initially find the target, before tracking it.

Another technique (the Dirichlet Multinomial pseudo-count) used to follow a moving target, after it has already been found, is shown in [26] and [32]. However, this technique requires the target to have habitual movement patterns in order to be effective. Additionally, the technique only becomes effective after the target has been followed for a long time. Moreover, the method does not consider how to initially find the target, before tracking it. Resultantly, it may not be ideal for this research's problem setup.

The authors of [19] consider both searching for, as well as tracking, a moving target. However they use a grounded robot to carry out the search. The grounded robot is similar to the drone in that it uses the same kind of sparse sensor to detect the presence of a nearby target. But the grounded robot searches while it moves, differently to the drone which does not search as it flies between search locations. Nevertheless, the authors also find that guiding the search using a variant of the the Bayesian Optimization technique (Sequential Bayesian Optimization) is most efficient.

However, the limitation of the algorithm developed in [19] is that it is only suitable for targets which move in a habitual manner. Additionally, it requires the target to move in a cyclical pattern, and for the length of each cycle to be known. These conditions do not hold in this research. In this research we will also consider targets which do not move in a habitual or cyclical manner. Hence our solution will require a different approach.

On the other hand, the existing research which considers how to use negative information does so from a different perspective. In [22], the authors define negative information similarly to how it is defined in this thesis. However they only attempt to use negative information to track, rather than search for, a target. Hence, this thesis considers a different way of using negative information.

Notably, none of the aforementioned related work directly addresses the problem introduced in this thesis. There are existing techniques [25] to efficiently search for a stationary target, using a drone with a sparse sensor. There are also existing techniques [4][26][32] for tracking a moving target after it has already been found. However, this thesis considers a subtly different problem: it attempts to use a drone with a sparse sensor to search for a moving target, and then track it.

Resultantly, the research aim of this thesis is to develop an algorithm to effectively search for a moving target, and then track it, using a drone (or more generally, a target tracking agent) equipped with a sparse sensor. The three primary constraints are limited signal information, time, and limited information about how the target moves. The limited signal information constraint requires the target tracking agent to use a sparse sensor, which only signals whether a

target is nearby or not. The time constraint means that the search has to be quick and efficient. We are interested in a solution which can find the target given a very limited number of search trials. Furthermore, the target tracking agent has limited information about how the target moves. Before beginning its search, it may know that the target moves at a constant speed. However, it would not know the exact speed and direction of the target before beginning its search.

The approach taken to solve the research problem draws inspiration from the related research. We consider a novel way of combining the aforementioned search techniques with the aforementioned tracking methods. This is done using components of what is known as a Wonham Filter [12][33]. A Wonham Filter allows one to model a process which cannot be directly observed. Here, the process is the intruder moving inside the restricted area. The intruder is never seen directly because the drone has a sparse sensor which only indicates if a target is nearby or not.

The use of components of the Wonham Filter also allows for the implicit use of negative information to make the search more efficient. However, this research also considers other ways of using negative information. It does so by attempting to predict where the target is *not* going, in order to predict where the target *is* going. This abstract concept is explained in Chapter 4.

Resultantly, the solution developed in this research is an improvement on [25] and [4], because they do not consider both the search and tracking aspects of the problem. The proposed algorithm also addresses the limitations of [19], [26] and [32], neither of which is suitable for a target which does not have habitual movement patterns.

The scope of this research is limited to finding and tracking a single target. If there are multiple targets in the area, then the solution proposed by this research would not be suitable. Additionally, this thesis only considers targets which are oblivious of the drone's presence. Such targets move independently of the drone's actions [1].

This thesis is structured as follows. Chapter 2 details all the relevant background theory. It discusses Reinforcement Learning which is a branch of research which this thesis draws inspiration from. It also covers the chosen search technique, a variant of Bayesian Optimization. Furthermore, it discusses the target tracking model used. Selected baseline and benchmark algorithms, inspired by the aforementioned related work, are also detailed in Chapter 2. Chapter 3 formally frames the research problem, and then describes the methodology used to solve it. Importantly, it details how the search technique is combined with the target tracking model. Chapter 4 details the algorithms proposed as solutions to the research question. Chapter 5 presents the results of the simulated experiments, alongside a discussion interpreting their implications. Chapter 6 concludes the research and gives direction to potentially fruitful future work.

Chapter 2

Background

2.1 Introduction

This chapter begins with an introduction to Reinforcement Learning [28] which is a topic from which this research draws inspiration (see Section 2.2). Reinforcement Learning is a field of research used for teaching autonomous systems how to act. Note that this research problem is *not* framed as a Reinforcement Learning problem. Reinforcement Learning is only a related topic, and an alternative framing of the problem, hence it is useful to discuss Reinforcement Learning.

Following the discussion of Reinforcement Learning is a background on the search technique to be adopted in this research, namely, Bayesian Optimization (see Section 2.3). Bayesian Optimization is a ubiquitous, and highly efficient search technique [6]. Particularly, Bayesian Optimization is a preferred search technique when there is only a limited search time, as is the case in this research. We attempt to find the target in as few trials as possible, hence the choice of Bayesian Optimization to guide the search.

The discussion of Bayesian Optimization is followed by a background on the chosen target tracking technique (see Section 2.4). The target tracking technique of choice is a Kinematic Motion Model [4], which is also well-established.

Following the discussion of Bayesian Optimization, we discuss Hidden Markov Models (a method for modeling a stochastic process) and the Wonham Filter [12][33]. The aforementioned search technique is only suitable for a stationary target, and the aforementioned target tracking technique is only suitable for tracking a target which has already been found. Hence, one requires a way to combine the search and tracking techniques for use in our unique problem setup. We use elements of the Wonham Filter to address this aspect, and its mechanics are detailed in Section 2.6.

Thereafter, the concept of negative information is discussed in Section 2.7. When a location is searched, but the target is *not* found, then negative information has been observed. This negative information can augment the search and tracking algorithm: it can be used to infer the target's actual location [22].

Finally, we discuss the mechanics of the selected benchmark algorithm, namely, the Dirichlet-Multinomial pseudo-count [32][26]. The benchmark algorithm is inspired by the related research outlined in Chapter 1. The benchmark

algorithm is used to measure the relative performance of the solution proposed by this research, hence its importance.

2.2 Reinforcement Learning

2.2.1 A brief introduction to Reinforcement Learning

Reinforcement Learning (RL) is a popular technique used to learn behaviors in autonomous systems [28]. It is used to teach a learning agent how to act in different situations. When the learning agent can effectively learn how to act in different situations, it is able to act without external guidance. Note that we do not frame the research problem of this thesis as an RL problem. We only discuss RL as it is a related topic and an alternative framing of the problem.

RL uses numerical rewards to teach the learner which actions are beneficial in different situations [28]. Considering a toddler interacting with its home environment, the toddler may burn its hand when it touches a hot stove, or be given a hug when behaving well. In both cases, the child takes an action and receives a negative or positive reward. The reward teaches it not to touch a hot stove, or to behave well, respectively.

An important aspect of RL is that the rewards may often be described as *sparse* [28]. Consider the aforementioned toddler which burns its hand after touching a hot stove. The negative reward does not teach the toddler under what conditions the stove is unsafe to touch. Nor does it teach the toddler *why* the stove is unsafe to touch. Instead, the toddler must infer that its action of touching a hot stove leads to the negative reward. In future, the toddler may still make the mistake of burning its hand on a different hot surface. However, after burning its hand multiple times, it would learn that the action of touching any hot metal surface will yield a negative reward. Resultantly, because the signals received by a typical learning agent may be sparse, RL may be an effective way of teaching a learning agent how to act.

2.2.2 A technical review of Reinforcement Learning

In RL [28] the learning agent exists in an environment. The environment consists of a set of possible states, notated as $\mathbf{S}$.

The learning agent interacts with the environment by taking an action, a, from a fixed set of possible actions $\boldsymbol{\mathcal{A}}$ [28].

At the end of each time t, after an action is taken, the new state of the environment at time $t+1$ is determined by a transition function $\boldsymbol{\mathcal{T}}(\mathbf{S}, \boldsymbol{\mathcal{A}})$. The transition function is generally dependent on the state of the environment, as well as the action taken. At each time t, the transition function is defined as [28]:

$$\mathcal{T}(s_t, a_t) = Pr(s_{t+1}|s_t, a_t).$$

where $s_t \in \mathbf{S}$ is the state of the environment at time t, and $a_t \in \boldsymbol{\mathcal{A}}$ is the action taken at time t.

After taking an action, the learning agent receives a reward which is dependent on the state of the environment before the action is taken. The reward received is determined by a reward function, notated as $\mathbf{R}(\mathbf{S}, \boldsymbol{\mathcal{A}})$ [28].

The aim of an RL algorithm is to maximize the rewards it receives. The current and future expected rewards can be described by a value function. Given a learning agent in a particular state, following a particular policy (defined below), the value function is defined as the total discounted expected rewards over all current and future time steps, and is notated as V [9]. The discount factor used to discount the expected future rewards received is notated $\gamma \in [0, 1]$. The value function can be written as [9]:

$$V_\pi(i) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = i\right),$$

where r_t is the reward received by the agent at time t after taking an action over the same time step, $s_t \in \mathbf{S}$ is the state of the agent at time step t, and π is the particular policy (defined below) used by the agent. The γ weights the relative importance of future rewards against immediate rewards. The value function is the total discounted expected rewards over all current and future time steps for an agent starting in state i, following policy π (defined below).

The policy of the agent governs the actions of the agent when in a particular state. Formally, one may define a policy for an agent as [28]:

$$\pi_{\mathbf{t}} = \pi(\boldsymbol{\mathcal{A}}|\mathbf{S})$$

where π_t is a mapping from the perceived state of the environment onto the agents set of actions at time t [28]. By receiving rewards after taking actions in different states, the learning agent may alter its policy, in an attempt to receive greater rewards when in that state again in future [28].

The reward function and the the value function are the cornerstones of determining an optimal policy function, and the determination of an optimal policy is the primary objective of RL, as it would maximize the rewards received by the RL algorithm.

In RL, the learning problem is typically modeled using Markov Decision Processes (MDPs) or Partially Observable Markov Decision Processes (POMDPs) [27]. An MDP is a method for modeling a stochastic process. It is constructed as a set of states, actions, transition probabilities and rewards. Formally, an MDP is defined by the 4-tuple [27]:

$$\{\mathbf{S}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{T}}, \mathbf{R}\}.$$

where $\mathbf{S}$ is the set of states of the environment, $\boldsymbol{\mathcal{A}}$ is the set of actions available to be taken, $\boldsymbol{\mathcal{T}}$ is the transition function and $\mathbf{R}$ is a reward function.

Closely related to MDPs are Partially Observed Markov Decision Processes (POMDPs). When one cannot directly observe the states of the MDP, or when one cannot reliably observe the state of the environment, then it is called a POMDP [27]. Instead of observing the states of the MDP, one only observes signals related to the states, notated as $\boldsymbol{\sigma}$. Resultantly, a POMDP requires a signal model, which gives the conditional probabilities of observing any signal given the true state of the system, notated as $\boldsymbol{\mathcal{O}} = \mathbf{Pr}(\boldsymbol{\sigma}|\mathbf{S})$.

More formally, let $\boldsymbol{\sigma}$ be the set of signals described above, which are observable from the POMDP. The set of signals are all the possible of observations of the environment which the agent can receive. Let $\boldsymbol{\mathcal{O}}$ be the conditional probabilities for all signals given the true state of the system. The conditional signal

probabilities give the probability of observing each signal given the true state of the system. Then a POMDP is defined by the 6-tuple [27]:

$$\{\mathbf{S}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{T}}, \mathbf{R}, \boldsymbol{\sigma}, \boldsymbol{\mathcal{O}}\}.$$

2.3 Searching with Bayesian Optimization

Bayesian Optimization [6][25] is a ubiquitous search technique which has been shown to be optimal when the search time is limited. The choice of Bayesian Optimization is inspired by related research [19][25], which proves that it is an efficient search technique for finding a stationary target. Here we describe Bayesian Optimization with reference to a task of searching for a stationary target, as in [25].

Bayesian Optimization is a highly efficient technique for finding the maximum of some objective function, using only noisy samples of the function [6]. Prior beliefs are updated after sampling from the function. In [25], it is shown how Bayesian Optimization is applied in a discrete domain.

Central to Bayesian Optimization is an equation known as Bayes' Theorem [6]. After observing a noisy sample from the objective function, the posterior belief is determined by Bayes' Theorem. Formally, let $\mathbf{L}$ represent all the possible locations in the discrete domain, let $l \in \mathbf{L}$ represent a location in the domain, let $\mathbf{Pr}(\mathbf{L})$ be the belief over locations, and let $\boldsymbol{\sigma}$ be the set of signals observable after a search. Also, let $\mathbf{Pr}(\mathbf{L}|\boldsymbol{\sigma})$ be the posterior belief over states given the observed noisy signals. Finally, let $C = \mathbf{Pr}(\boldsymbol{\sigma})$ be a normalizing constant. Then Bayes' Theorem is given by [6][25]:

$$\mathbf{Pr}(\mathbf{L}|\boldsymbol{\sigma}) = \frac{\mathbf{Pr}(\boldsymbol{\sigma}|\mathbf{L})\mathbf{Pr}(\mathbf{L})}{C} \tag{2.1}$$

Bayes' Theorem updates the belief after sampling the objective function. However, Bayesian Optimization requires an acquisition function to determine *where* to sample. A common acquisition function is the Expected Improvement (EI) [6] acquisition function, defined below.

Definition: The EI acquisition function (a_{EI})

Let f be an objective function which we would like to maximize on some domain which consists of the set of locations notated as $\boldsymbol{L}$. At the beginning of any time $t+1$, the EI acquisition function, a_{EI}, chooses the next search (or sample) location by [25][6]:

$$l_{t+1} = \underset{l}{\operatorname{argmax}} \left[\mathbb{E}(max\{0, f(l_{t+1}) - f(l^{max})\})|l_0, \ldots, l_t)\right] \tag{2.2}$$

where f is the objective function, $l_{t+1} \in \boldsymbol{L}$ is the location to be searched at time $t+1$, and $f(l^{max})$ is the highest value of the objective function observed given all previous samples up to and including time t.

The EI acquisition function samples at the location which is expected to give the greatest incremental value of the objective function, over the highest observed value of the objective function given all previous samples. One may use the EI acquisition function (or possibly a different acquisition function) to

determine where to sample, and after sampling and observing a signal, one would update the belief using Bayes' Theorem.

Bayesian Optimization is most effective when the objective function is expensive to evaluate. In such a case, the number of permissible samples, or search trials, would be limited. The fixed number of search trials is typically termed an episode [6]. For a limited number of search trials n, the episode length would be said to be of length n. Hence, Bayesian Optimization is an efficient search technique when the length of an episode is short, or when n is small.

In [25], it is shown how the Bayesian Optimization algorithm is used on a discrete grid domain. In [25], they use Bayesian Optimization to search for a stationary target in a discrete grid domain, and find the Expected Improvement variant to be an optimal solution. In [25], the problem is framed as an RL problem, in which the learning agent searches for the target using Bayesian Optimization, and is given a positive reward for finding the target, and a zero reward otherwise.

The Bayesian Optimization algorithm is a simple yet powerful technique for finding the maximum of an objective function, given a limited number of trials. The acquisition function determines where to search, thereafter the belief is updated using Bayes' Theorem. This technique has been shown to be a highly efficient search technique [6].

2.4 Tracking with Kinematic Motion Models

When a target has already been found, target tracking is a very well-established topic. Typically, Kinematic Motion Models are used to track targets which have already been detected [4]. Such models work by estimating the target's future location as a function of its current location and its kinematics.

Arguably two of the most ubiquitous Kinematic Motion Models are second-order models and third-order models (otherwise known as constant velocity and constant acceleration models, respectively) [4]. As implied by their names, each of these models assumes that the target moves with a constant velocity, or with a constant acceleration, respectively. This research only considers second-order models. Such an approach has been taken in related literature such as [16] and [3]. In [3], they consider a similar problem of target tracking using sparse signals, in a similar domain. They find that one is able to infer a target's trajectory given estimates of the target's last two locations [3].

A second-order Kinematic Motion Model is specified as follows. The target's location and velocity must be estimated over time. It is required that there be a model for the target's movement, which is specified as follows. Given the target's location and velocity at time t, the target's location at time $t+1$ is given as [4]:

$$\mathbf{X}_{t+1}|\mathbf{X}_t = \mathbf{G}\mathbf{X}_t + \omega, \qquad \omega \sim \ N[\mathbf{0}, \mathbf{W}] \tag{2.3}$$

where $\mathbf{X_t}$ is a vector of two elements, in which the first element is the target's location and the second element is the target's velocity, $\mathbf{G}$ is a design matrix which characterizes the motion of the target, and ω is a Gaussian error term with a mean of zero.

The aforementioned equation models the target's actual movement. One also requires a model for the observations $\mathbf{Y_t}$, given that it is assumed that the target is not directly observable. The model linking the observations, $\mathbf{Y_t}$, to the target's actual location is given by [4]:

$$\mathbf{Y}_t = \mathbf{H}\mathbf{X}_t + \nu, \qquad \nu \sim N[\mathbf{0}, \mathbf{V}] \tag{2.4}$$

where $\mathbf{Y_t}$ is a vector of the observed location and velocity, $\mathbf{H}$ is a parameter which links the observations of the target to the target's true location, and ν is the observation noise, which depends on the sensor but is also typically taken to have a Gaussian distribution [25][4].

An illustration of a prediction from such a model with Gaussian observation noise is shown in Figure 2.1.

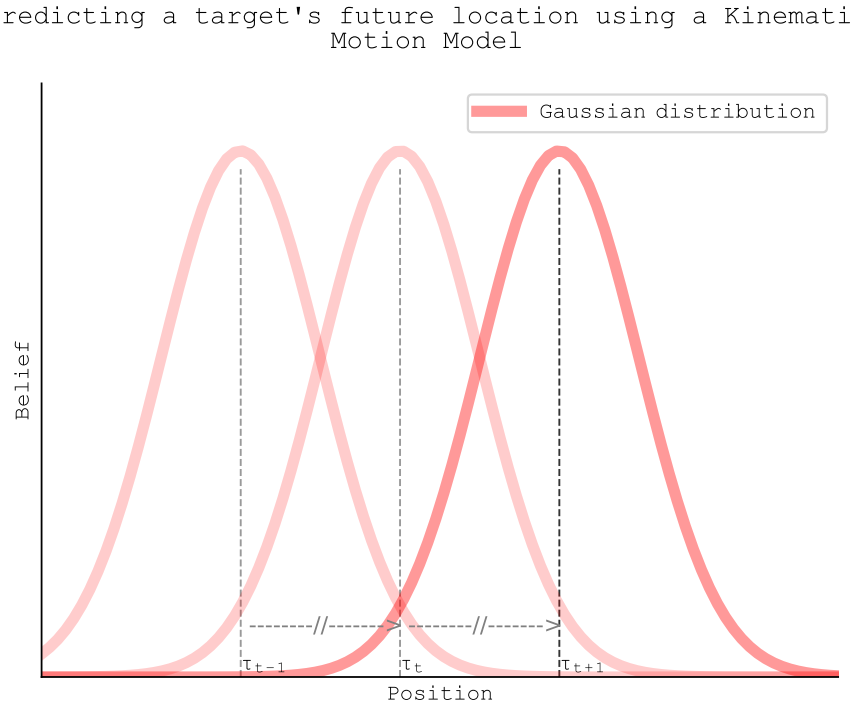


Figure 2.1: Estimates of the target's position, τ, are made at times $t-1$ and t. At the end of time t, the target's predicted position is shown by τ_{t+1}.

To fully detail the model, it is left to specify the parameters of $\mathbf{G}$, and the covariance matrix of the process noise, $\mathbf{W}$. We here describe the model when observations are taken at discrete time points. The design matrix, $\mathbf{G}$, is characterized as [4]:

$$\mathbf{G} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \tag{2.5}$$

where T is the fixed length of time between sampling intervals. When the

observations are taken at discrete times, [4] shows that the covariance matrix of the process noise is then detailed as [4]:

$$\mathbf{W} = \begin{bmatrix} \frac{1}{3}T^3 & \frac{1}{2}T^2 \\ \frac{1}{2}T^2 & T \end{bmatrix} q, \tag{2.6}$$

where q is the process noise intensity. For small q, the motion of the target is expected to be relatively smooth. For the observation model, the design matrix $\mathbf{H}$ and the observation noise covariance matrix $\mathbf{V}$ are specified according to the particular application [4].

Typically, when modeling target motion, movement is assumed to be independent across each coordinate [4][18]. Therefore, for a target moving in a $2D$ domain, two independent Kinematic Motion Models are defined for motion in each coordinate. This is the generally accepted approach, which is also adopted in this research. Moreover, typically the same motion model is used for motion in each coordinate [4][18].

2.5 Hidden Markov Models

Hidden Markov Models (HMMs) [24][31] are related to MDPs and POMDPs. All three of the aforementioned techniques are used to model stochastic processes, in which the state of the process evolves stochastically over time. However, differently to MDPs and POMDPs, HMMs assume that the state of the process evolves independently of any external actions taken [24][31]. HMMs are not typically used to model RL problems.

An HMM models a stochastic process which cannot be directly observed [24][31]. Instead of directly observing the process, one perceives signals related to the process. Based on the signals, one may be able to infer the state of the underlying process.

HMMs are discrete-time stochastic processes. The states of the system are discrete, and the signals observable from the system are discrete. Also required is a signal model which maps the probability of observing a signal to the underlying discrete state of the system [24][31].

Formally, an HMM consists of a set of states, a transition function, a set of signals, and a signal model [24][31]. The evolution of the state of the system is not dependent on any external actions, hence the transition function can be written as $\boldsymbol{\mathcal{T}}(\mathbf{S})$. Now let $\boldsymbol{\sigma}$ be the set of observable signals related to the underlying process. Also, let $\boldsymbol{\mathcal{O}} = \mathbf{Pr}(\boldsymbol{\sigma}|\mathbf{S})$ be the conditional probabilities of observing the signals, given the true state of the system. Then, an HMM is characterized by the 4-tuple [24][31]:

$$\{\mathbf{S}, \boldsymbol{\sigma}, \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}}\}.$$

2.6 The Wonham Filter for Hidden Markov Models

The Wonham Filter [12] is a well-established method for predicting the evolution of a discrete system which is not directly observable. Additionally, when the

observations may have an irregular distribution, then a Wonham Filter may be suitable.

The Wonham Filter draws many similarities to the more popular Kalman Filter [29]. Both follow the same steps, and both were developed around the same time. However, the Kalman Filter is only suitable for continuous domains, and requires the observations to have Gaussian errors.

The Wonham Filter requires a model for the evolution of the underlying system. Additionally, it requires an observation model: a model for the noisy signals which are observable from the system. Given these models the Wonham Filter predicts the future state of the system as being a function of the prior belief over states, the noisy observations of the system, and the model for the evolution of the underlying system.

For a discrete system with N states, the model for the evolution of the state of the underlying system, notated as $\mathbf{A}$, is characterized as [12]:

$$\mathbf{A} = \begin{bmatrix} A(1,1) & A(1,2) & \dots & A(1,N) \\ A(2,1) & A(2,2) & \dots & A(2,N) \\ \vdots & \vdots & \ddots & \vdots \\ A(N,1) & A(N,2) & \dots & A(N,N) \end{bmatrix} \tag{2.7}$$

where $A(j,k)$ is the probability of the system transitioning to state $k \in \{1,\dots,N\}$ given that it is already in state $j \in \{1,\dots,N\}$.

The other required component of the filter is the observation model, notated as $\mathbf{B}$. Assuming a range of M observable signals from system, the observation model is characterized as [12]:

$$\mathbf{B} = \begin{bmatrix} B(1,\sigma^1) & B(1,\sigma^2) & \dots & B(1,\sigma^M) \\ B(2,\sigma^1) & B(2,\sigma^2) & \dots & B(2,\sigma^M) \\ \vdots & \vdots & \ddots & \vdots \\ B(N,\sigma^1) & B(N,\sigma^2) & \dots & B(N,\sigma^M) \end{bmatrix} \tag{2.8}$$

where $B(j,\sigma)$ is the probability of observing a noisy signal of $\sigma \in \{\sigma^1,\dots,\sigma^M\}$ when the true state of the system is $j \in \{1,\dots,N\}$.

Given the above, one can now detail the mechanics of the Wonham Filter. The Wonham Filter follows three steps [12]: (1) the filter takes in the prior belief over the states of the system, as well as the noisy observations of the system; (2) these inputs are used to compute the updated belief over each state using a Bayesian update similar to Equation 2.1; and (3) the next state of the system is predicted as a function of the updated belief, and the state transition model $\mathbf{A}$.

At time t, the filter takes in the prior belief over states, $\mathbf{Pr(S)}_{t^-}$, as well as an observed signal from the system, $\sigma \in \{\sigma^1,\dots,\sigma^M\}$. Let $\odot$ denote the Hadamard product operator [12], which represents entry-wise matrix multiplication. Then the posterior, or filtered, belief over the states of the system is given by [12]:

$$\mathbf{Pr}(\mathbf{S})_{t^+} = \frac{\mathbf{B}(*,\sigma) \odot \mathbf{Pr}(\mathbf{S})_{t^-}}{\mathbf{B}(*,\sigma)^T \mathbf{Pr}(\mathbf{S})_{t^-}} \tag{2.9}$$

where $\sigma \in \{\sigma^1,\dots,\sigma^M\}$ is the noisy observation of the system, $\mathbf{B}(*,\sigma)$ is column σ of the observation model matrix, and $\mathbf{B}(*,\sigma)^T$ is its transpose. The term $\mathbf{B}(*,\sigma)^T \mathbf{Pr(S)}_{t^-}$ can be considered to be a normalizing constant.

The predicted state of the system for time $t+1$, is computed as [12]:

$$\mathbf{Pr(S)}_{(t+1)^-} = \mathbf{A^T Pr(S)}_{t^+} \tag{2.10}$$

where $\mathbf{A^T}$ is the transposed state transition model matrix. At the beginning of time $t+1$, the predicted state vector given in Equation 2.10 is used as the prior belief over the states, which is the input into Equation 2.9. This process is carried out recursively over time and is an optimal method for estimating and predicting the state of a discrete system which cannot be directly observed [12].

2.7 Estimation and tracking using negative information

Estimating the location of a target is considerably challenging without a reliable signal from the target. In a target tracking context, when one searches a location, but does not detect the target, one is said to have observed negative information [22]. Naturally, in such a case, the target's location is assumed to be in any location other than the location which was searched [22].

Formally, assume a discrete spatial domain with N locations, notated as $\mathbf{L} = \{1, \ldots, N\}$. The domain may be as simple as a grid, such as the one illustrated in Figure 1.1. Assume that one can search any one of the N locations, only one of which occupies the target at any given time. Without loss of generality, after searching any location i, if the target is not believed to be in the location, then the target's inferred location is notated as $\mathbf{L^{Neg}}$, where [22]:

$$\mathbf{L^{Neg}} = \mathbf{L} - \{i\}$$

which is the original set $\mathbf{L}$ excluding the location i. After searching location i one is said to have observed negative information.

Clearly, negative information is of greatest value when the set $\mathbf{L^{Neg}}$ is considerably smaller than the set $\mathbf{L}$. To illustrate this, consider the case in which $N-1$ locations in the aforementioned domain are searched, but the target is not believed to be in any of the searched locations. Assume, without loss of generality, that the $N-1$ locations searched include all locations in the domain except location j. Then the set $\mathbf{L^{Neg}}$ is given as:

$$\begin{aligned}
\mathbf{L^{Neg}} &= \mathbf{L} - \{1, 2, \ldots, j-1, j+1, \ldots, N\} \\
&= \mathbf{L} - \{\mathbf{L} - \{j\}\} \\
&= \{j\} \\
&= j
\end{aligned}$$

Hence the target's inferred location is the single location j. This motivates an interest in utilizing negative information.

2.8 The DM pseudo-count

The section discusses the DM pseudo-count which is a technique which has been used to model target movement [26][13][32]. We will use it as a benchmark for

the performance of the algorithms which we develop in Chapter 4, hence its importance. The DM pseudo-count is based on the problem of attempting to estimate a discrete probability distribution given a sample drawn from the distribution (in the form of a count) as well as a prior estimate of the true distribution [8]. Using a discrete probability distribution (and the DM pseudo-count) to model target movement has been explored in [26], [13] and [32].

The DM pseudo-count requires a target's movement to be observed over a long period of time. This is done in order to determine the target's movement policy. Thereafter, the observed movement policy is averaged with the prior belief of the movement policy, to produce a final estimate of how the target moves.

For a domain with N locations, N DM pseudo-counts are maintained. Put differently, each location maintains its own DM pseudo-count. The DM pseudo-count for location $j \in \{1, \ldots, N\}$ is used to empirically estimate the transition probability out of location j.

Intuitively, the empirical transition probability out of any location j could be calculated as a running average of the observed transitions out of location j. More formally, the empirical transition probability between locations $j \in \{1, \ldots, N\}$ and $k \in \{1, \ldots, N\}$, denoted $Pr(\widetilde{l}^{jk})$, may be calculated as [8]:

$$Pr(\widetilde{l}^{jk}) = \frac{n_{jk}}{\sum_{u=1}^{N} n_{ju}}$$

where n_{jk} is the number of times a transition from location j to k was observed, and $\sum_{u=1}^{N} n_{ju}$ is the total number of observed transitions out of location j. The shortcoming of this method is that it does not allow one to account for a prior belief of the transition probability out of location j.

The DM pseudo-count does allow one to account for a prior belief, and it also allows one to give a weighting to the confidence in the prior belief. Furthermore, the DM pseudo-count accounts for the sampling error inherent when estimating the transition probabilities empirically [8].

Formally, for any time t, let $Pr(\widetilde{l}^{jk})_{t^-}$ be the prior belief of the target's transition probability between locations j and k. Let $\alpha > 0$ be a weighting given to the prior belief of the transition probability between locations j and k. Typically, α may be taken to be 1, or the number of observations which were used to determine the prior belief [8]. The DM pseudo-count's estimate of the transition probability from location j to k is given by [8]:

$$Pr(\widetilde{l}^{jk})_{t^+} = \frac{\alpha . Pr(\widetilde{l}^{jk})_{t^-} + n_{jk}}{\alpha + \sum_{u=1}^{N} n_{ju}}. \tag{2.11}$$

The shortcomings of the DM pseudo-count are that it requires a target to be observed for a long time before it is effective. Additionally, it requires the target to have habitual movement patterns. However, if these two conditions are fulfilled, then the DM pseudo-count can be highly effective [8]. By regularly observing the target move between any two distinct locations j and k, one can update the DM pseudo-count for location j so that the next time the target is observed to be in location j, it will accurately be predicted to move to location k. Although the DM pseudo-count assumes that the observations of the target's transitions are certain, in Chapter 3 we describe how we apply the DM

pseudo-count when the target's transitions between locations are not directly observable.

2.9 Conclusion

This chapter detailed the related research which this thesis contributes to. It detailed popular search and tracking algorithms, as well as a technique for modeling an HMM. Furthermore, it discussed the topic of negative information.

In this Chapter we also discussed the mechanics of the DM pseudo-count, which will be used as a benchmark algorithm to compare the performance of our proposed solution. Note that one may initially expect Sequential Bayesian Optimization [20] (where Sequential Bayesian Optimization may be described as a sequential decision making problem framed in terms of a POMDP) to also be used as a benchmark algorithm. However, to the knowledge of the authors of this thesis, Sequential Bayesian Optimization has not yet been directly proven to be a successful solution to the problem setup of this research, hence it may not be applied as a *benchmark* algorithm. Instead, an application of Sequential Bayesian Optimization to this problem setup would be a *contribution* to the related literature. Note that Sequential Bayesian Optimization has been applied to somewhat similar problems, but the problems have different assumptions and therefore are not directly comparable. For example, in [19], Sequential Bayesian Optimization is applied in a continuous domain, to guide a robot equipped with a sparse sensor in its search for a moving target. However, the assumptions of [19] differ in that the robot considered in [19] is able to search whilst it moves around in the domain, whereas we will only consider the case in which the target tracking agent cannot search whilst it moves.

The next chapter details the theoretical framing of the problem. Moreover, it discusses the conceptual approach to the proposed solution.

Chapter 3

Theoretical framing of the problem and solution

3.1 Introduction

This chapter outlines the theoretical framing of the problem, and the conceptual approach to the solution. The first section (see Section 3.2) gives a concrete formulation of the problem setup. It gives a detailed description of the domain under consideration.

The chapter concludes by detailing the framing of the solution to the problem (see Section 3.3). An explanation is given on how the drone's learning algorithm estimates the target's position, infers its trajectory, and then attempts to track it. We reiterate that in this thesis we use the example of a drone to motivate for a more general target search and tracking (with sparse signals) problem.

3.2 Framing of the problem

3.2.1 Modeling the domain

The problem is considered over a discrete space. The area in which the target moves is modeled as an evenly spaced 2D grid, and only one grid cell may be occupied by the target at any given time. Each grid cell is termed a location, or cell, or state. Note that here the term *state* should not be confused with its meaning in an RL context as described in Chapter 2.

The problem is modeled in discrete time. In discrete time we consider two time units, namely, single time steps (also called trials) and episodes (described in Section 2.3). An episode, E, is a fixed number of time steps, n. Therefore, if the episode length is set to $n = 3$, then at the end of time 6, two full episodes will have been counted.

During the course of each episode, the target remains stationary. At the end of each episode, the target can move between locations in the grid. Additionally, the target can transition through more than one location between episodes. An example of such a target is shown in Figure 3.1.

During the course of each episode, whilst the target is stationary, the drone

attempts to find the target. It does so by flying to a location, lowering itself, and searching the location for the target. This process takes a full time step. Hence, for an episode of length n, the drone will have n search trials during which to find the target, before the target moves to a new location.

Of note, we assume that during each time step, or search trial, the drone can search *any* location in the domain. In the following time step, it can then fly to *any* other location in the domain, and search the location. A more realistic assumption may be to include a cost function for the distance between successive locations searched. This is the approach taken in much of the related research [20]. However, we do not consider a cost function for the distance between successive locations searched as we are only using the drone setup as an example. The algorithm developed as a solution is meant to have broader applications to general target search and tracking. The use of a cost function for the distance between successive locations searched would be relevant if we were developing an algorithm only meant for drones.

We reiterate that the algorithm developed in this research is not solely meant for use in guiding a drone. It is meant to be a general contribution to the target search and tracking literature. As discussed in Chapter 1, the problem setup is also relevant for use in guiding other search agents, such as a Wireless Sensor Network (WSN), to find and track a target. In such a case, a cost function may not be particularly relevant as the WSN may not incur a cost for the distance between the sensors which it activates. Additionally, the assumption that any location can be searched during each trial is more realistic if the search agent is a WSN as described in Chapter 1. Nevertheless, we continue the discussion using the example of a drone.

When the drone searches a location, its sensor gives a noisy signal which indicates whether the target is nearby. Naturally, the signal is strongest when the target is in the same location being searched by the drone. However, if the target is in a nearby location, the sensor may still detect a weak signal of the target.

Also recall from Section 2.2.1 that the drone has a sensor which only detects sparse signals. Therefore, if the target is nearby, the sensor would only be able to detect that a target is in a nearby location, but would not be able to determine where, exactly, the target is. Figure 3.1 fully illustrates how the search and tracking problem is modeled.

3.2.2 An alternative framing of the problem with RL

We draw inspiration from RL in how we frame the problem. As discussed in Section 2.2, RL is a popular technique used to learn behaviors in autonomous systems [28]. RL is a commonly used framework for sequential decision making, and so it is worthwhile to consider a framing of the research problem using RL.

Although we do not explicitly model the problem as an RL problem, this section outlines one variant problem formulation if it were strictly modeled as an RL problem. Later in this section we give a motivation for why we do not explicitly model the problem as an RL problem.

If one were to consider the problem as an RL problem, its framing could be as follows. The environment could be modeled as a POMDP (described in Chapter 2), in which a state of the system is the true location of the target, as

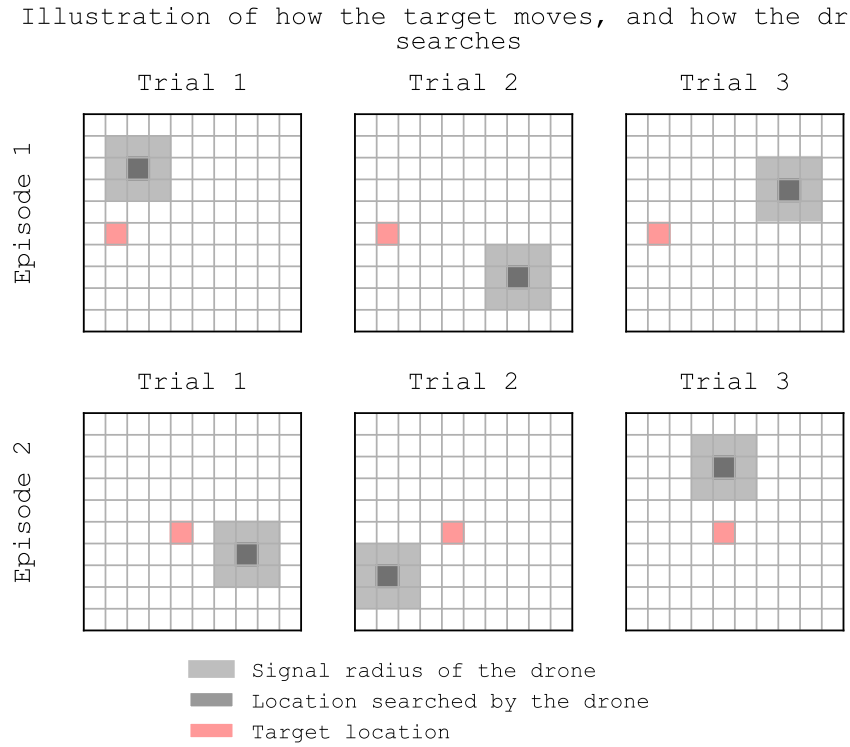


Figure 3.1: Here, the drone (guided by its learning algorithm) is given three search trials per episode, before the target moves. The drone's signal radius can detect a target which is no more than a single cell away. The target transitions through three cells per episode, to the right. In this example, the drone would not have found the target.

well as its true velocity. The reason it could be modeled as a POMDP rather than an MDP, is because the states of the system are not directly observable. Instead, only signals related to the states of the system are observable. This is discussed below, where we describe the states of the environment as well as the signals observable in the environment.

Recall from Section 2.2.2 that a POMDP is defined by the 6-tuple:

$$\{\mathbf{S}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{T}}, \mathbf{R}, \boldsymbol{\sigma}, \boldsymbol{\mathcal{O}}\}.$$

where $\mathbf{S}$ is the set of states of the environment, $\boldsymbol{\mathcal{A}}$ are the set of actions available to the learning agent, $\boldsymbol{\mathcal{T}}$ is the transition function, $\mathbf{R}$ is the reward function, $\boldsymbol{\sigma}$ is the set of observable signals, and $\boldsymbol{\mathcal{O}}$ is the observation model.

Under this formulation, the state of the environment would be the tuple consisting of the true location of the target, as well as the target's true velocity. The drone's location is not included in the tuple because the learning algorithm only needs to learn which location to search, rather than how to get to the particular search location, meaning that the search strategy is independent of the drone's location. However, including the drone in the state tuple would not

change the validity of the solution.

The actions, $\boldsymbol{\mathcal{A}}$, would correspond to the drone flying to a particular location and searching it. For a domain with N grid cells, the drone would have N actions available to choose from [25].

The signals, $\boldsymbol{\sigma}$, would depend on the particular sensors of the tracking agent. In the example discussed in this thesis we would model the signals as sparse, meaning that they would not indicate the exact location of the target. Instead, they would only be able to indicate if a target is nearby or not, with a relative strength indicator. The sensors and signals are discussed in more detail in Section 3.3.4.

The transition function, $\boldsymbol{\mathcal{T}}(\mathbf{S}, \boldsymbol{\mathcal{A}})$, is typically dependent on the state of the environment and the actions taken by the learning agent. Given that the target is assumed to be oblivious of the drone, then, assuming an N location domain, one can model the target's transition between any states to be independent of any action taken. Considering the dependence upon the state of the system, one can see from Equation 2.3 that the future location of a target, whose movement is here modeled using a discrete transition model (discussed later in Section 3.3.6), is dependent on the target's current location as well as its velocity, both of which are included in the state tuple in this formulation of the problem.

The observation model would depend on the sensors of the search agent. As discussed in Chapter 2, the observation model, $\boldsymbol{\mathcal{O}} = \mathbf{Pr}(\boldsymbol{\sigma}|\mathbf{S})$, would be a mapping which gives the conditional probabilities of observing a particular signal from the sensor, given the true location of the target.

The reward model, $\mathbf{R}(\mathbf{S}, \boldsymbol{\mathcal{A}})$, would be dependent on the actions taken by the drone, as well as the state of the environment. The learning agent should be rewarded for finding the target. It would receive a positive reward when it searched the true location of the target. If the drone searched a location which did not have the target in it, then the learning algorithm would receive a zero reward, or possibly a negative reward. A similar reward model is considered in [14] and [25], in which they set their reward function to be dependent on finding the target, and if the learning agent does not find the target it gets a zero reward.

The above formulation gives one overview as to how the problem may be formulated if it were explicitly framed as an RL problem. However, we only draw inspiration from RL in framing our problem, but do not explicitly frame our research problem as an RL problem. Although solutions to such POMDP problems exist [14][21], they are highly computationally intensive, particularly for large domains. Solving the aforementioned problem setup as an RL problem would be computationally intensive [14] because even in a small domain, the belief space is relatively large, as it is a function of the state space, which itself increases exponentially with the number of locations in the domain [14]. For example, in a $1D$ grid domain with three grid cells, a target in the central state of the grid may be moving at either 1 grid cell per time step to the left or to the right, or may not be moving at all (a total of three potential states for one particular location). However, a target in the central state of a grid with five cells in a $1D$ domain may be moving at either 1 or 2 grid cells to the left or right, or may not be moving at all (a total of 5 states for one particular location). Hence, once can see that as the size of the grid increases, the number of potential states of the target increases at a faster rate than the increase in

grid size. Note that in Section 3.3.6 we give a detailed description of how we model the target's movement in this domain.

Hence, we only draw inspiration from RL in framing our solution. We do, however, adopt a variant of the aforementioned reward model, or utility model, for the drone's learning algorithm. Such a reward (or utility) model will be useful for measuring performance in the experiments in Chapter 5. We do not model the problem as a POMDP, hence we do not adopt the aforementioned definition of a state. Instead we will model the movement of the target as an HMM (defined in Section 2.5). A detailed formulation of the solution is given in the following sections.

3.3 Framing of the solution

3.3.1 Modeling the target's motion

Broadly, there are two considerations taken into account when modeling how the target moves. The first major consideration is whether the target is adversarial or oblivious. The second major consideration is whether the limits of the target's velocity are known.

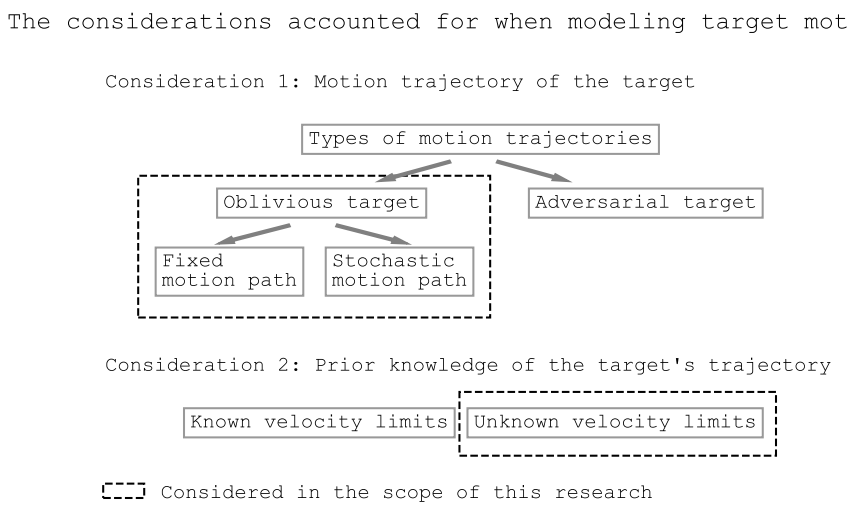


Figure 3.2: A chart showing the assumptions considered when modeling target motion. This research only considers oblivious targets. Additionally, only the assumption of unknown velocity limits is considered.

Under the first consideration, we only study oblivious targets. Finding a solution to the oblivious target problem is important on its own, and is also a step towards solving the adversarial target problem [1], hence we begin by attempting to solve the oblivious target problem. Going on to explore the adversarial target problem is beyond the scope of this research, as adversarial target modeling is an extensive topic on its own. Hence, the scope of this research is limited to oblivious targets.

An oblivious target does not receive any information about the drone's movement. Furthermore, it is implicitly assumed that it is unaware of the drone's presence. In discussing oblivious targets, one can further categorize their movement as either fixed or stochastic, both of which are considered in this research.

The second major consideration taken into account when modeling the target's movement, is whether the limits of its velocity are known. We only consider the case in which the limits are unknown, and the target's velocity must be estimated empirically. Figure 3.2 summarizes the considerations accounted for when modeling target motion.

3.3.2 Searching for the target using Bayesian Optimization

Given the framing of the problem, one can search for the target using the Bayesian Optimization technique described in Section 2.3. The target remains stationary for the length of an episode, during which the drone can search for the target using Bayesian Optimization. Such an approach has been taken in [25], when using a drone to search for a stationary target in a similar discrete domain.

At the beginning of each episode, the drone's learning algorithm may have a prior belief of where the target is. Within the course of the episode, the drone uses its n search trials to search different locations in which it believes the target to be, guided by Bayesian Optimization. At the end of the episode, it would have a final, or posterior, belief of where the target is.

The objective function which we consider is based on the location of the target in the restricted area. Sampling the function is equivalent to the drone searching the area. The noisy signals observed from the samples are the noisy signals observed by the drone's sensor.

We apply Bayesian Optimization by setting the objective function equal to the value of 1 at the location of the target, 0.5 at the locations adjacent to the target, and 0 everywhere else. More concretely, for each location denoted as l, we set the objective function, f, as:

$$f(l) = 1 \quad \text{at the true location of the target} \tag{3.1}$$

$$f(l) = 0.5 \quad \text{at the locations adjacent to the target's true location} \tag{3.2}$$

$$f(l) = 0 \quad \text{everywhere else} \tag{3.3}$$

The objective function is illustrated in Figure 3.3.

Our motivation for choice of objective function is inspired by [25]. In [25], they find Bayesian Optimization to be an efficient search algorithm for guiding a learning agent to find a stationary target in a similar discrete domain. In [25], their choice of objective function is similar to the one described above. Hence, given the similarities of the research problems, we use a similar objective function.

3.3.3 Modeling the target's movement as an HMM

We model the movement of the target using an HMM [24]. Modeling target motion using an HMM has been done in related literature such as [2] and [30].

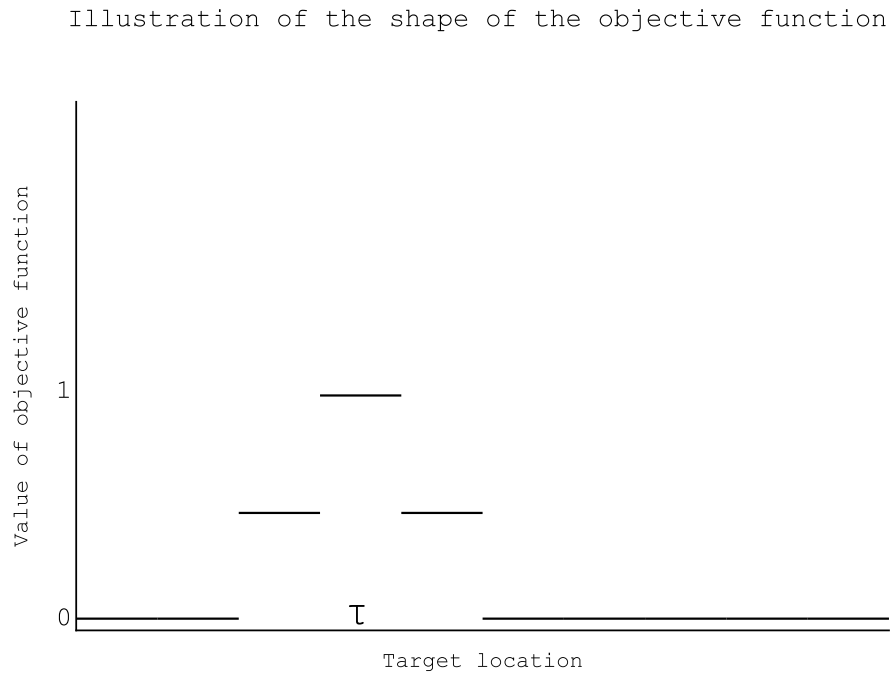


Figure 3.3: The objective function is 1 at the location of the target, 0.5 at the locations directly adjacent to the target's location, and 0 everywhere else. The location of the target is represented by τ.

We model the movement of the target as an HMM because the target is oblivious and its movement is independent of where the drone searches (see Section 3.3.1).

The HMM is defined as follows. The state of the system is defined as the true location of the target in the grid. The true location of the target changes independently of where the drone searches. Furthermore, given that we assume that the target moves with constant velocity with very small, infrequent changes in velocity (see Section 2.4), the Markov property of the HMM still approximately holds. In other words, given a target moving at a constant velocity, the future location of the target is dependent on the current location of the target.

The signals observable, $\boldsymbol{\sigma}$, are as described later in Section 3.3.4. Similarly, the observation model, $\boldsymbol{\mathcal{O}} = \mathbf{Pr}(\boldsymbol{\sigma}|\mathbf{S})$, is as described later in Section 3.3.4. The signals are the sparse and noisy signals observable by the drone's sensor, which indicate whether a target is nearby or not. The observation model gives the conditional probabilities of observing a particular signal, given the true location of the target.

The transition function, $\boldsymbol{\mathcal{T}}(\mathbf{S})$, is dependent on the location of the target, given its velocity (or pseudo-velocity, which is described later in Equation 3.6). After estimating the target's location, then given a velocity estimate of the target, its future location is a function of its current location (shown later in Equation 3.6).

Given that the states of the system cannot be directly observed, one maintains a belief over the states. Note the similarities between an HMM and a

POMDP: both model systems which cannot be directly observed and require a belief to be maintained over the states. The belief over the states of the system is a probability distribution which must sum to unity. An illustration of how the belief may be represented is shown in Figure 3.4.

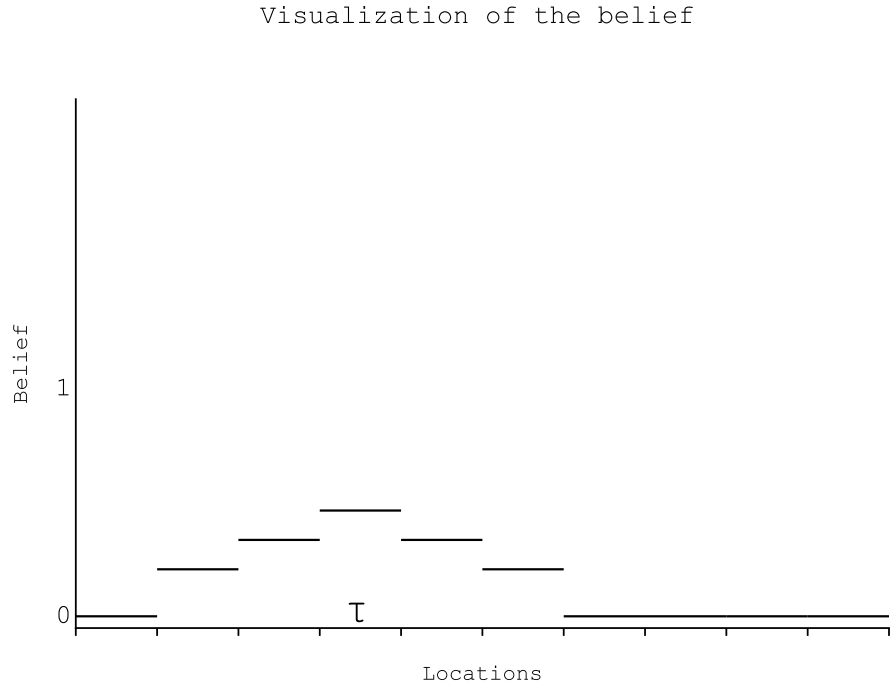


Figure 3.4: Here, the belief is highest at the location of the target. In this case, the target moves in a $1D$ domain. The location of the target is represented by τ. The shape of the belief is dependent on where the drone searches, and the signals it receives from its sensor.

We do note that an alternative way of framing the problem may have been to consider the state as the tuple consisting of the target's location as well as its velocity. However, given that the core problem which we are trying to solve is that of developing an algorithm to guide the *drone's* search and tracking tasks, we adopt the aforementioned simple model for the target's motion and acknowledge that there are other ways of modeling the target's motion which are also valid.

3.3.4 Detecting the target from sparse signals

After searching a location for the target, the drone's sensor returns a signal. Recall that the drone's sensor returns sparse signals, which only indicate whether the target is nearby. The drone's learning algorithm needs a model to translate these signals into meaningful information. Put differently, the drone's learning algorithm must have a model which gives the conditional signal probabilities, given the true location of the target.

The model for the conditional signal probabilities is termed the observation model, and is notated as $\mathbf{B}$. It defines the probability of observing each signal after searching a particular location, conditional on the true location of the target.

More formally, assume there exists a domain with N grid cells. Also assume that the range of signals perceivable by the drone's sensor are given by $\boldsymbol{\sigma} = \{\sigma^1, \ldots, \sigma^M\}$. Here, $\boldsymbol{\sigma}$ is a digital signal with M levels, each of which is a relative strength indicator. Now let k^σ denote that signal σ was observed by the drone after searching location $k \in \{1, \ldots, N\}$. Then the observation model may be characterized as:

$$\mathbf{B} = \begin{bmatrix} B(1,1^{\sigma^1}) & \ldots & B(1,1^{\sigma^M}) & \ldots & \ldots & B(1,N^{\sigma^1}) & \ldots & B(1,N^{\sigma^M}) \\ B(2,1^{\sigma^1}) & \ldots & B(2,1^{\sigma^M}) & \ldots & \ldots & B(2,N^{\sigma^1}) & \ldots & B(2,N^{\sigma^M}) \\ \vdots & \ddots & \vdots & \ddots & \ddots & \vdots & \ddots & \vdots \\ B(N,1^{\sigma^1}) & \ldots & B(N,1^{\sigma^M}) & \ldots & \ldots & B(N,N^{\sigma^1}) & \ldots & B(N,N^{\sigma^M}) \end{bmatrix}$$

where each $B(j, k^\sigma)$ gives the probability of the drone receiving signal σ after searching location $k \in \{1, \ldots, N\}$, when the target is actually in location $j \in \{1, \ldots, N\}$.

Equipped with such a model, the drone's learning algorithm can translate the sensor's sparse signals into meaningful information about the target's location. This model is used for the conditional signal probabilities required by Bayesian Optimization (see Section 2.3).

As a concrete illustration of how $\mathbf{B}$ would be populated, consider the following example. Assume that there is a 10×10 grid domain as illustrated in Figure 3.5. Assume that the drone's sensor receives a signal of: 1 when it searches the same location that the target is in; 0.5 when it searches a location which is vertically, horizontally or diagonally adjacent to the location of the target; and 0 when the target is neither in the location occupied by the drone, nor in any of the adjacent surrounding states. An illustration of such a signal model is shown in Figure 3.5.

Now assume that the sensors are subject to signal noise, which can be characterized by the simple model, $\mathbf{B}'$, given below:

$$\mathbf{B}' = \begin{bmatrix} B'(0,0) & B'(0,0.5) & B'(0,1) \\ B'(0.5,0) & B'(0.5,0.5) & B'(0.5,1) \\ B'(1,0) & B'(1,0.5) & B'(1,1) \end{bmatrix} = \begin{bmatrix} 0.95 & 0.0375 & 0.0125 \\ 0.025 & 0.95 & 0.025 \\ 0.0125 & 0.0375 & 0.95 \end{bmatrix}$$

where $B'(\sigma, \sigma')$ is the probability of observing a noisy signal of σ', when the noiseless signal would be σ. Note that the characterization of the above model is for illustrative purposes, and the choice of numbers used to populate the above model are not meant to represent an actual signal noise model used by drones in practice. We also acknowledge that this is a very simple model, which is only meant to illustrate how the matrix $\mathbf{B}$ would be populated.

Now let $i - j$ denote the grid cell which corresponds to the coordinates of row i and column j of the domain. Given the above signal model and signal noise model, the elements of $\mathbf{B}$ would be populated as shown below:

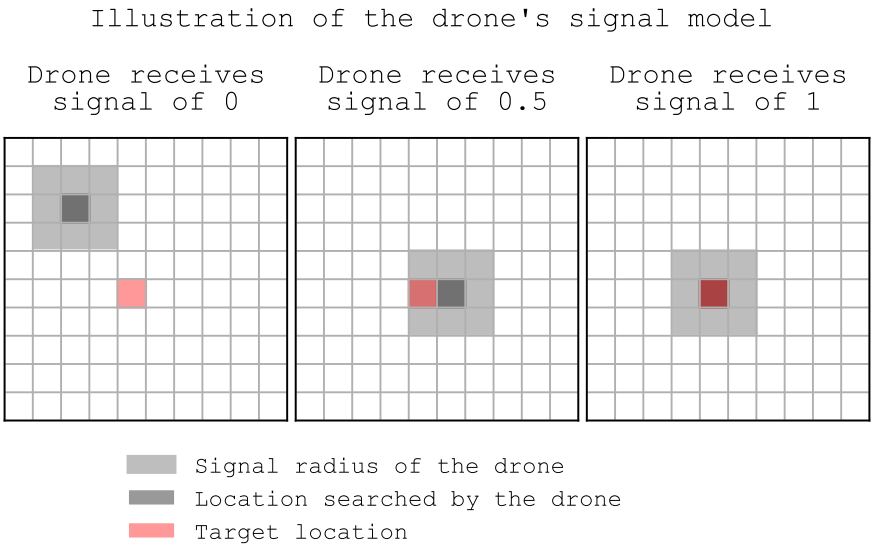


Figure 3.5: In the leftmost frame, the drone searches a location which is far from the target, and receives a signal of 0. The frame in the middle depicts the drone searching a location adjacent to the target, and receiving a signal of 0.5. The frame on the right shows the drone searching the same location occupied by the target, and receiving a signal of 1. In this figure, no signal noise is assumed.

$$\mathbf{B} = \begin{bmatrix} B(1-1,1-1^1) & \dots & B(1-1,1-1^0) & \dots & B(1-1,10-10^1) & \dots & B(1-1,10-10^0) \\ B(1-2,1-1^1) & \dots & B(1-2,1-1^0) & \dots & B(1-2,10-10^1) & \dots & B(1-2,10-10^0) \\ \vdots & \ddots & \vdots & \ddots & \ddots & \ddots & \vdots \\ B(10-9,1-1^1) & \dots & B(10-9,1-1^0) & \dots & B(10-9,10-10^1) & \dots & B(10-9,10-10^0) \\ B(10-10,1-1^1) & \dots & B(10-10,1-1^0) & \dots & B(10-10,10-10^1) & \dots & B(10-10,10-10^0) \end{bmatrix}$$

$$= \begin{bmatrix} 0.95 & \dots & 0.0125 & \dots & 0.0125 & \dots & 0.95 \\ 0.025 & \dots & 0.025 & \dots & 0.0125 & \dots & 0.95 \\ \vdots & \ddots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0.0125 & \dots & 0.95 & \dots & 0.025 & \dots & 0.025 \\ 0.0125 & \dots & 0.95 & \dots & 0.95 & \dots & 0.0125 \end{bmatrix}$$

We will measure performance of the search and tracking algorithm as follows. When the drone's sensor receives a signal of 1 indicating that the target is in the same location as the drone, then the learning algorithm's utility will be 1. Otherwise, its utility will be 0. Related research [14][25] also uses a similar measure of performance, which indicates how successful the drone has been at finding the target. Note that the aforementioned measure of performance is not used in the algorithm which the drone's learning algorithm uses to select locations to search, and is only used to measure the performance of the drone's learning algorithm in the experiments in Chapter 5.

3.3.5 Inferring the target's location from the belief

The drone is equipped with a sparse sensor, which is also subject to signal noise. Resultantly, the drone never directly observes the target. Instead, the drone's learning algorithm maintains a belief of where the target is. The belief itself is a probability distribution over all locations, which must sum to unity.

The belief over each location gives the probability that the target is, or is not, in the particular location. Recall from Chapter 2.7 that when the target is estimated to be in a particular location, that location is said to yield *positive information*. Similarly, when the target is estimated *not* to be in a particular location, that location is classified as having *negative information*. Resultantly, the drone's learning algorithm must have thresholds for the belief, past which it classifies a location as having either positive or negative information. This concept is illustrated in Figure 3.6.

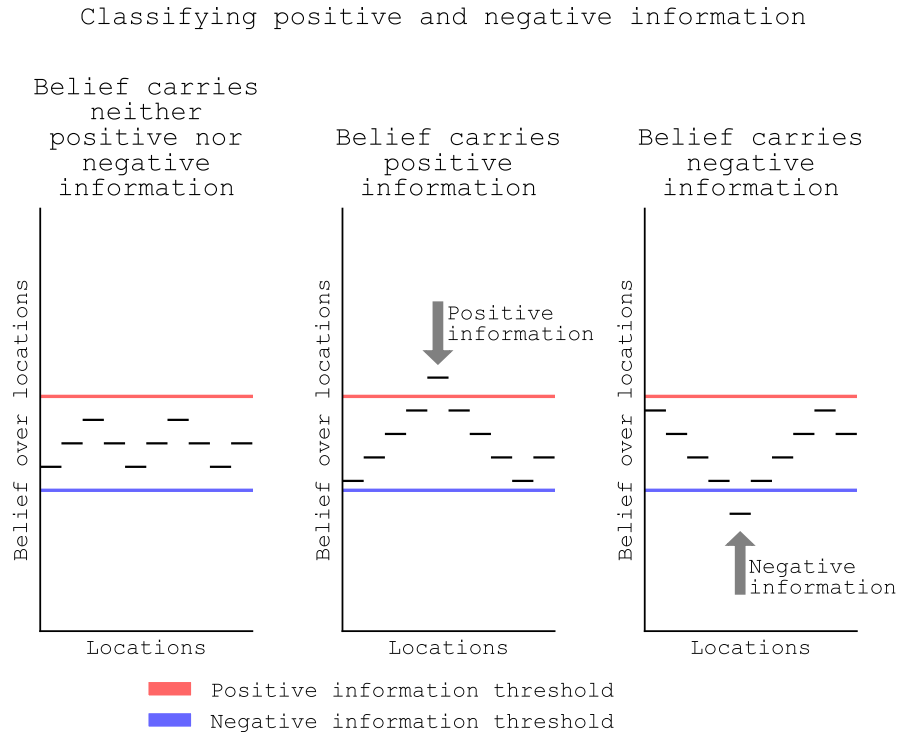


Figure 3.6: The drone's learning algorithm must set a positive and negative threshold, above and below which it classifies a location as having either positive or negative information.

Therefore, when one refers to a location with positive or negative information, it means that the belief of the target being in the location exceeds the positive or negative threshold, respectively. For example, consider a drone which is beginning its search for a target. Without any prior information about the target's location, it may begin its search with a uniform belief over the domain. Now consider a single episode of the search. In the first few trials

of the episode, it may search for the target without success. However, given enough time, it would eventually detect a signal of the target. It would then search the nearby area and continue to get strong signals of the target, which would increase its belief of the target being in that area. As soon as the belief over a location exceeds the positive information threshold, it would classify the location as having positive information. Put differently, it would consider its search successful. Figure 3.7 illustrates such a case.

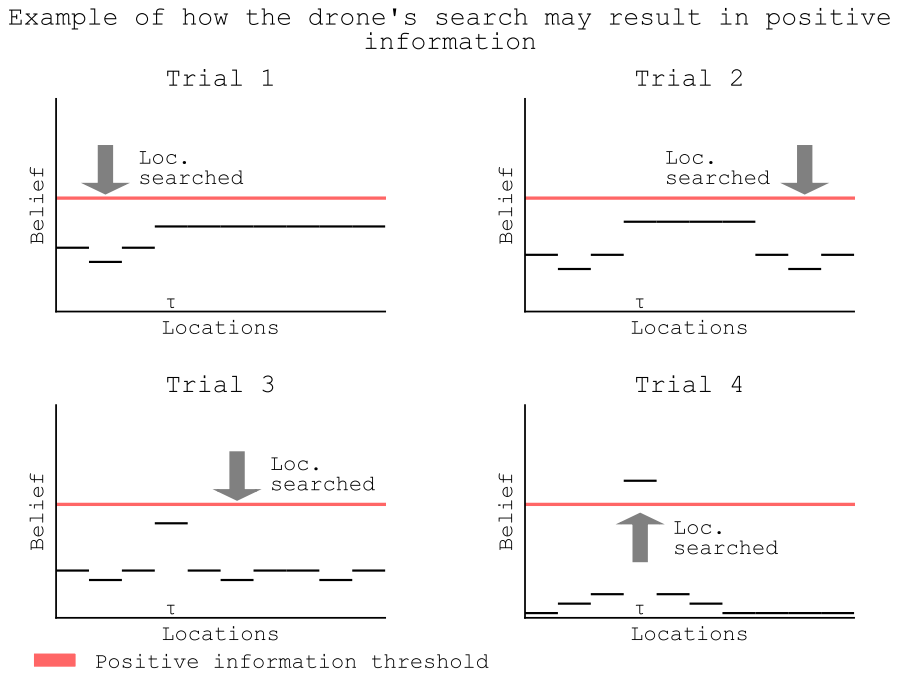


Figure 3.7: The target is represented by τ. After the first three search trials, the drone does not get any signal of the target. However, after its fourth trial, it gets a strong signal of the target. Resultantly, its belief of the target's location crosses the positive information threshold. Therefore, after the drone's fourth search trial, it considers the target to have been found. This figure illustrates a single episode.

The aforementioned method of inferring the target's location, will infer the target's location to be at the location where the belief exceeds the positive information threshold. We do acknowledge that there are other methods for inferring the target's location, which include the maximum a posteriori estimator [23]. The maximum a posteriori method would infer the target to be in the location which had the greatest belief. This is similar to the approach outlined in this section, except the maximum a posteriori estimator does not require the belief to exceed any threshold.

The Viterbi algorithm [15] could be used too, as it is also used to determine the likelihood of an underlying sequence of states in an HMM, given a sequence of observed signals related to the underlying states [15]. It recursively deter-

mines the most likely sequence of states which would have led to the observed sequence of signals. Hence, at the end of each search trial, after observing a signal, one could use the Viterbi algorithm to improve the accuracy of the estimate of the target's location.

However, the Viterbi algorithm only determines the likelihood of an underlying sequence of states given observed signals. Therefore, after using the Viterbi algorithm to improve the accuracy of the location estimate, one could infer the target's location using the maximum a posteriori estimator, which would be more accurate after using the Viterbi algorithm to improve the estimate of the target's location.

Nevertheless we use the aforementioned concept of positive and negative information thresholds as a simple method for inferring the target's location from the belief. Moreover, the use of positive and negative information thresholds is able to handle the way we use negative information (described in Section 4.3.1). Our use of negative information considers all locations in which negative information is observed (even if there may be multiple locations). Hence, the use of thresholds easily allows one to account for multiple locations in which negative information is observed, as any location which exceeds the negative information threshold is deemed to have negative information.

3.3.6 Tracking the target

Defining the target's velocity

In the grid domain considered, the target moves differently to a target moving in a continuous domain. Here, the target transitions between locations, rather than maintaining smooth, continuous motion. Resultantly, one must define how velocity is calculated in this domain.

The Kinematic Motion Model introduced in Section 2.4 assumes that the target moves at a constant velocity (it also assumes that the domain is continuous, hence later in this section we describe how we use concepts from Kinematic Motion Models to characterize a discrete transition model for use in a discrete domain). In this domain, the classic velocity definition is replaced by a concept introduced in this thesis, which is termed pseudo-velocity. Similar to classic velocity, pseudo-velocity is the rate at which the target transitions between grid cells, over some fixed duration.

Formally, consider a target moving in a continuous domain. Classic velocity, v, is defined as [4]:

$$v = \frac{x_f - x_i}{t}$$

where t is the duration over which velocity is measured, and x_f and x_i are the final and initial positions of the target, respectively.

In the discrete grid domain, the pseudo-velocity, v_p, is defined for each coordinate of motion. It is assumed that the locations have an ordinal mapping and that each location has the same dimensions in an evenly spaced grid. The pseudo-velocity is the rate at which a target transitions through grid locations per time period. Here, let x and y be used to notate two coordinates of motion in a $2D$ grid domain. Across each coordinate of motion, pseudo-velocity is then defined as:

$$v_p^x = \frac{x_f - x_i}{t} \tag{3.4}$$

and

$$v_p^y = \frac{y_f - y_i}{t} \tag{3.5}$$

where t represents the duration, or number of episodes, over which the pseudo-velocity is estimated, and x_f, y_f, x_i, and y_i are the final and initial row numbers or column numbers of the target, across the x and y coordinates of motion, respectively. Note that $x_f - x_i$ and $y_f - y_i$ represent the number of rows or columns of the grid which the target moved through, across the respective coordinate of motion. Also, because of the way that the problem is framed, t would generally represent a number of episodes. Recall that the problem formulation considers a target which changes locations between episodes, hence, the time step generally referred to is an episode.

We generally use the term v_p to represent the target's pseudo-velocity across both coordinates of motion. Hence, we loosely define v_p in a $2D$ domain as the tuple:

$$v_p = \{v_p^x, v_p^y\} \tag{3.6}$$

where v_p^x and v_p^y are defined in Equations 3.4 and 3.5. If the domain is only 1 dimensional, then the pseudo-velocity would only be defined across a single dimension. However, when we describe pseudo-velocity in text, we may not always explicitly refer to the aforementioned tuple. For example, looking at Figure 3.8, the target in the leftmost frame would be described as moving with a pseudo-velocity of 1 location per episode to the right. Another way of describing the aforementioned pseudo-velocity would be as 1 location per episode to the right across the x coordinate of motion, and 0 locations per episode across the y coordinate of motion. This of course assumes that the horizontal axis in the figure is defined as the x axis, and the vertical axis is defined as the y axis.

To predict the future location of the target in this domain, one would use its pseudo-velocity. Without consideration for process noise, the predicted location, notated as l_{t+1}, is calculated as:

$$l_{t+1} = l_t + v_p. \tag{3.7}$$

where v_p is as defined in Equation 3.6, and l_t is the location, or coordinates, of the target at time t.

Using the target's velocity to predict its movement

In a continuous domain, the target may be tracked using the Kinematic Motion Model described in Section 2.4. In Appendix A we discuss how we use concepts from the Kinematic Motion Model to inspire our characterization of the target's discrete transition model, or discrete movement model. Please refer to Appendix A for a discussion on how we characterize the target's discrete movement model, or transition model, using concepts from Kinematic Motion Models (which are described in Section 2.4). In this discrete grid domain, the target's discrete movement model, or transition model, is notated as $\mathbf{A}$.

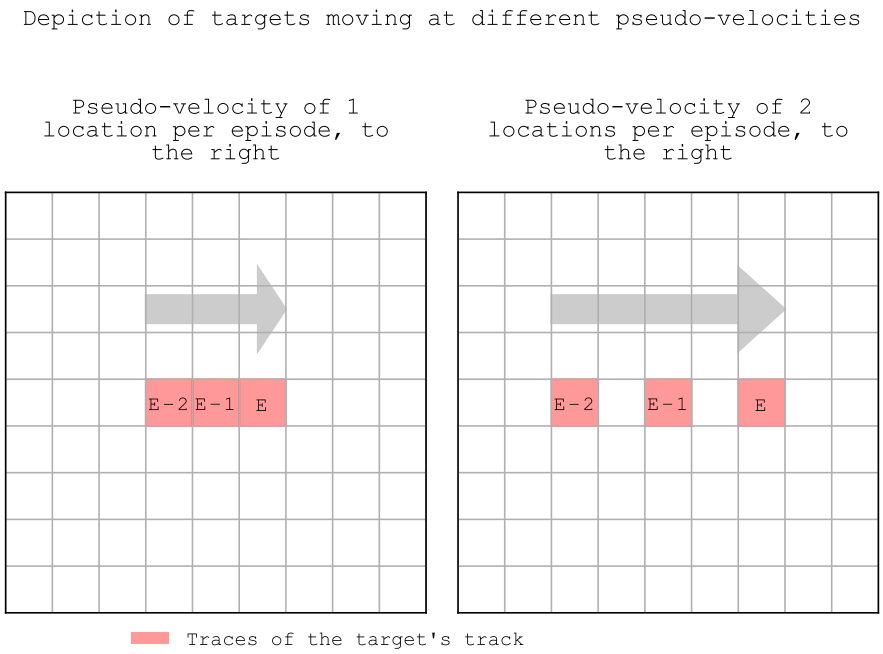


Figure 3.8: Here, each target's pseudo-velocity is estimated over three episodes, notated as $E-2$, $E-1$ and E. The frame on the left shows the traces of a target moving at a pseudo-velocity of 1 location per episode, to the right. The frame on the right illustrates a target moving at a pseudo-velocity of 2 locations per episode, to the right.

Over a continuous domain, the Kinematic Motion Model's prediction of the target's future location is characterized by the target's velocity, as well as Gaussian process noise (see Section 2.4). Similarly, the discrete transition model is characterized by the target's pseudo-velocity, as well as approximately Gaussian process noise.

More concretely, at any time t, for a target in location l_t (here l_t is given in coordinates) with a pseudo-velocity of v_p (defined in Equation 3.6), its predicted location will be:

$$l_{t+1} = l_t + v_p$$

Due to the allowance for process noise, the transition model will assign a high transition probability to location l_{t+1}, low transition probabilities to the locations adjacent to l_{t+1}, and even lower probabilities to the locations which are further away from l_{t+1}. Please refer to Appendix A for a discussion on how we use concepts from Kinematic Motion Models to characterize the target's discrete transition model, or discrete movement model.

The model for the process noise is set before the drone begins its search for the target. When the process noise is low, the probability of transitioning to location $l_{t+1} = l_t + v_p$ will be much higher than the transition probability to any other location. When the process noise is relatively high, then the transition

probabilities out of location l_t will be closer to a uniform distribution.

At each time t, given a pseudo-velocity v_p (defined in Equation 3.6), each row $j \in \{1, \ldots, N\}$ of the transition model is characterized as shown in Figure 3.9. Note that the transition model, $\mathbf{A}$, is the same state transition model to be used in the Wonham Filter (see Equation 2.7). Hence, each row j represents the transition probabilities out of location j, assuming that the target is currently in location j.

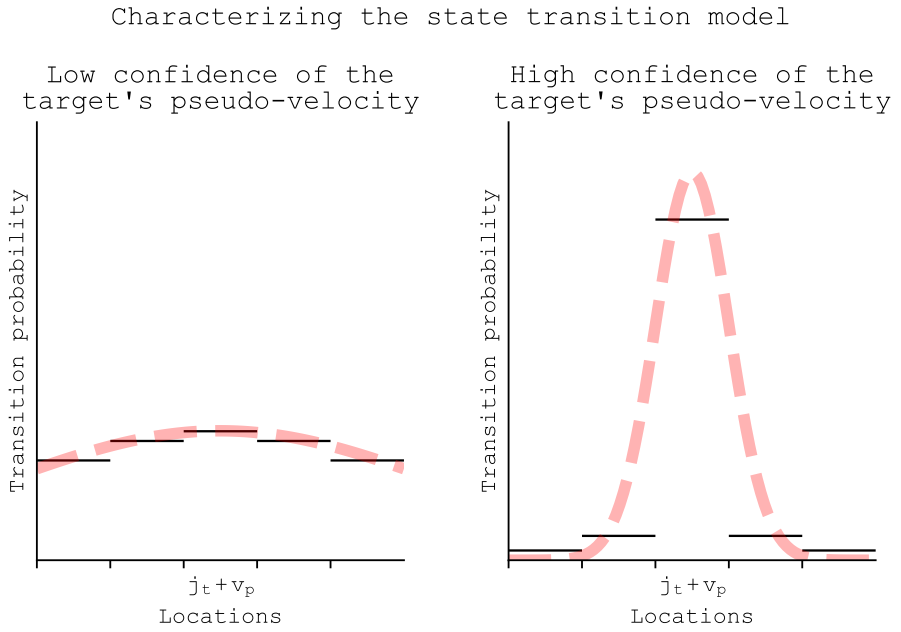


Figure 3.9: A visualization of a single row, j_t, of the transition model. The target's estimated pseudo-velocity is v_p (defined in Equation 3.6). The plots can be interpreted as the probability of transitioning out of location j_t. The plot on the left shows row j_t of transition model when the process noise is high. The plot on the right shows the same row j_t when the process noise is much lower.

We note that using only a single estimate of the target's pseudo-velocity disregards the uncertainty in the pseudo-velocity estimates. If one wished to improve the accuracy of the pseudo-velocity estimates, one may use the standard Viterbi algorithm [15] (see Section 3.3.5). The Viterbi algorithm is used to determine the likelihood of the history of states (locations of the target) in an HMM, given a history of observed signals. By using a longer history of observed signals and estimated locations to estimate the current location of the target, the accuracy of the estimates of the target's location improves. With more accurate locations estimates, the accuracy of the pseudo-velocity estimates will also improve.

Chapter 4

The proposed algorithms

4.1 Introduction

This chapter fully describes two algorithms proposed as solutions to the problem formulation described in the previous chapter. The first algorithm, Algorithm 1, is called Bayesian Search for Target Tracking - Expected Improvement (BSTT-EI). It is the primary algorithm contributed by this thesis. The second algorithm, Algorithm 2, is an extension of the first algorithm, and is called Bayesian Search for Target Tracking - Expected Improvement and Expected Negative Improvement (BSTT-EI-ENI).

The algorithms work as follows. First, Bayesian Optimization is used to search for the target during the course of an episode, whilst the target is stationary. Thereafter, the drone's learning algorithm uses the predict step of the Wonham Filter to predict where the target will move to in the next episode (using the estimates of the target's location and pseudo-velocity). This process is carried out recursively over episodes.

Notably, we only use the predict step of the Wonham Filter, rather than using the whole filter. Put differently, we are only using a specific component of the filter. In [12] it is shown how the predict component of the Wonham Filter is applied in a similar method to how we have used it, in a discrete time and discrete state system.

Notably, the Wonham Filter [12][33] is a well-established method for predicting the evolution of a discrete system which is not directly observable. Additionally, when the observations may have an irregular distribution, then a Wonham Filter may be suitable. Appendix B shows why the distribution of the observation errors may have an irregular distribution in this research.

We now give a more concrete description of the mechanics of the algorithms. Both algorithms follow two steps: (1) search for the target using Bayesian Optimization; and (2) predict the target's next location using the Wonham Filter's state transition model $\mathbf{A}$ (given in Equation 2.7), which itself is characterized either by the target's pseudo-velocity, v_p (given in Equation 3.6), or alternatively it may be characterized by a trajectory of negative information (defined later in Section 4.3).

In the first step, the algorithms differ only by the acquisition function used to search for the target. Recall from Section 2.3 that Bayesian Optimization

requires an acquisition function to guide its search. Algorithm 1 uses the EI acquisition function (see Equation 2.2). Differently, Algorithm 2 uses an acquisition function introduced in this thesis, called the EI-ENI acquisition function. The EI-ENI acquisition function is an augmented version of the EI acquisition function, and is defined later in Section 4.3.

The algorithms differ in how they carry out the second step. Both algorithms initially attempt to use the target's pseudo-velocity to characterize the state transition model of the Wonham Filter. However, when the drone fails to locate the target and resultantly is unable to estimate its pseudo-velocity, then Algorithm 2 uses an alternative method. Without an estimate of the target's pseudo-velocity, Algorithm 1 would have to assume the target to be anywhere in the domain, with equal likelihood. Differently, Algorithm 2 would characterize the state transition model using an estimated trajectory of negative information, which is notated as v_p^- (defined later in Equation 4.1). This idea, which is introduced in this thesis, is fully detailed in Section 4.3. Figure 4.1 summarizes the two proposed algorithms.

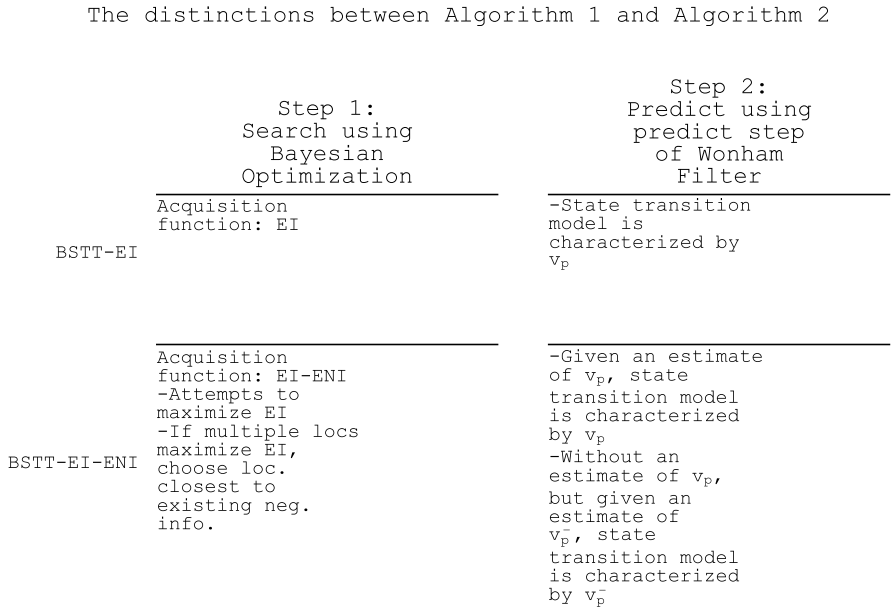


Figure 4.1: Algorithm 1 requires an estimate of the target's pseudo-velocity, v_p (defined in Equation 3.6). Algorithm 2 is an augmented version of Algorithm 1. It also ideally requires an estimate of the the target's pseudo-velocity, without which, it uses an estimate of an observed trajectory of negative information (defined in Section 4.3), which is notated as v_p^- (defined later in Equation 4.1).

Note that one may initially expect the algorithms to follow three steps, rather than two: (1) search using Bayesian Optimization; (2) filter the belief using the filter step of the Wonham Filter; and (3) predict where the target will move to next using the predict step of the Wonham Filter. However, it turns out that

an explicit filtering step using the Wonham Filter is not required. This is shown in Appendix C. The reason that an explicit filtering step is not required by the Wonham Filter is because the Bayesian Optimization step implicitly carries out the filtering step, hence the filtering step which one may have expected from the Wonham Filter, is not required.

4.2 Algorithm 1: BSTT-EI

Algorithm 1 is detailed as follows. Before beginning its search for the target, the drone's learning algorithm initializes its belief with a uniform distribution (line 1 of Algorithm 1). During each episode, the target is searched for using Bayesian Optimization, with an acquisition function guided by EI (lines $4-6$ of Algorithm 1). At the end of each episode, if the drone's learning algorithm has an estimate of the target's pseudo-velocity, it uses its estimate to predict where the target will move to in the next episode (lines $9-11$ of Algorithm 1).

Algorithm 1: The BSTT-EI algorithm

Require: prior belief over locations $\mathbf{Pr(L)}_{t=0}$,
EI acquisition function a_{EI} (see Equation 2.2),
number of episodes in the simulation E,
number of trials per episode n,
observation model $\mathbf{B}$,
state transition model $\mathbf{A}$
1: Initialize the belief with $\mathbf{Pr(L)}_{t=0}$
2: **For** c in $[1, E]$ **do**:
3: **For** k in $[1, n]$ **do**:
4: Use acquisition function a_{EI} to pick search location l
5: Search location l and receive a signal from the sensor
6: Update the belief using Bayes' Theorem, given in Equation 2.1
7: **end for**
8: **If** there is an estimate of v_p (see Equation 3.6) **then:**
9: Forecast the target's future location using the predict step of the
10: Wonham Filter with the state transition
11: model $\mathbf{A}$, characterized by v_p
12: **Else If** there is no estimate of v_p **then:**
13: State transition model $\mathbf{A}$ is uniformly distributed
14: **end for**

4.3 Algorithm 2: BSTT-EI-ENI

As discussed in Section 4.1, this algorithm is an extension of Algorithm 1, but differs from Algorithm 1 in two ways. Firstly, the Bayesian Optimization search for the target is guided by a different acquisition function, namely, an acquisition function introduced in this thesis, called EI-ENI (defined later in Section 4.3.1). Secondly, when the drone's learning algorithm does not have an estimate of the target's pseudo-velocity, then the discrete transition model, $\mathbf{A}$, is characterized by what is defined as an estimated trajectory of negative

information (also defined later in Section 4.3.1). The intuition behind this methodology is explained below.

4.3.1 An alternative approach to using negative information

In this section we introduce a novel approach to using negative information. This approach to using negative information attempts to estimate a trajectory of negative information similarly to how a trajectory of positive information is calculated. The trajectory of negative information is used to reduce the belief of the target being in certain locations. Note that the approach described in this section is based on counting potential paths to arrive at a location, rather than propagating negative information. In other words, we do not attempt to model how free space moves, instead, our methodology is based on counting the number of potential paths which could be taken to arrive at a location.

To illustrate this concept, one must first define what is meant by a trajectory of negative information. If negative information is observed in locations i_{t-1}^- and j_t^-, then we calculate a pseudo-velocity for the negative information, similarly to how a pseudo-velocity for positive information is calculated. Note that we are not attempting to model how free space moves. Instead, we are estimating where the target is less likely to be under the constant pseudo-velocity assumption, which will be explained later in this section. Assume that the x and y coordinates of j_t^- are given as $(x_f)^-$ and $(y_f)^-$, respectively. Also assume that the x and y coordinates of i_{t-1}^- are given as $(x_i)^-$ and $(y_i)^-$, respectively. The pseudo-velocity of negative information, or the trajectory of negative information, is notated as v_p^-, and is defined similarly to the pseudo-velocity given in Equation 3.6. In a $2D$ domain, it is defined as the tuple:

$$v_p^- = \{(v_p^x)^-, (v_p^y)^-\} \tag{4.1}$$

where $(v_p^x)^-$ and $(v_p^y)^-$ are defined as below:

$$(v_p^x)^- = \frac{(x_f)^- - (x_i)^-}{t} \tag{4.2}$$

and

$$(v_p^y)^- = \frac{(y_f)^- - (y_i)^-}{t} \tag{4.3}$$

where t represents the duration, or number of episodes, over which the pseudo-velocity of negative information is calculated, and $(x_f)^-$, $(y_f)^-$, $(x_i)^-$, and $(y_i)^-$ are the final and initial row numbers or column numbers of the locations observed to have negative information, across the x and y coordinates of motion, respectively. Note that we assume that the horizontal and vertical axes in the domain (pictured in Figure 4.2) are termed the x and y coordinates of motion, respectively.

The trajectory of negative information is used to predict which location will have a reduced chance of the target being in it. Assuming a trajectory of negative information, v_p^- (defined in Equation 4.1), we posit that there is a reduced chance of the target being in location k_{t+1}^-, where k_{t+1}^- is calculated as:

$$k_{t+1}^{-} = j_t^{-} + v_p^{-}$$

A visualization of how we calculate a trajectory of negative information, and then use it to predict where the target is less likely to be, is shown in Figure 4.2.

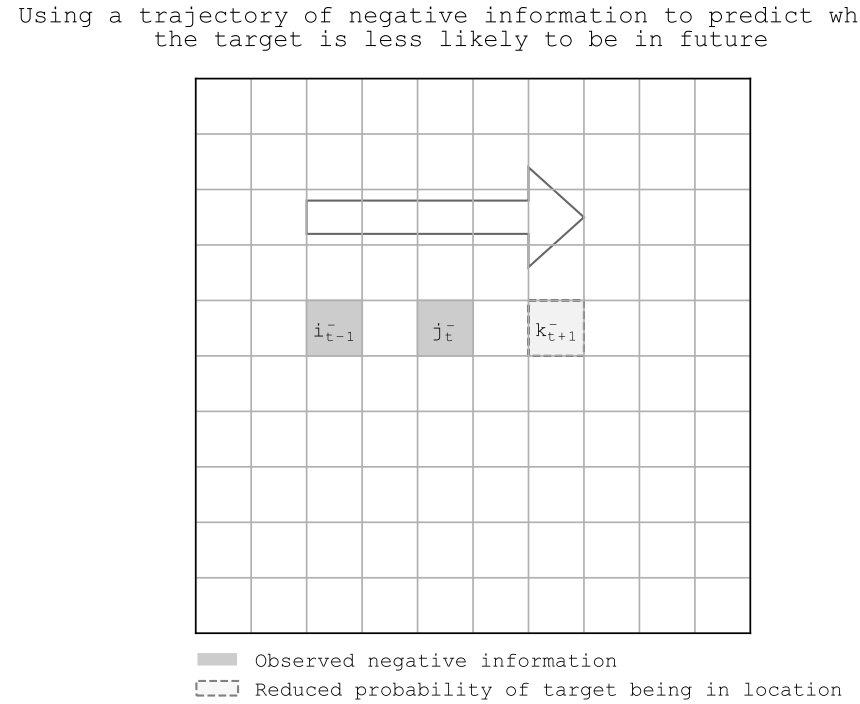


Figure 4.2: Here, each time t is an episode. The drone searches locations i and j at times $t-1$ and t, respectively. It observes negative information, but would still like to use its observations to predict where the target is less likely to be at time $t+1$. Assuming that the target moves at a constant pseudo-velocity, then using the trajectory of negative information, we posit that at time $t+1$ there may be a reduced chance of the target being in location k.

The reason that the target is less likely to be in location k_{t+1}^{-} is based on the target's constant pseudo-velocity assumption. Under the constant pseudo-velocity assumption, there are a limited number of paths, say P, that the target may take to reach location k_{t+1}^{-}. After observing negative information in locations i_{t-1}^{-} and j_t^{-}, one may rule out one of the P paths which the target may have taken to reach location k_{t+1}^{-}. Hence, we posit that under the constant pseudo-velocity assumption, there may be a reduced chance of the target being in location k_{t+1}^{-}.

The motivation for this approach is not based on attempting to model how free space moves. Instead, it is based on approximating the number of potential

paths, or trajectories, which the target may have otherwise taken to arrive at location k_{t+1}^-. As an example, consider a 10×10 grid domain as pictured in Figure 4.3. If the target is assumed to move at a constant pseudo-velocity, then the potential pseudo-velocities it may have taken to arrive at state k_{t+1}^- are as follows:

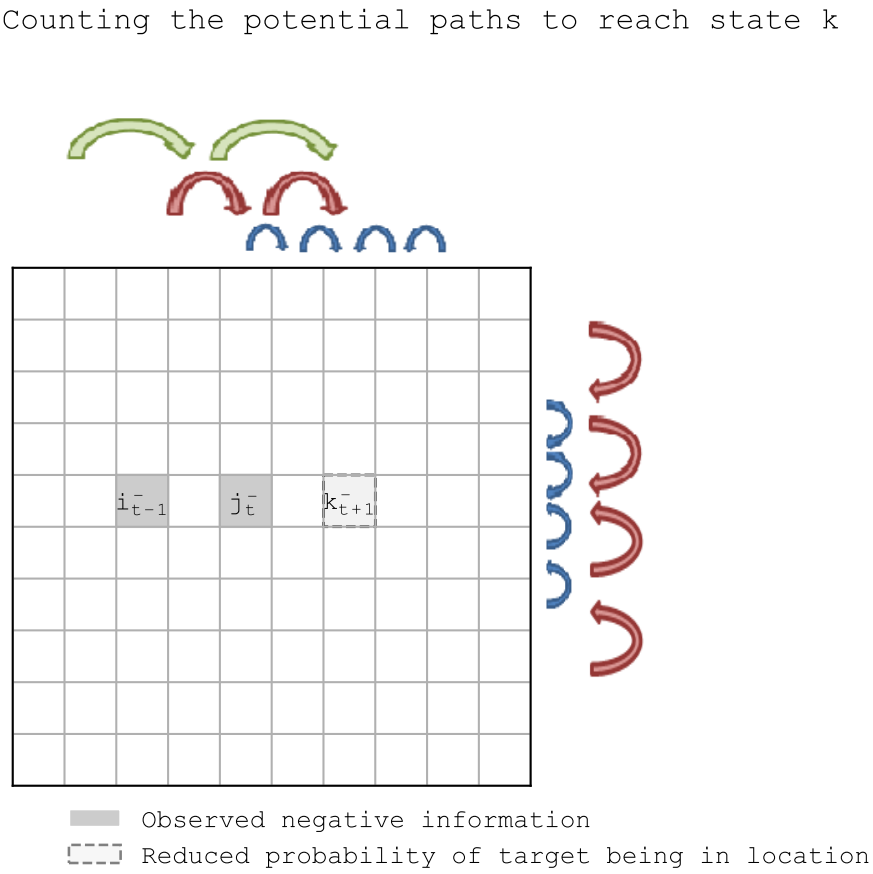


Figure 4.3: This figure shows the same setup as in Figure 4.2. The arrows show the potential paths the target may take when moving at a particular pseudo-velocity across each coordinate of motion. The blue arrows show the potential path (or trajectory) the target may have taken if it was moving at a pseudo-velocity of 1 location per episode across the respective coordinate of motion. The red arrows are used for a pseudo-velocity of 2 locations per episode across the respective coordinate of motion. The green arrows are used for a pseudo-velocity of 3 locations per episode across the respective coordinate of motion.

- Across the horizontal axis (across columns):
  - 0 columns per episode;
  - 1 column per episode to the right;

  - 1 column per episode to the left;
  - 2 columns per episode to the right; and
  - 3 columns per episode to the right.

- Across the vertical axis (across rows):
  - 0 rows per episode;
  - 1 row per episode down;
  - 1 row per episode up;
  - 2 rows per episode down; and
  - 2 rows per episode up.

The example above shows that there are 5 potential pseudo-velocities which the target may have taken across each coordinate of motion, assuming that the target moves with a constant pseudo-velocity. Therefore, given the definition of pseudo-velocity in Equation 3.6, in which it is defined as a tuple across both coordinates, there are $5 \times 5 = 25$ potential paths, or trajectories, which the target may have taken to reach location k^-_{t+1}. However, given that we have observed negative information in states i^-_{t-1} and j^-_t, we can rule out one of the 25 paths. Resultantly, by counting paths, or trajectories, we expect there to be a slightly reduced chance of the target being in location k^-_{t+1}, under the assumption of constant pseudo-velocity.

In this thesis, we do not attempt to find an exact solution to determining the number of paths which the target may have taken to reach state k^-_{t+1}. Instead, we only attempt to explore the concept of using negative information to determine where the target would be less likely to be, hence we reduce the belief over state k^-_{t+1} by a factor of ϵ, which is discussed below.

We do note that in order to calculate the reduction in the probability of the target being in state k^-_{t+1}, the drone would have to calculate the number of potential paths which the agent may have taken to reach state k^-_{t+1}, which may be computationally intensive, particularly for large domains. Nevertheless we explore the idea of using negative information to predict where the target is less likely to be.

A potential scenario in which negative information may be useful may be when there are obstacles in the domain which limit the free space and the potential paths which the target may take to reach state k^-_{t+1}. In such a case, counting the number of potential paths to location k^-_{t+1} may be more plausible, as the number of potential paths would be limited by the obstacles in the domain.

Using a trajectory of negative information to influence the belief of where the target will be, is less effectual than using a trajectory of positive information for the same purpose. Getting actual observations of the target, or positive information, is the most important aspect of estimating the target's future location [22]. Resultantly, we note that although counting the number of paths which the target may have taken to reach state k^-_{t+1} may be used to infer where the target is less likely to be in future, it is still not as effective as actually observing the target. This abstract concept is illustrated in Figure 4.4.

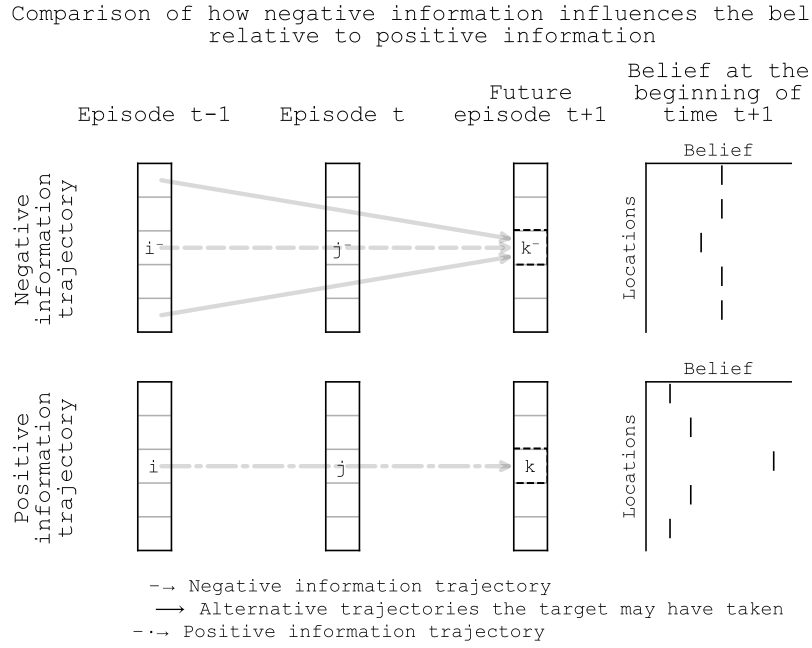


Figure 4.4: This figure shows a 1D grid. The topmost panel shows how the belief may be influenced after observing negative information. Based on the observed negative information in locations i^-_{t-1} and j^-_t, there is a slightly reduced chance of the target being in location k^-_{t+1}. However, this prediction has uncertainty because, assuming constant pseudo-velocity, the target may have taken one of two alternative paths to reach location k^-_{t+1}. Contrastingly, a trajectory of positive information influences the belief much more, as is shown in the lower panel of the figure. Note that the beliefs are uniform prior to episode $t-1$, and minimal noise is assumed. Note that we are not modeling how free space moves, instead we are counting the number of paths which may have been taken to reach k^-_{t+1} under the assumption of constant pseudo-velocity, and simply reducing the belief that the target will be in k^-_{t+1} at time $t+1$.

The small amount by which a trajectory of negative information influences the predicted belief, is a function of ϵ. Figure 4.5 gives a more detailed visualization of how a trajectory of positive information influences the belief relative to a trajectory of negative information. We note that reducing the belief by a factor of ϵ is a simple approximation which we use to augment the belief of where the target will be in future. In this thesis, we use this simple approximation as we are only exploring the concept of using negative information to infer where the target is less likely to be, and it is not the primary contribution of this thesis.

This thesis does not attempt to give an exact solution to define ϵ. However, one notes that intuitively $\epsilon \propto \frac{1}{P}$, where P is the number of paths which the target may have taken to reach location k^-_{t+1}, under the assumption of constant pseudo-velocity. Therefore, in the example shown in Figure 4.3, ϵ would be proportional to $\frac{1}{25}$. We note that calculating P may be computationally intensive

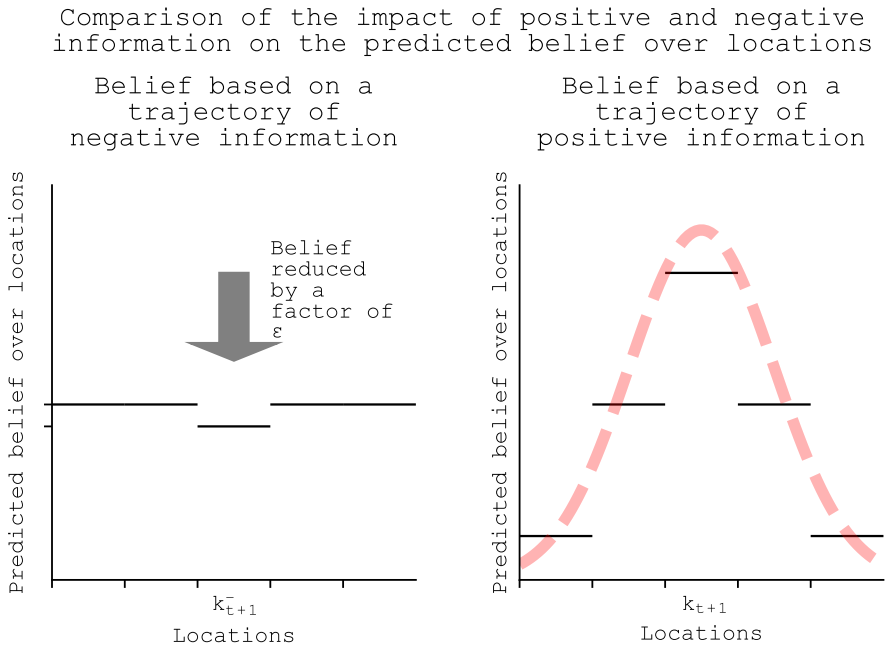


Figure 4.5: This figure shows the beliefs at the beginning of time $t+1$, which were shown in the rightmost panel of Figure 4.4. A trajectory of negative information influences the belief by a small factor which is approximated by a function of ϵ. Whereas a trajectory of positive information has a larger influence on the belief. As discussed in Section 3.3.6, when a trajectory of positive information is observed, the belief is shaped similar to a Gaussian distribution centered at k_{t+1}. We note that reducing the belief by a function of ϵ is only a simple approximation.

on large domains. And we also do not attempt to find an exact solution to determining P. Instead, we only attempt to explore the idea of whether negative information may be useful in estimating where the target is less likely to be in future.

More granularly, we update the belief of where the target will be as follows. For any location k^-_{t+1} which is predicted to have a reduced chance of the target being in it, the belief over that location is reduced by a function of ϵ. If the prior belief over the location is given as $Pr(k^-_{t+1})^-$, then the unnormalized posterior belief after the update, $Pr(k^-_{t+1})^+$, will be given as:

$$Pr(k^-_{t+1})^+ = Pr(k^-_{t+1})^-.(1-\epsilon)$$

where $Pr(k^-_{t+1})^-$ is the prior belief of the target being in location k^-_{t+1}, and $Pr(k^-_{t+1})^+$ is the updated belief of the target being in location k^-_{t+1} after reducing the belief by a factor of ϵ. Note that after updating the belief by a factor of epsilon, the belief over all locations is then normalized so that it sums to 1.

If the drone's learning algorithm observes multiple locations to have negative

information in each episode, and the locations are far apart from each other, then it would have to calculate multiple trajectories of negative information. This would be computationally intensive for the drone's learning algorithm. However, if the multiple locations observed to have negative information are grouped in a set (if all the locations observed to have negative information are adjacent to one another), then the drone's learning algorithm could limit the computational load by only calculating a single trajectory for the set of negative information. This idea is illustrated in Figure 4.6.

In order for the observations of negative information to be grouped in a set (to be adjacent to each other), the drone would have to search adjacent locations as it looks for the target. That way, if it searched locations and observed negative information in the searched locations, then the negative information would be grouped in a set. Resultantly, we must incorporate this idea into the acquisition function for Bayesian Optimization which is used to guide the drone's search. More practically, consider the topmost panel in Figure 4.6. Assume that the belief is initially uniformly distributed. In episode $t-1$ the drone searches two locations for the target, and observes negative information in both searches. In episode t the drone again searches two locations for the target and again observes negative information in both searches. In each episode the drone searches locations which are not adjacent to each other, hence the observed negative information is not grouped in a set. Differently, in the bottom panel of Figure 4.6, the drone searches two locations which are adjacent to each other in each of episodes $t-1$ and t. Resultantly, when it observes negative information, the negative information is grouped in a set. Hence, in the acquisition function used to guide the Bayesian Optimization search in Algorithm 2, we incorporate this idea.

In Algorithm 1 the drone's search used Bayesian Optimization with the EI acquisition function. In Algorithm 2, we use an augmented version of the EI acquisition function, called the EI-ENI acquisition function. The EI-ENI acquisition function is the same as the EI acquisition function, however, when the there is more than 1 location which maximizes EI, then the EI-ENI acquisition function will choose to search the location which is closest to any locations which have been observed to have negative information. That way, if the current search results in negative information, then it would be grouped in a set with any other observed negative information.

Definition: The EI-ENI acquisition function (a_{EI-ENI})

The EI-ENI acquisition function, notated as a_{EI-ENI}, would work as follows. The search would initially be guided by EI as described in Section 2.3 (line 1 *of The EI-ENI acquisition function algorithm). However, whenever there are multiple locations which would maximize EI, then the drone would choose the location which is closest to any existing negative observation (line* 4 *of The EI-ENI acquisition function algorithm). This way, if the search resulted in negative information, the negative information would be grouped in a set. This augmented acquisition function, which we introduce in this thesis, is termed the EI-ENI acquisition function.*

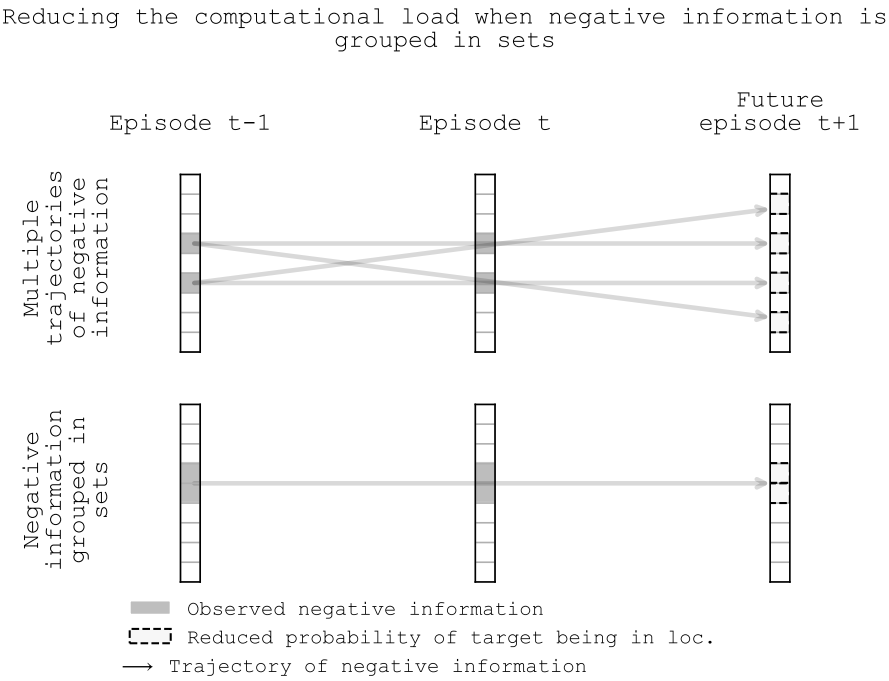


Figure 4.6: In the upper panel, the drone's search results in multiple observations of negative information, which are far apart from each other. Resultantly, the number of trajectories which can be calculated increases factorially with the number of locations which have yielded negative information. However, if the observed negative information is grouped in a set, then the drone's learning algorithm may limit its computational load by only calculating a single trajectory of negative information.

The EI-ENI acquisition function algorithm
1: Using Equation 2.2, determine $\mathbf{L}^+$, the vector of locations which maximize EI
2: **If** length($\mathbf{L}^+$) = 1 **then** chosen search location is l^+
3: **Else If** length($\mathbf{L}^+$) > 1 **then**:
4: From $\mathbf{L}^-$, choose a search location which is adjacent to any locations which have been observed to have negative information

When estimating a trajectory of negative information, the computational load is minimized by only considering the single largest common set of negative information. Let $\mathbf{i^-_{t-1}} = \{\mathbf{i^-_{t-1,1}}, \ldots, \mathbf{i^-_{t-1,n_1}}\}$ and $\mathbf{j^-_t} = \{\mathbf{j^-_{t,1}}, \ldots, \mathbf{j^-_{t,n_2}}\}$ be observed sets of negative information, where n_1 and n_2 represent the number of locations in the respective sets. Note that $|\mathbf{i^-_{t-1}}|$ need not be equal to $|\mathbf{j^-_t}|$. Let $\mathbf{i^-_{t-1,\Gamma}} \subset \mathbf{i^-_{t-1}}$ and $\mathbf{j^-_{t,\Gamma}} \subset \mathbf{j^-_t}$ be the the largest common sets of negative information, where Γ is simply notation used to represent the largest common set of negative information. Being common sets of negative information here means

that $\mathbf{i}^{-}_{\mathbf{t-1},\Gamma}$ and $\mathbf{j}^{-}_{\mathbf{t},\Gamma}$ have the same cardinality, and are exactly the same size and shape. Then there is a reduced probability of the target being in locations $\mathbf{k}^{-}_{\mathbf{t+1},\Gamma}$, which is of the same cardinality, size and shape as sets $\mathbf{i}^{-}_{\mathbf{t-1},\Gamma}$ and $\mathbf{j}^{-}_{\mathbf{t},\Gamma}$. This concept is illustrated in Figure 4.7.

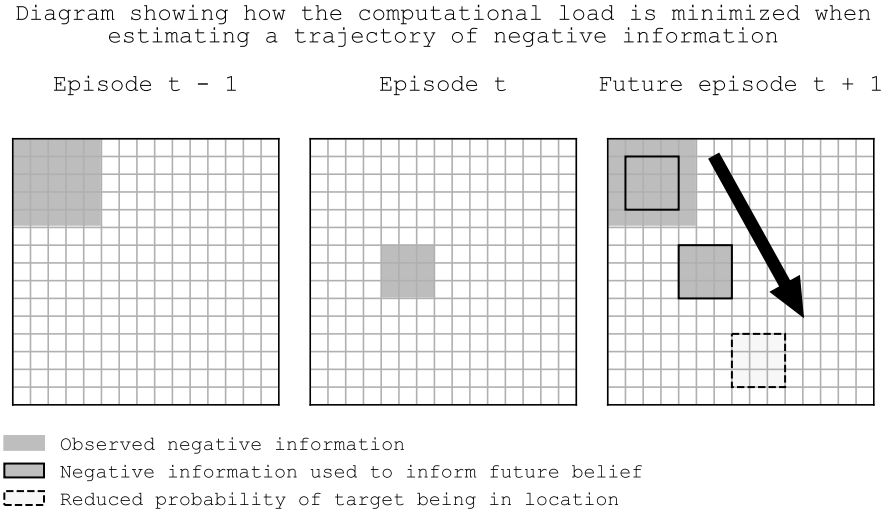


Figure 4.7: Here, negative information is observed during episodes $t-1$ and t. The rightmost frame illustrates how the largest common set of negative information is used to calculate a trajectory of negative information. The frames do not explicitly represent the belief over the domain, and are only meant to show how a trajectory of negative information is calculated.

Given the above, one can now fully describe Algorithm 2, BSTT-EI-ENI. Before the search, the drone's learning algorithm initializes its belief with a uniform distribution (line 1 of Algorithm 2). During each episode, the drone (guided by its learning algorithm) searches for the target using Bayesian Optimization, guided by the EI-ENI acquisition function (lines $4-6$ of Algorithm 2). At the end of each episode, if the drone's learning algorithm has an estimate of the target's pseudo-velocity, it uses that estimate to predict where the target will move to in the next episode (line 9 of Algorithm 2). If the drone's learning algorithm does not have an estimate of the target's pseudo-velocity, but has observed a trajectory of negative information, then it uses the observed trajectory of negative information to predict where the target is less likely to be in the next episode (lines $11-12$ of Algorithm 2). See Algorithm 2 for the full algorithm.

Algorithm 2: The BSTT-EI-ENI algorithm

Require: prior belief over locations $\mathbf{Pr(L)}_{t=0}$,
EI-ENI acquisition function a_{EI-ENI} (see The EI-ENI acquisition function algorithm),
number of episodes in the simulation E,
number of trials per episode n,
observation model $\mathbf{B}$,
state transition model $\mathbf{A}$,
ϵ parameter (see Section 4.3.1)

1: Initialize the belief with $\mathbf{Pr(L)}_{t=0}$
2: **For** c in $[1, E]$ **do**:
3: **For** k in $[1, n]$ **do**:
4: Use acquisition function a_{EI-ENI} to pick search location l
5: Search location l and receive a signal from the sensor
6: Update belief using Bayes' Theorem in Equation 2.1
7: **end for**
8: **If** there is an estimate of v_p (see Equation 3.6) **then:**
9: State transition model, $\mathbf{A}$, is characterized by v_p
10: **Else If** there is an estimate of v_p^- (see Equation 4.1) **then:**
11: State transition model, $\mathbf{A}$, is characterized by v_p^-
12: Belief over $l_{c+1}^- = l_c^- + v_p^-$ is reduced by a factor of ϵ
13: **Else If** there is no estimate of either v_p or v_p^- **then:**
14: State transition model, $\mathbf{A}$, is uniformly distributed
15: **end for**

4.4 The baseline and benchmark algorithms

4.4.1 Baseline algorithm: Standard Bayesian Optimization

A natural baseline is standard Bayesian Optimization with an acquisition function guided by EI. It is a baseline algorithm because it has been shown to be effective at guiding a learning agent equipped with sparse sensors, to find a stationary target in a similar discrete domain [25]. Hence, we use it as a baseline to compare performance against. Standard Bayesian Optimization is detailed in Section 2.3.

This algorithm is exactly the same as the search step of Algorithm 1. At the beginning of each episode, the belief is initialized with a uniform distribution (line 2 of The Bayesian Optimization algorithm). During each episode, the drone (guided by its learning algorithm) searches for the target using standard Bayesian Optimization with the EI acquisition function (lines $4 - 6$ of The Bayesian Optimization algorithm).

The Bayesian Optimization algorithm

Require: prior belief over locations $\mathbf{Pr(L)}_{t=0}$,
EI acquisition function a_{EI} (see Equation 2.2),
number of episodes in the simulation E,
number of trials per episode n
For c in $[1, E]$ **do**:
Initialize the belief over locations with $\mathbf{Pr(L)}_{t=0}$
  For k in $[1, n]$ **do**:
    Use acquisition function a_{EI} to pick a search location l
    Search location l and receive a signal from the sensor
    Update belief using Bayes' Theorem, given in Equation 2.1
  end for
end for

4.4.2 Benchmark algorithm: Augmented DM pseudo-count

When the drone (guided by its learning algorithm) is given a long period of time over which to observe the target, then a suitable benchmark is the Augmented DM pseudo-count. The Augmented DM pseudo-count attempts to empirically estimate the transition probabilities out of each location in the domain. The mechanics of the Augmented DM pseudo-count are detailed in Section 2.8.

Because the classic DM pseudo-count does not have a search mechanism, we search for the target using standard Bayesian Optimization (detailed in Section 4.4), and then use the DM pseudo-count to track the target. Nevertheless, we refer to this combined algorithm as the DM pseudo-count, or the Augmented DM pseudo-count.

Here, the tracking component of the Augmented DM pseudo-count is implemented by maintaining a pseudo-count for each location in the domain. Therefore, for a domain with N locations, N pseudo-counts are maintained. For each location $j \in \{1, \ldots, N\}$, the DM pseudo-count can be represented by:

$$\{Pr(\widetilde{l}^{j1}), \ldots, Pr(\widetilde{l}^{jN})\} \approx \{Pr(l_{t+1} = 1 | l_t = j), \ldots, Pr(l_{t+1} = N | l_t = j)\}$$

where $Pr(\widetilde{l}^{jk})$ represents the empirically estimated transition probability from location j to location k. After observing the target for a long time, and if the target moves in a habitual manner, then the empirical transition probabilities are expected to converge to the true transition probabilities [8].

Given that the estimates of the target's location are uncertain, we update the transition probabilities of the Augmented DM pseudo-count using the positive information threshold described in Section 3.3.5. When the belief of the target being in a particular location exceeds the positive information threshold (when positive information is observed), then one is able to use that estimate to update the Augmented DM pseudo-count. In other words, one can consider the target to have been observed, and resultantly the Augmented DM pseudo-count is updated as described in Section 2.8.

The algorithm for the Augmented DM pseudo-count is given below. Remember that the Augmented DM pseudo-count algorithm described here initially searches for the target using Bayesian Optimization, and then attempts

to predict where the target will move to using the DM pseudo-count described in Chapter 2. However, we refer to this combined algorithm as the DM pseudo-count, or the Augmented DM pseudo-count. Before beginning its search, the drone's learning algorithm initializes its belief with a uniform distribution (line 1 of The Augmented DM pseudo-count algorithm). During each episode, the drone (guided by its learning algorithm) searches for the target using Bayesian Optimization, with the EI acquisition function (lines $4-6$ of The Augmented DM pseudo-count algorithm). If the drone finds the target after two successive episodes (by observing positive information as described in Section 3.3.5), $E-1$ and E, then it updates the Augmented DM pseudo-count for the target's observed location in episode $E-1$ (lines $9-10$ of The Augmented DM pseudo-count algorithm).

The Augmented DM pseudo-count algorithm

Require: prior belief over locations $\mathbf{Pr(L)}_{t=0}$,
EI acquisition function a_{EI} (see Equation 2.2),
number of episodes in the simulation E,
number of trials per episode n,
DM pseudo-count prior confidence parameter α (see Eq. 2.11),
uniform prior DM pseudo-counts for all N locations
1: Initialize the belief with $\mathbf{Pr(L)}_{t=0}$
2: **For** c in $[1, E]$ **do**:
3: **For** k in $[1, n]$ **do**:
4: Use acquisition function a_{EI} to pick search location l
5: Search location l and receive a signal from the sensor
6: Update belief using Bayes' Theorem, given in Eq. 2.1
7: **end for**
8: **If** there is positive information, j_c, **and**
9: positive information in the previous episode, i_{c-1}, **then:**
10: Update DM pseudo-count for location i_{c-1} using Eq. 2.11
11: **Else If** there is positive information, j_c, **then:**
12: Belief for next ep. is initialized with DM pseudo-count for j_c
13: **Else If** there is no positive information, j_c, **then:**
14: Belief for the next ep. is initialized with uniform dist.
15: **end for**

4.5 Conclusion

This chapter gave a detailed description of the algorithms proposed as solutions to the research problem. Algorithm 1 is the primary solution proposed by this thesis. Algorithm 2 is similar to Algorithm 1, but secondarily considers negative information after considering positive information. In the next chapter, the algorithms are rigorously tested.

Chapter 5

Experiments

5.1 Experimental setup

In this chapter, experiments are simulated in order to test the algorithms which have been developed to address the research question. Recall from Chapter 1 that the research aim is to develop an algorithm that can guide a target search and tracking agent to efficiently search for a target, given three primary constraints: sparse signal information, time, and minimal information about how the target moves. These constraints require the drone (we use the motivating example of a drone in this thesis) to be able to find the target in a short space of time, when equipped with a sensor which only detects sparse signals. Moreover, the drone (guided by its learning algorithm) should be able to do this given limited information about how the target moves.

The research hypothesis is that we can develop an algorithm to address the research question, by drawing on the techniques of Bayesian Optimization and Kinematic Motion Models, and by using negative information to augment the drone's search. The algorithms proposed as solutions to the research question (see Chapter 4) are expected to outperform the benchmark and baseline algorithms, namely, the Augmented DM pseudo-count and standard Bayesian Optimization, respectively. However, in the case that the drone has a long time to follow the target and learn its movement patterns, it is expected that the Augmented DM pseudo-count may perform equally as well as the algorithms developed in this thesis.

The algorithms proposed as solutions to the research question are expected to outperform the benchmark and baseline algorithms because they are able to use the information learned from previous searches to predict where the target will be in future. Moreover, they are expected to be able to do so given a limited search history of only two previous episodes. The standard Bayesian Optimization baseline algorithm does not use the information learned from any previous searches to predict where the target will be in future. Contrastingly, the Augmented DM pseudo-count does use the search history to predict where the target will move to. However, the Augmented DM pseudo-count may require a long training period in order to be effective. Hence our expectation of the proposed solution algorithms outperforming the benchmark and baseline algorithms.

The experiments are carried out in a bounded, spatial, 10×10 discrete grid domain. A depiction of the domain is illustrated in Figure 3.5. We do also show the results of experiments on a larger 20×20 domain in Section 5.5.

It is assumed that the target cannot move through the bounds. Therefore, when the target hits the bounds, it loses speed and stops. However, it does not rebound against the boundary. Instead, its pseudo-velocity would be reduced to zero. Given that the target follows a constant pseudo-velocity model, it would maintain its pseudo-velocity of zero, or, with a lower probability, it may accelerate in a different direction. An illustration of how the boundaries affect the target's movement is given in Appendix D.

All experiments are carried out in discrete time. The target remains stationary for the length of an episode, during which the drone (guided by its learning algorithm) is given n trials to localize the target. At the end of the episode, the target may move to another location. Note that the experiments are carried out under the same assumptions as given in Section 3.2, which gives a detailed description of how the target is assumed to move and how the drone's search (guided by its learning algorithm) is modeled.

The drone's signal model uses simple, digital signals. Before considering signal noise: a signal of 1 is received when the target is in the same location searched by the drone; a signal of 0.5 is received when the target is in a location which is vertically, horizontally, or diagonally adjacent to the drone's location; and a signal of 0 is received when the target is neither in the location occupied by the drone, nor in any of the adjacent surrounding locations. Figure 3.5 in Section 3.3.4 illustrates how the drone's signal model works. Note that this is the same signal model described in Section 3.3.4.

In the experiments, moderate signal noise is assumed, the model for which, $\mathbf{B}^{'}$, is given below:

$$\mathbf{B}^{'} = \begin{bmatrix} B^{'}(0,0) & B^{'}(0,0.5) & B^{'}(0,1) \\ B^{'}(0.5,0) & B^{'}(0.5,0.5) & B^{'}(0.5,1) \\ B^{'}(1,0) & B^{'}(1,0.5) & B^{'}(1,1) \end{bmatrix} = \begin{bmatrix} 0.95 & 0.0375 & 0.0125 \\ 0.025 & 0.95 & 0.025 \\ 0.0125 & 0.0375 & 0.95 \end{bmatrix}$$

where $B^{'}(\sigma, \sigma^{'})$ is the probability of observing a noisy signal of $\sigma^{'}$, when the noiseless signal would be σ. Note that this is the same signal noise model described in Section 3.3.4.

Based on the characterization of the signal noise model, the drone's sensor will return an accurate signal with probability 0.95. There is a small chance of returning a signal which is inaccurate, and there is a negligible chance of returning a signal which is two tiers away from the accurate signal. For example, if the drone searches the location in which the target is located, it will most likely receive a signal of 1. It may otherwise receive a signal of 0.5, but very rarely will its sensor indicate that the target is not in any of the nearby surrounds.

We note that a similar discrete measurement model has been used in related research such as [17], which also uses an HMM to detect a target using sparse sensors. Other related research has used a different measurement model which assumes that the signal noise takes a Gaussian form [25]. We here reiterate that the drone example described in this research is used as a motivating example for a more general problem of target search and tracking using sparse signals. Hence, we use the simple measurement model described above for illustrative

purposes, whilst acknowledging that there are other forms that the measurement model may take.

The utility model is based on the noisy signal model, and is the same for each algorithm. A utility of 1 is given when a signal of 1 is observed. Otherwise, no utility is received. An alternative utility model may consider allocating utility when a weak signal of 0.5 is observed. However, here the drone is only considered to be successful if it actually finds the target, hence our choice of utility model.

The thresholds described in Section 3.3 are set as follows. The positive information threshold is set at 0.5 and the negative information threshold is set at 0.01. The rationale behind these choices is as follows. By setting the positive information threshold *below* 0.5, it would be possible to observe multiple locations with a belief exceeding the threshold. This would not be intuitive and would add additional complexity to the problem. Oppositely, if the positive information threshold were set very high, say at 0.99, then the drone's learning algorithm would require a very large number of search trials to form a belief which exceeds the threshold. However, in this research, it is assumed that the drone only has a limited time to find the target.

The negative information threshold is set at $0.01 = \frac{1}{10 \times 10}$, which is the belief over each location under a uniform distribution. A uniform distribution implies that the drone (guided by its learning algorithm) has *no* information about the target's location. Therefore, if the belief crosses the negative information threshold of 0.01, it means that the drone's learning algorithm has *some* information about where the target is *not*. As with the positive information threshold, the negative information threshold could be reduced further, however, the drone (guided by its learning algorithm) would then require a larger number of search trials in order to be able to leverage negative information.

The experiments are carried out on a target which moves along a fixed motion path, as well as a target which has a stochastic motion path. For the fixed motion path, the target is simulated to move in a clockwise circular path inside the domain. A sample trace of this motion path is shown in Appendix E.

For the stochastic motion path, the target's motion is simulated from a constant pseudo-velocity model. The target maintains a constant pseudo-velocity with probability 0.7, and increases or decreases its pseudo-velocity by ± 1 location per episode, with probability 0.15 or probability 0.15, respectively. Sample traces of the stochastic motion path are shown in Appendix E.

We note the inspiration for the aforementioned target model and target behavior. The aforementioned fixed circular motion path is inspired by [19], in which they attempt to use a robot equipped with sparse sensors to find and track a pollutant. In [19] they also test their algorithm on a similar fixed motion path as described above.

For the stochastic motion path, we note related research which also assumes the target follows a constant velocity, with small expected changes in velocity [3][5][7]. In many applications, estimating the trajectory of a target (using only the last two of the target's location estimates) is sufficient for successful target tracking with sparse sensors [3]. Such targets include wildlife and vehicle convoys [3]. We note that the aforementioned target behavior and target models used in the experiments are only one way of testing the experiments, and we do acknowledge that there are other motion models which could be used to model target behavior. In doing so, we also run an experiment for a target following

a linear motion path in Appendix I.

The drone's learning algorithm incorporates a model of the target's movement. As discussed in Chapter 1, this may be possible if one knows what type of target is being tracked. In [20] the authors develop an algorithm to guide a robot to search for and track concentrations of pollution in a similar problem setup as in this thesis. In such a setting, it is plausible that the algorithm has some knowledge of how quickly the pollutant moves through the environment. In [25] an algorithm is developed to guide a drone to search for a poacher in a similar problem setup. If the drone's sole purpose is to find poachers, then it is plausible that the algorithm guiding the drone may have some prior knowledge about how the poachers move.

The benchmark and baseline algorithms are characterized as follows. At the beginning of each experiment, the prior beliefs for the benchmark and baseline algorithms are initialized with a uniform distribution. The Augmented DM pseudo-count algorithm is set with the parameter $\alpha = 1$, where α is as given in Equation 2.11. The parameter choice of $\alpha = 1$ is one of the most typical choices for the parameter when applying a DM pseudo-count [8].

In the first experiment, both Algorithm 1 and Algorithm 2 are tested over 40 episodes. In the second experiment, both Algorithm 1 and Algorithm 2 are tested over 1500 episodes. The third experiment tests the algorithms with varied numbers of search trials allowed in each episode. The fourth experiment tests the algorithms on a larger grid domain. In all experiments it is assumed that the drone's learning algorithm does not know the bounds of the target's pseudo-velocity.

5.2 Experiment 1: short-term performance

This experiment is carried out over 40 episodes. The drone (guided by its learning algorithm) is tasked with detecting the target, and then estimating its pseudo-velocity in order to track it.

The drone's learning algorithm has incorporated the assumption that the target moves with a constant pseudo-velocity, with small, infrequent changes in the pseudo-velocity. The drone's learning algorithm uses the same model as the target: it assumes a constant pseudo-velocity is maintained with probability 0.7, and a change in pseudo-velocity of ± 1 location per episode with probability 0.3. The drone's learning algorithm uses this model when the target follows a stochastic motion path, and it also uses this model when the target follows a fixed motion path.

The experiments are run over $E = 40$ episodes, with $n = 5$ trials per episode. The ϵ parameter (see Section 4.3) of BSTT-EI-ENI is set to $\epsilon = 0.05$.

5.2.1 Experiment 1.1: Fixed motion path

In this experiment, the target moves along a fixed track, as shown in Figure E.1 of Appendix E. The results of the experiment are shown in Figure 5.1.

The results of Figure 5.1 show that both BSTT-EI, and its extension BSTT-EI-ENI, appear to be effective at detecting and tracking the target. However, BSTT-EI-ENI exhibits superior performance to BSTT-EI. BSTT-EI-ENI is quicker to initially detect the target, and it detects a stronger signal of the

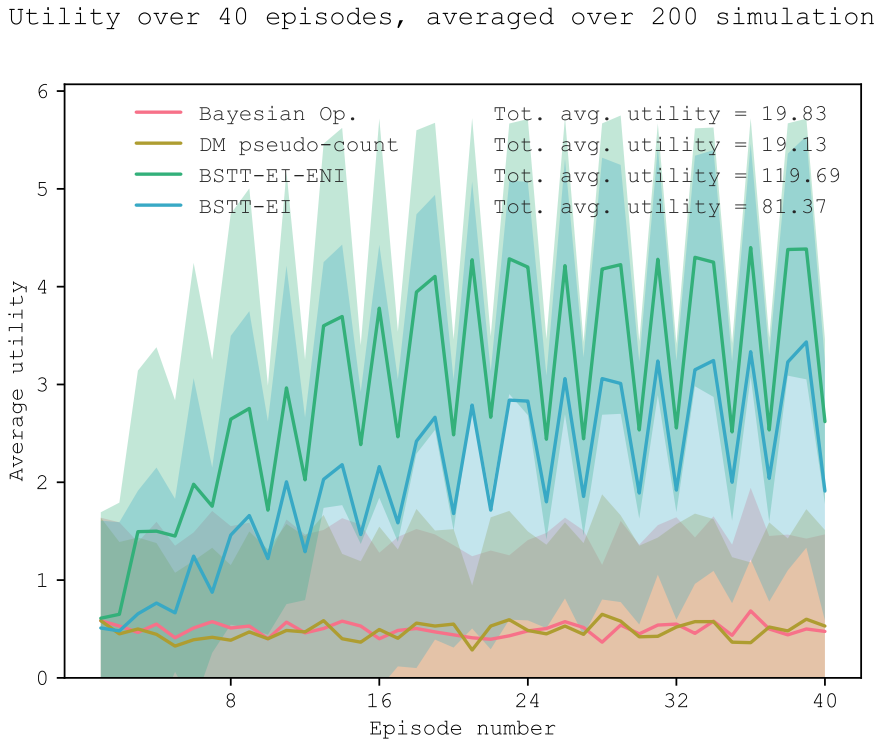


Figure 5.1: The target follows a fixed, circular motion path. Shaded regions represent ±1 standard deviation from the average utility. The drone's learning algorithm has no prior knowledge of the target's pseudo-velocity. The maximum reward achievable in each episode is 5. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

target. Resultantly, it is able to track the target better, which is shown by the steeper gradient of the reward plot.

However, BSTT-EI-ENI only has marginally superior performance than BSTT-EI. Although this experiment shows that BSTT-EI-ENI performs better than BSTT-EI, we must first evaluate the algorithms under different assumptions in order to determine if the results generalize.

Note the jagged shape of the utility plots for BSTT-EI and BSTT-ENI. This is a result of the target following a fixed, circular motion path, and the algorithms using the target's kinematics to predict where it will move to next. In Figure E.1, one can see a sample trace of the target's fixed, circular motion path. Note how the motion path is not exactly circular. Instead, the target moves straight either vertically or horizontally, and then turns and continues to move straight along the perpendicular axis. It repeatedly moves like this, and its track is only approximately circular. After moving along a straight path, the target turns at which point the BSTT-EI and BSTT-EI-ENI algorithms may incorrectly predict the target to continue moving straight given its kinematics.

Resultantly, because the two algorithms momentarily lose track of the target when it makes a turning motion, the utility plots for the two algorithms have a jagged shape.

5.2.2 Experiment 1.2: Stochastic motion path

This experiment tests a target following a stochastic motion path, as described in Section 5.1. All other assumptions remain the same as in Section 5.1. The results of the experiment are shown in Figure 5.2.

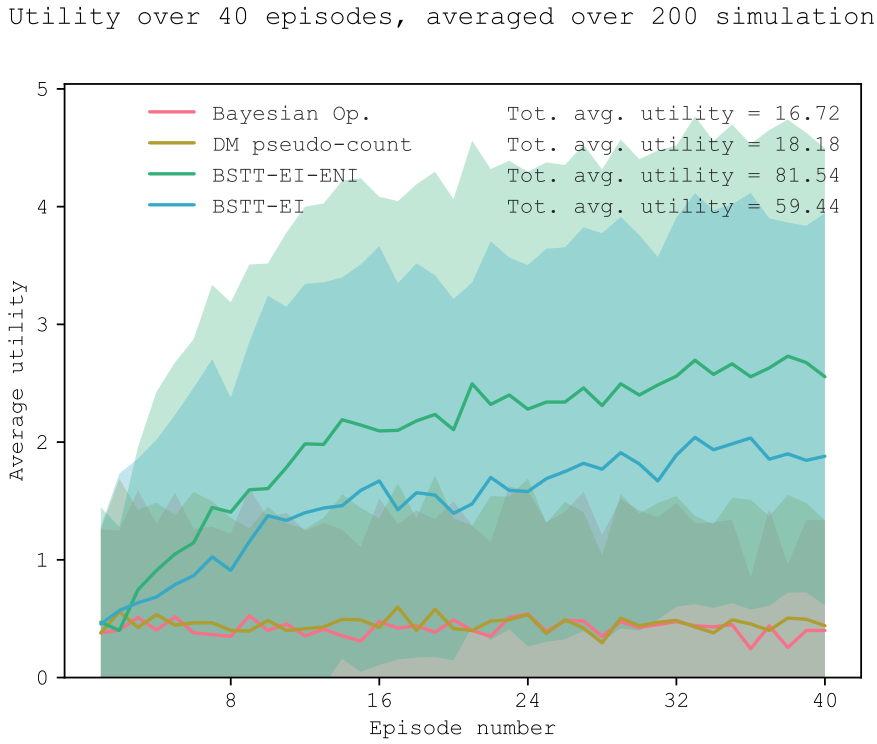


Figure 5.2: The target follows a constant pseudo-velocity model, with small expected changes in pseudo-velocity. Shaded regions represent ±1 standard deviation from the average utility. The drone's learning algorithm has no prior knowledge of the target's pseudo-velocity. The maximum reward achievable in each episode is 5. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

Figure 5.2 shows results similar to Figure 5.1. BSTT-EI and BSTT-EI-ENI are both successful at detecting and tracking the target. BSTT-EI-ENI again exhibits marginally superior performance compared to BSTT-EI, and both BSTT-EI and BSTT-EI-ENI exhibit superior performance to the Augmented DM pseudo-count and Bayesian Optimization.

5.3 Experiment 2: long-term performance

This experiment considers the performance of the algorithms over a long period of time. The experimental setup is exactly the same as in the previous section, Section 5.2, with the exception that the experiment is run over $E = 1500$ episodes. Figures 5.3 and 5.4 show the results of the experiment for a target following a fixed motion path, and a stochastic motion path, respectively.

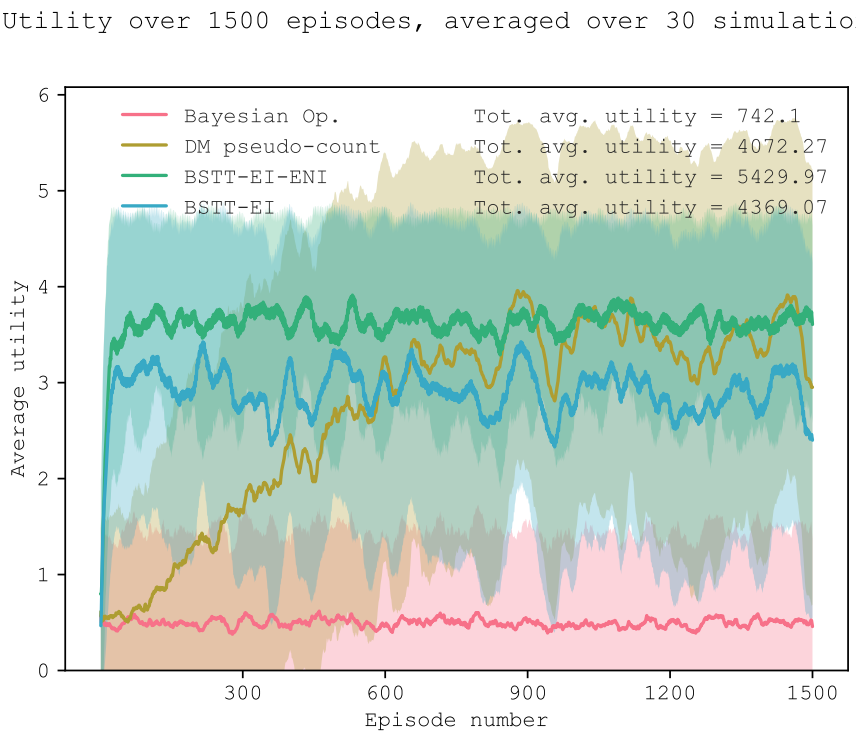


Figure 5.3: We test the algorithms over an extended number of episodes, when the target follows a fixed motion path. Shaded regions represent ± 1 standard deviation from the average utility. The drone's learning algorithm has no prior knowledge of the target's pseudo-velocity. The maximum reward achievable in each episode is 5. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

Considering the long-term performance of the algorithms when the target is following a fixed motion path, the Augmented DM pseudo-count's performance eventually matches that of BSTT-EI-ENI, and also exceeds that of BSTT-EI. Although BSTT-EI-ENI shows the best performance, one would still need to test its effectiveness under different scenarios in order to confirm its effectiveness. Particularly, one would need to test its effectiveness on a larger domain before drawing any conclusions.

However, we do note that the Augmented DM pseudo-count's performance exceeds that of BSTT-EI when the target follows a fixed motion path (see Figure

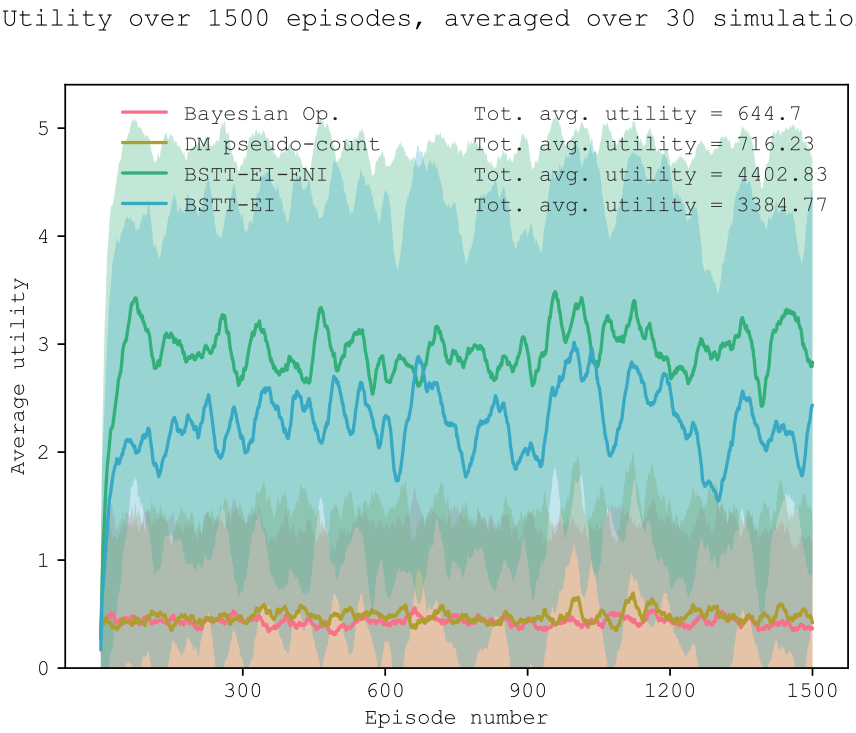


Figure 5.4: We test the algorithms over an extended number of episodes, when the target follows a constant pseudo-velocity model, with small expected changes in pseudo-velocity. Shaded regions represent ± 1 standard deviation from the average utility. The drone's learning algorithm has no prior knowledge of the target's pseudo-velocity. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

5.3). This result implies that BSTT-EI may be a substitute for the Augmented DM pseudo-count when the training period is of insufficient length to apply the Augmented DM pseudo-count. However, over a longer period of time, if the target follows a fixed motion path, then one may use the Augmented DM pseudo-count. Later in Section 5.6 we discuss the results of experiments over a larger domain, before drawing conclusions on the effectiveness of BSTT-EI-ENI.

Notably, Figure 5.3 shows that the Augmented DM pseudo-count exhibits highly variable results. The standard deviation of utility is relatively high, when compared to the other algorithms. This is a result of being given a limited number of search trials in each episode. Also of note is the poor performance of the Augmented DM pseudo-count when the target follows a stochastic motion path, even over a long period of time (see Figure 5.4). This result shows that even over a long period of time, the Augmented DM pseudo-count is ineffective if the target does not follow a habitual movement pattern.

In the following experiments, we test the algorithms under different conditions, before drawing conclusions on their performance.

5.4 Experimental results with varied trial numbers

The experiments in Sections 5.2 and 5.3 are carried out with $n = 5$ trials per episode. Here, the results of the experiments are shown when the drone (guided by its learning algorithm) is given $n = 3$ and $n = 7$ search trials per episode. Figures 5.5, 5.6, 5.7 and 5.8 show the results for the aforementioned experiments, with varied numbers of allowable trials.

Here, the performance charts show that the variability of the utility decreases when the number of permissible trials is increased. This result is intuitive because the drone has more time to find the target, and hence is also able to track the target better.

Moreover, we do note that when the drone (guided by its learning algorithm) is given either less or more search trials to find the target, then the performance of BSTT-EI-ENI more closely matches that of BSTT-EI. The reasoning is as follows. When the drone is given more trials to find the target, then it generally has enough time to find the target. In this case, BSTT-EI-ENI follows the same steps as BSTT-EI. This is because BSTT-EI-ENI only differs from BSTT-EI when it searches for, but does not find the target (and observes negative information). On the other hand, when there are a limited number of trials, then it becomes less increasingly ineffective to attempt to count the number of paths which the target may have taken to reach a particular location.

We also note that when there are a limited number of trials, BSTT-EI still shows relatively better performance than the baseline and benchmark algorithms. Moreover, BSTT-EI may marginally outperform BSTT-EI-ENI (see Figures 5.6 and 5.8). This does question the effectiveness of our method of using negative information. Particularly, it indicates that negative information may only be useful in a limited number of scenarios. On the other hand, BSTT-EI more consistently outperforms the baseline and benchmark algorithms.

5.5 Experimental results on a larger grid

The experiments in Sections 5.2 and 5.3 are carried out on a 10×10 grid domain. Here, the results of the experiments are shown when the grid size is increased to 15×15 and 20×20. Figures 5.9, 5.10, 5.11 and 5.12 show the results for the experiments in Sections 5.2 and 5.3, when the grid size is increased to 15×15 and 20×20. All other assumptions are the same as in Chapter 5. The drone (guided by its learning algorithm) is given $n = 5$ search trials in each episode, and the target follows the exact same motion paths as in Chapter 5.

Here, the performance charts show that BSTT-EI and BSTT-EI-ENI still relatively outperform the baseline and benchmark algorithms. However, one can clearly see that BSTT-EI-ENI does not perform as well as BSTT-EI as the domain gets larger. As the domain becomes larger, our method of using negative information (counting the potential paths which the target may have taken to reach a particular location) becomes less effective. Hence one can see that the way we have used negative information may only be effective in smaller domains.

However, even as the domain gets larger, BSTT-EI still outperforms the

baseline and benchmark algorithms. This result also holds for both the fixed motion path as well as the stochastic motion path. When there are a limited number of trials which ordinarily would be insufficient to find the target, BSTT-EI is sometimes able to track the target given only sparse signals.

5.6 Discussion

The first observation is that standard Bayesian Optimization and the Augmented DM pseudo-count generally fail to find the target in any given episode. This is because the drone has only a limited number of search trials, which are insufficient for the algorithms to detect the target. The exception to this is when the Augmented DM pseudo-count is used to search for a target following a fixed motion path, over a long period of time. In this case, the Augmented DM pseudo-count becomes highly effective for finding and tracking the target. However, even in this isolated case, the Augmented DM pseudo-count still exhibits highly variable utility.

Oppositely, the primary proposed solution algorithm, BSTT-EI proves to be effective at finding and tracking the target given a limited number of search trials. BSTT-EI is quick to initially find the target, and is able to track the target effectively.

Notably, over the short term, BSTT-EI consistently performs better than the benchmark and baseline algorithms, even when the target follows a fixed motion path. Even when the target's movement is location-dependent, BSTT-EI still outperforms the Augmented DM pseudo-count, which accounts for the target's location-dependent movement.

All of the algorithms displayed relatively high variability in the performance charts. This is a natural consequence of the drone being given a small number of trials to find the target in each episode. However, of note is that even at the lower bounds of the variability shown in the performance charts, BSTT-EI generally outperforms the mean utility achieved by the baseline and benchmark algorithms.

In Section 5.4, experiments are carried out with drones which are given more, or less, time to search for the target in each episode. The results in Section 5.4 show that the variability of the drone's performance, measured by the standard deviation of utility, is dependent on the length of the episode. When the drone (guided by its learning algorithm) is given more time to search for the target, the performance of the algorithms is less variable. This is intuitive, as the drone has more time to detect a stronger signal of the target, and then track it.

Moreover, we do note that when the drone (guided by its learning algorithm) is given either less or more search trials to find the target, then BSTT-EI-ENI becomes less effective. When the drone is given lots of time to find the target, then the mechanics of BSTT-EI-ENI are the same as that of BSTT-EI, as both will use the resultant positive information to predict where the target will move to. Oppositely, when the drone is given a smaller number of trials, then it becomes ineffective to count the potential paths which the target may have taken to reach a particular location. This does question the effectiveness of our method of using negative information. Particularly, it indicates that negative information may only be useful in a limited number of scenarios. How-

ever, BSTT-EI still shows relatively better performance than the baseline and benchmark algorithms.

One does note that the performance of the Augmented DM pseudo-count does not increase significantly when the number of permissible trials is increased from $n = 5$ to $n = 7$. The only exception being when the number of episodes is increased to 1500, for a target following a fixed motion path. In this case, the Augmented DM pseudo-count exhibits superior performance to all of the algorithms. This result confirms that the Augmented DM pseudo-count is only effective when the drone has a considerably long time to search for the target in each episode.

In Appendix F, selected experiments are carried out with increased signal noise. The results are intuitive: when the signal noise assumption is increased, all of the algorithms become ineffective because the drone's sensor generally detects erroneous signals which are unreliable.

Appendix G shows the experimental results when the drone's learning algorithm has an inaccurately specified motion model. In other words, the learning algorithm's model of the target's motion is inaccurate. The drone's learning algorithm assumes that the target is highly likely to change its pseudo-velocity, whereas in reality the target generally maintains a constant pseudo-velocity. The results show that BSTT-EI is robust to an inaccurately specified motion model of the target. This holds for as long as the drone's learning algorithm knows that any changes in the target's pseudo-velocity are generally of a small magnitude. In practice, this may be possible. For example, if the drone's learning algorithm knows that the target is a large ship moving in restricted waters, then it may correctly model the ship to have small changes in speed.

In Appendix H, a sample trace of the drone's search is shown, for all the algorithms. Notably, it shows how BSTT-EI (and BSTT-EI-ENI) are quickest to detect the target.

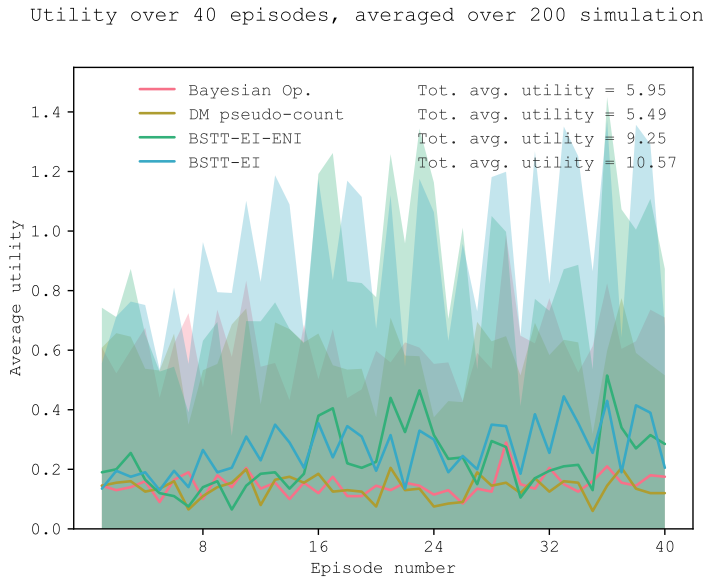
In Section 5.5 we show the results of the experiments of Section 5.2, when carried out on larger grid domains. Particularly, we simulate the experiments on grid sizes 15×15 and 20×20. Notably, we see that BSTT-EI relatively outperforms the baseline and benchmark algorithms in all experiments. Moreover, one can clearly see that BSTT-EI-ENI does not perform as well as BSTT-EI as the domain gets larger. Hence one can see that the way we have used negative information may only be effective in very contrived scenarios. As the domain gets larger, counting the potential paths which the target may have taken to reach a particular location becomes ineffective.

In Appendix J, the results of Section 5.2 are carried out with the positive information threshold (described in Section 3.3.5) is reduced to 0.01. The first thing to note is that when the positive information threshold is reduced to 0.01, then BSTT-EI-ENI has the same mechanics as BSTT-EI. This is because when the positive information threshold is reduced to $\frac{1}{100} = 0.01$, every search trial will result in positive information being observed. Recall from Section 4.3 that BSTT-EI-ENI only differs from BSTT-EI when no positive information is observed after a search.

The second thing to note from Appendix J is that BSTT-EI exhibits superior performance when compared to the experiments in Section 5.2. When the positive information threshold is reduced to 0.01, then the method for inferring the target's location from the belief is the same as the maximum a posteriori

method (see Section 3.3.5), which is a more ubiquitous method for inferring the target's location from the belief. When one considers these results in conjunction with the results of Sections 5.4 and 5.5, which show that BSTT-EI-ENI is only effective in very contrived scenarios, one can see that the use of BSTT-EI with a low positive information threshold (or when the maximum a posteriori method is used to infer the target's location from the belief) gives the best performance.

Experiment 1.1 with varied trials



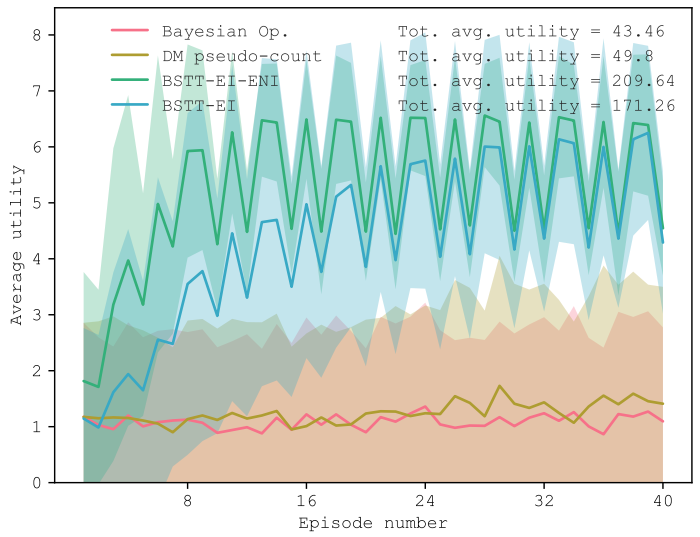


Figure 5.5: Results for Experiment 1.1 with varied trials per episode. In the topmost frame, the drone only has $n = 3$ trials, instead of $n = 5$ trials. In the lower frame, the drone has $n = 7$ trials. The target follows a fixed, circular motion path. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

Experiment 1.2 with varied trials

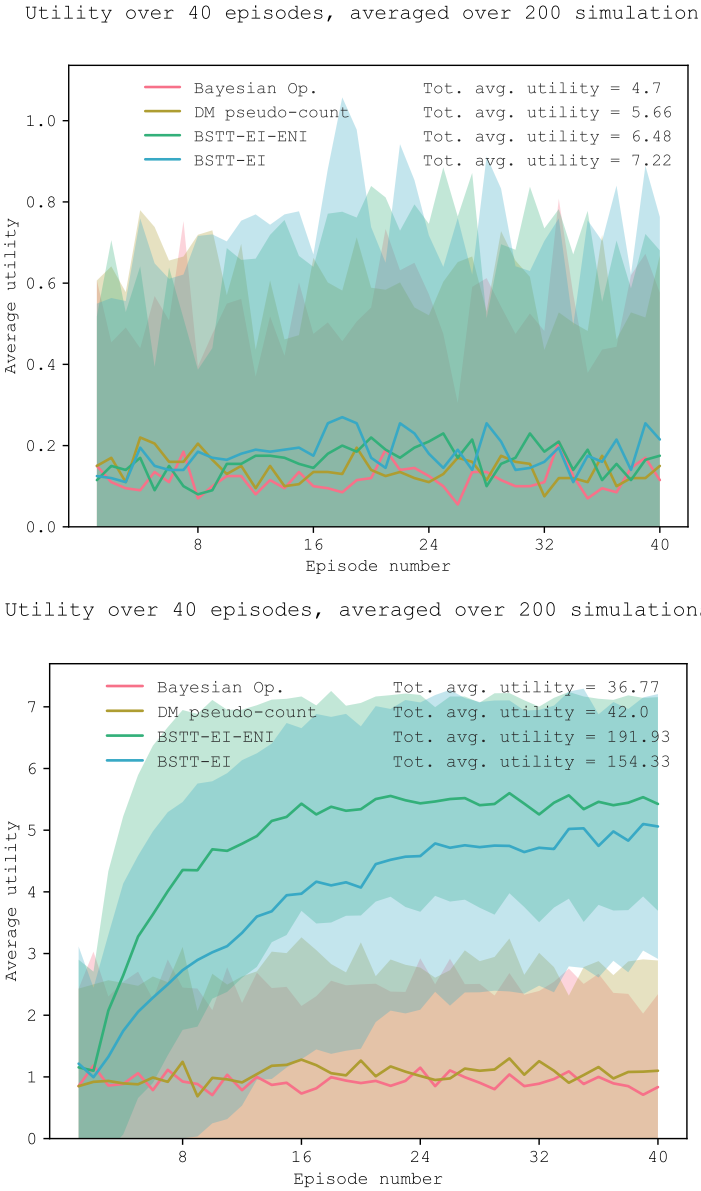


Figure 5.6: Results for Experiment 1.2 with varied trials per episode. In the topmost frame, the drone only has $n = 3$ trials, instead of $n = 5$ trials. In the lower frame, the drone has $n = 7$ trials. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

Experiment 2 (fixed motion path) with varied trials

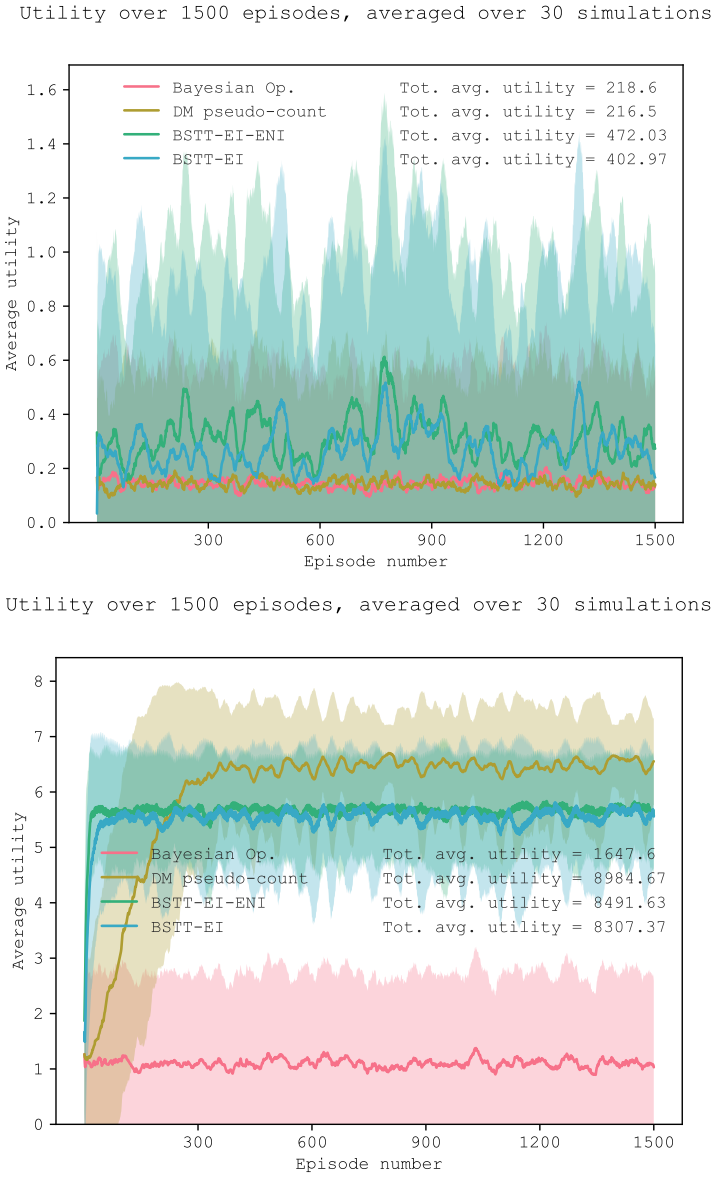
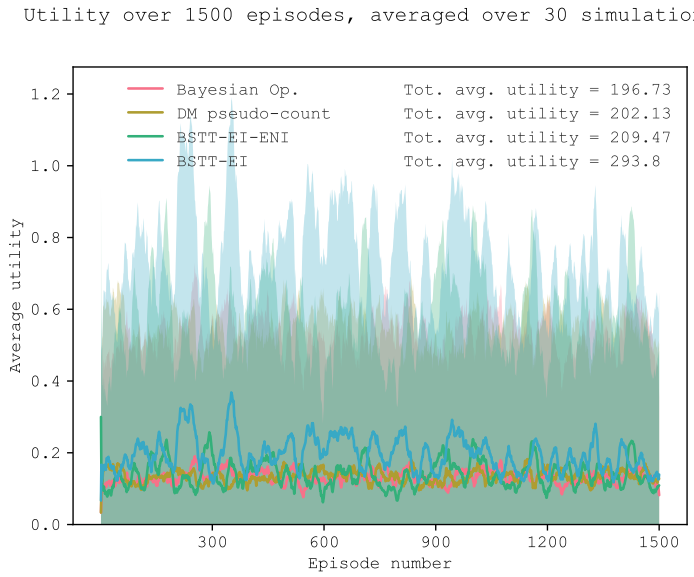


Figure 5.7: Results for Experiment 3 (fixed motion path) with varied trials per episode. In the topmost frame, the drone only has $n = 3$ trials, instead of $n = 5$ trials. In the lower frame, the drone has $n = 7$ trials. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

Experiment 2 (stochastic motion path) with varied trials



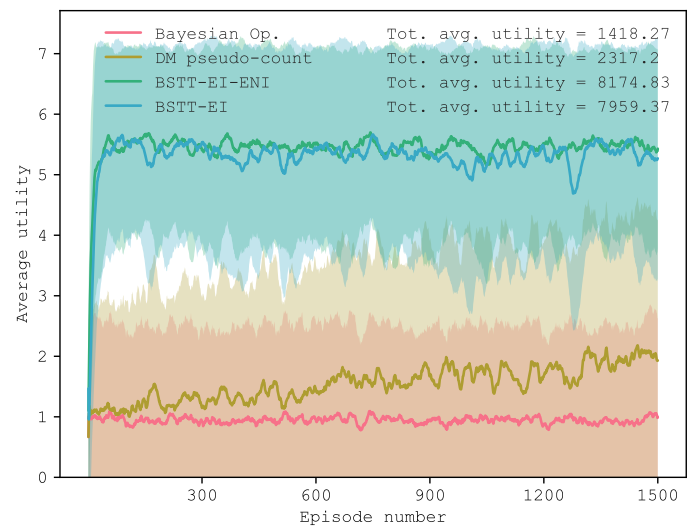


Figure 5.8: Results for Experiment 3 (stochastic motion path) with varied trials per episode. In the topmost frame, the drone only has $n = 3$ trials, instead of $n = 5$ trials. In the lower frame, the drone has $n = 7$ trials. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

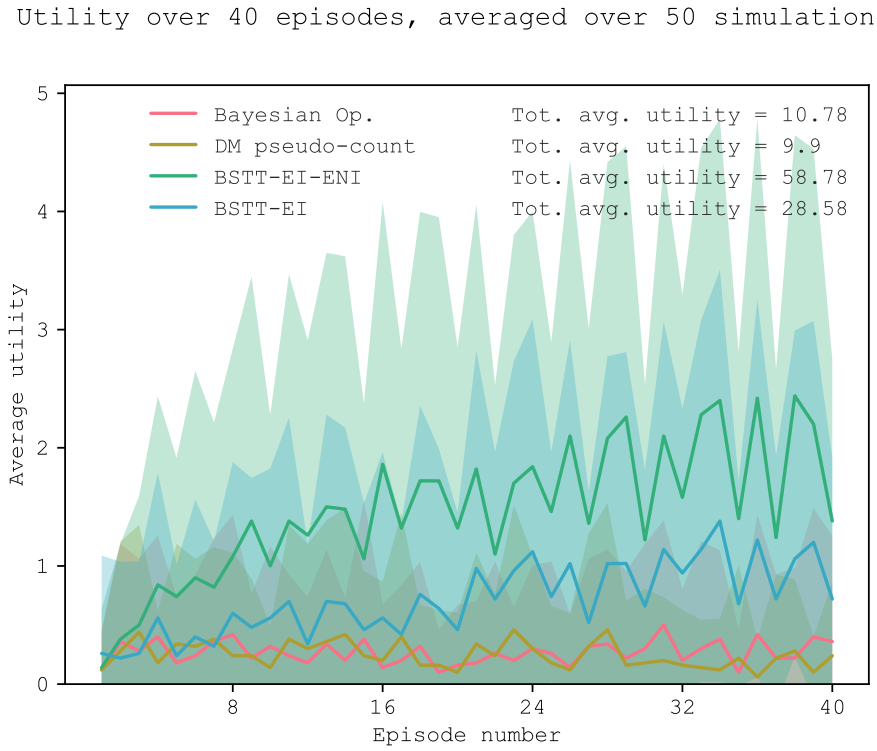


Figure 5.9: Results for Experiment 1.1 with an increased grid size of 15×15. The target follows a fixed, circular motion path. Shaded regions represent ± 1 standard deviation from the average utility. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

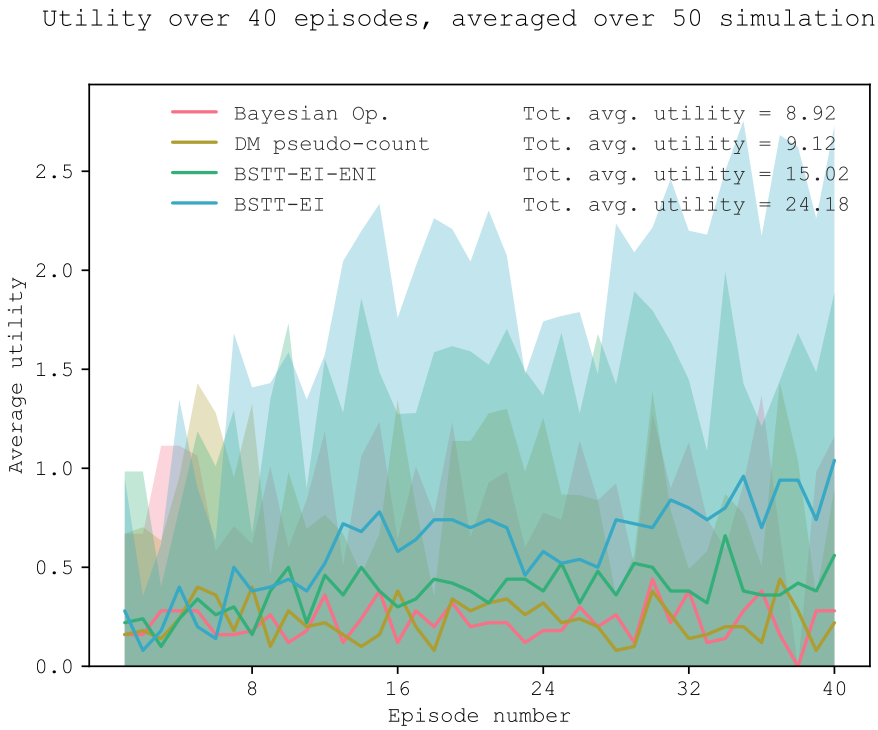


Figure 5.10: Results for Experiment 1.2 with an increased grid size of 15 × 15. The target follows a stochastic motion path. Shaded regions represent ±1 standard deviation from the average utility. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

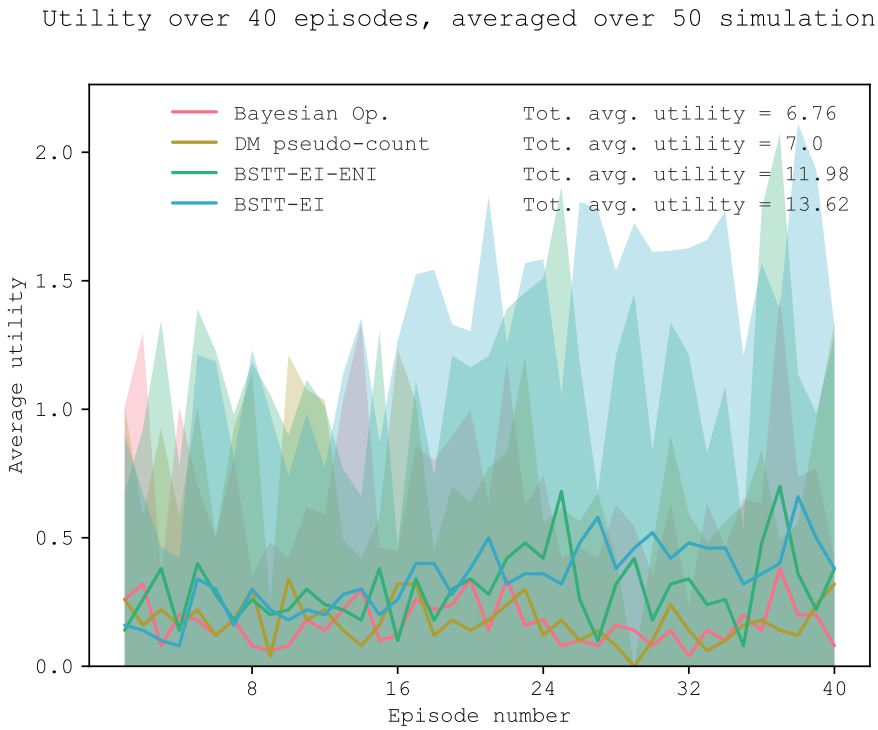


Figure 5.11: Results for Experiment 1.1 with an increased grid size of 20×20. The target follows a fixed, circular motion path. Shaded regions represent ± 1 standard deviation from the average utility. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

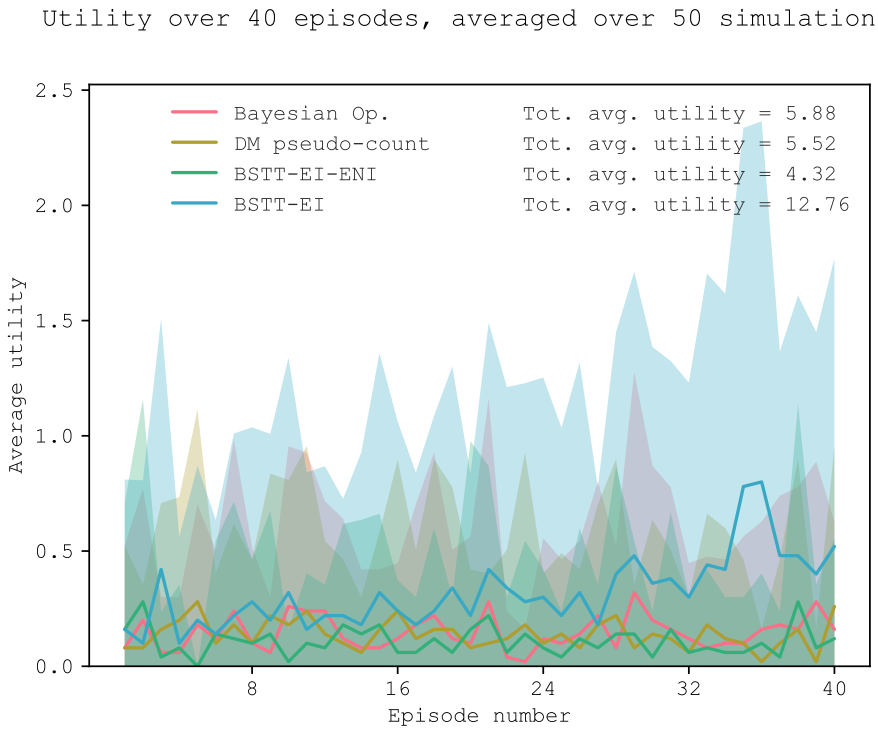


Figure 5.12: Results for Experiment 1.2 with an increased grid size of 20 × 20. The target follows a stochastic motion path. Shaded regions represent ±1 standard deviation from the average utility. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

Chapter 6

Conclusion

The findings of this research show that an autonomous learning agent may be used to search for a moving target, and track it, when given only a limited time to search for the target. Importantly, when guided by the BSTT-EI algorithm introduced in this thesis, the learning agent may be able to find and track the target in a limited number of trials, which otherwise would not be sufficient to detect the target. A target search and tracking algorithm, with an expensive search task, potentially may be guided successfully when the algorithm only knows the basic form of the target's movement model.

Furthermore, this research suggests that an autonomous target tracking agent can be deployed with a sparse sensor. Such an agent would only require sparse signals from its sensor, rather than exact coordinate information. Moreover, the agent would not require a long training period, which would limit its effectiveness in practice.

We do note that the conclusions which can be drawn about BSTT-EI-ENI are limited. BSTT-EI-ENI may only be effective in smaller domains, and future research would consider a more rigorous mathematical analysis of how one could count the potential paths which a target may have taken in order to predict where the target is less likely to be.

The success of BSTT-EI depends on the suitability of the Kinematic Motion Model used. Here, it was assumed that a constant pseudo-velocity model was suitable. However, specifying a Kinematic Motion Model is a challenging task on its own and the success of such a model depends on the accuracy of the specified parameters [4]. To practically apply these algorithms, one would need to know that the target's motion could be modeled by a constant pseudo-velocity model. Furthermore, one would need to know the magnitude of the target's changes in pseudo-velocity, as well as their likelihood. All of these assumptions may be difficult to model in practice.

The implication is that the algorithms are suitable when the learning agent knows what type of target it is searching for. For example, if the target is a large, slow-moving animal, then it can be assumed to move at a constant velocity with very small expected changes in velocity. Whereas if the target is more agile, then its expected changes in velocity would be greater. In such cases, specifying an accurate Kinematic Motion Model may be plausible.

This thesis does make simplistic assumptions about the drone's signal model

and the signal noise. In reality, these are very challenging to model. Firstly, both are usually dynamic and change over time. Yet, in this thesis the signal noise was modeled as remaining constant over time. Secondly, building a model of either generally requires a long history of data. This is analogous to a requiring long training period during which to observe the environment. Such an assumption is not in line with the research aim. However, when signal noise is limited or when the learning agent is deployed in a familiar domain, then modeling signal noise is less of an issue.

Furthermore, when the learning agent has a sparse, binary sensor, it is difficult to differentiate between the target and any other objects in the environment. Ideally, the target would be the only object in the area which emits a signal. If not, the sensor would require a way of differentiating between the target and other objects in the domain. To do this, it would require a sensor which gives more granular signals.

Another limitation of the proposed algorithms is that they are not suitable for tracking multiple targets. The tracking task would be more computationally intensive when the targets cannot be uniquely identified. As the number of targets increases, the computational load increases polynomially [1]. As the computational load increases, the learning agent becomes slower to find and track the target.

Augmenting the search using trajectories of negative information is a novel approach and this thesis serves as a basic proof of the concept. Future work would consider more rigorously testing its theoretical basis, and more rigorously defining the conditions under which the methodology is suitable. Additionally, future work would quantify the amount by which a trajectory of negative information influences the belief of where the target is less likely to be.

Using negative information may be necessary if one has limited knowledge of the target's motion model, but does not wish to discard learnings from searches which did not yield positive information. If one has a prior estimate of the target's pseudo-velocity, or its limits, then BSTT-EI is suitable, and explicit use of negative information is unnecessary. However, without a prior pseudo-velocity estimate, then one may consider using BSTT-EI-ENI. However, we note that the results of the experiments using BSTT-EI-ENI should be interpreted with caution. BSTT-EI-ENI may be effective only in limited scenarios.

Future work may also consider adversarial targets. An adversarial target is aware of the drone and attempts to evade detection. Such targets adapt their motion path based on the observed search strategy of the drone. Arguably, a solution for detecting and tracking an adversarial target may have a broad set of applications [1].

Adversarial work would consider modeling the target's motion with more dimensions [1]. This means that one would consider more than the target's kinematics when modeling its movement. Other potential considerations include a utility model for the target, as well as a model for the target's movement strategy. A utility model would assign values to different locations in the domain. The target may have an inclination to move to particular locations which may be more beneficial to the target. A movement strategy may consider the evasion tactics of the target. For example, the target may attempt to change its speed in an attempt to avoid detection. Needless to say, adversarial target modeling is a complex topic on its own, hence it has not been explored in this thesis.

As discussed in Section 3.2, other future work may consider a cost function for the distance between the drone's search locations. This approach was taken in the related research discussed in [19]. In reality, the drone would incur a cost when moving between locations. This cost may be a tangible cost such as fuel, or an intangible cost such as an opportunity cost. In such a case, the solution may differ materially from the algorithms proposed in this research. However, we did not consider a cost function as we only use the drone example as motivation for a more general problem, namely, target search and tracking using sparse signals.

Nevertheless, the findings of this research do contribute meaningfully to the related literature. We have developed a target search and tracking algorithm, BSTT-EI, which can operate with sparse signals, given a limited time to search for the target.

Appendices

Appendix A

Using concepts from Kinematic Motion Models to characterize the discrete transition probabilities

Kinematic Motion Models are generally used on continuous domains. These models have proven to be effective [4], hence here we use concepts from Kinematic Motion Models to characterize a model for transition probabilities in a discrete grid domain. In our discrete grid domain, given a target's predicted location, the process noise is approximated by a Gaussian distribution, centered around the predicted location.

Our method for using concepts from Kinematic Motion Models to characterize the discrete movement model is best described with an example. Particularly, we consider the model of the target's movement from the Kinematic Motion Model (see Equation 2.3) when characterizing the discrete transition probabilities. Now consider a target in a 1D domain. It maintains a constant pseudo-velocity, v_p (defined in Equation 3.6), with probability 0.7, with small expected changes in its pseudo-velocity. Then, the discrete transition model may be set as follows. At the end of time t, the probability of transitioning to the location predicted by the constant pseudo-velocity model, location $l_{t+1} = l_t + v_p$, will be 0.7. There is a lower probability of transitioning to the locations adjacent to location l_{t+1}, and an even lower probability of the target transitioning to the locations which are much further away from location l_{t+1}. These probabilities are set so that the transition probabilities take on an approximately Gaussian form, centered around location l_{t+1}. Figure A.1 illustrates the characterization of the discrete transition probabilities using concepts from Kinematic Motion Models.

Note that strictly speaking, we are not discretizing the Kinematic Motion Model described in Section 2.4. We are using the concepts from Kinematic Motion Models to characterize how we model the target's movement in a discrete grid domain. We have defined pseudo-velocity (see Equation 3.6) based on the classic definition of velocity. Furthermore, we set the process noise in our model

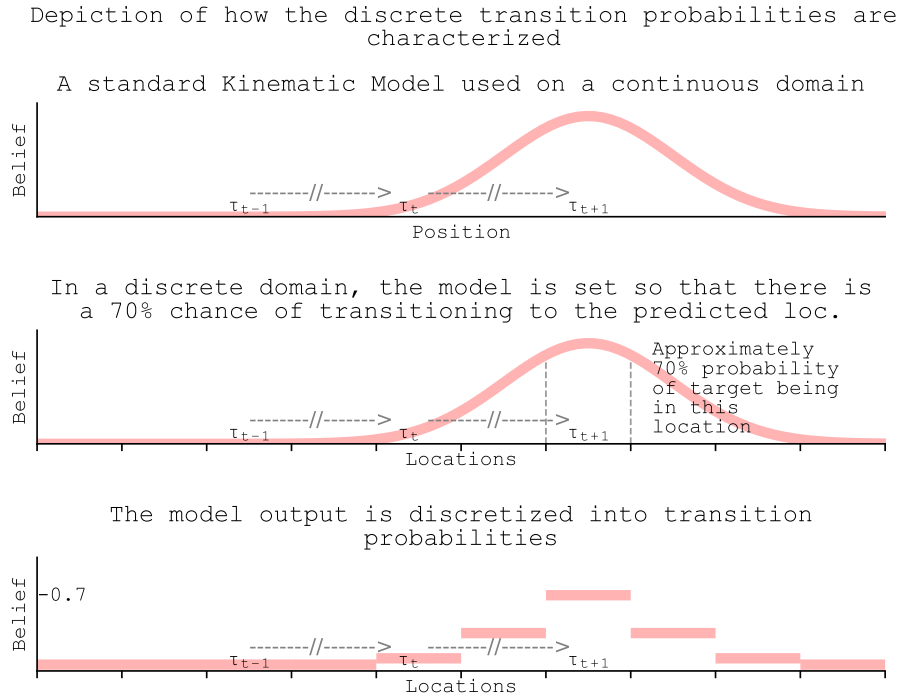


Figure A.1: This figure shows the time $t+1$ prediction after observing a target, whose estimated location is notated by τ, at times $t-1$ and t. The topmost frame shows the Gaussian process noise for a prediction on a continuous domain (see Equation 2.3). The center frame shows how the model is set in a discrete domain. The bottom frame shows how the model is then discretized into transition probabilities. This figure is only meant to illustrate how we use concepts from Kinematic Motion Models to characterize a discrete movement model, hence it does not show the shape of the belief over the target estimates at times prior to $t+1$.

to approximate the Gaussian process noise of a Kinematic Motion Model. For each coordinate of motion, the steps followed to use the Kinematic Motion Model to characterize the discrete movement model are shown below:

1. Predict where the target will move to in the discrete domain, using Equation 3.7;
2. Overlay a Gaussian distribution onto the spatial domain under consideration (see Figure A.2), centered at the target's predicted location (as per Equation 3.7);
3. Truncate the Gaussian distribution at the end points of the spatial domain;
4. Normalize the distribution so that the area under the truncated plot sums to unity; and

5. Discretize the continuous distribution into transition probabilities. The area under the distribution which falls over each location will be its transition probability (see the lower panel of Figure A.1).

Note that the shape of the Gaussian distribution is dependent on the process noise intensity parameter, defined in Section 2.4, in Equation 2.6. See Figure A.2 for a depiction of how the process noise intensity parameter affects the shape of the Gaussian distribution. Also note that in a $2D$ domain, independent motion is assumed across each coordinate of motion. Therefore after computing the transition probabilities across each coordinate of motion as described above, the transition probabilities across the entire $2D$ domain would be normalized again to sum to unity.

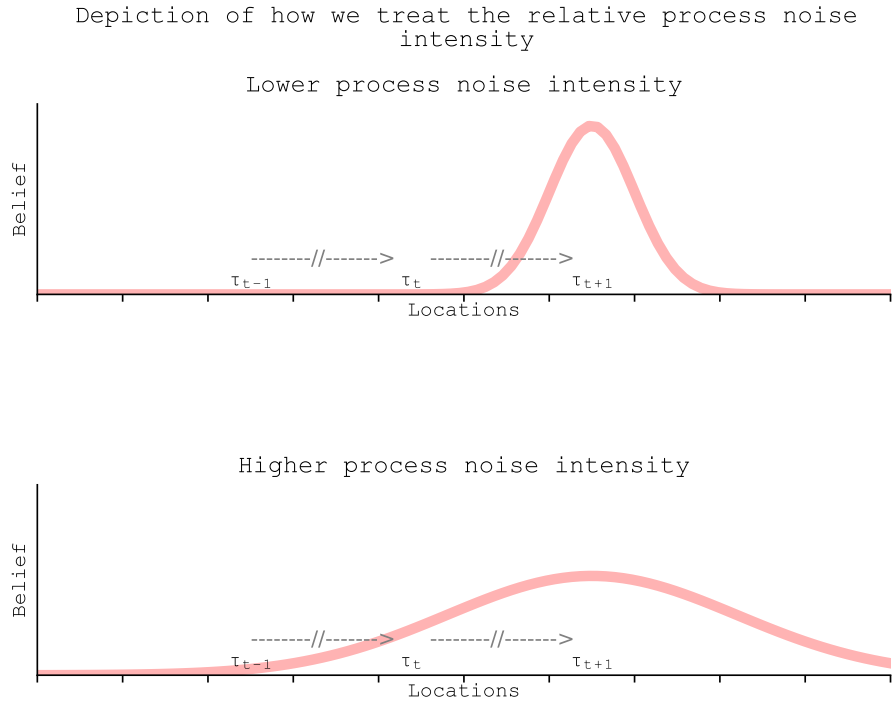


Figure A.2: This figure shows the Gaussian distribution in Figure A.1, with different process noise intensities, where the process noise intensity parameter is as described in Equation 2.6.

Appendix B

The distribution of the observation errors

The target tracking model described in Chapter 2 assumes that both the noisy observations of the target, as well as the predictions of the target's future location, have Gaussian errors. However, these assumptions may not hold in this research due to the target tracking agent being given a limited time to find the target. Given a limited number of search trials in each episode, the estimate of the target's location at the end of an episode may have an irregular distribution, even on a continuous domain. This concept is illustrated in Figure B.1.

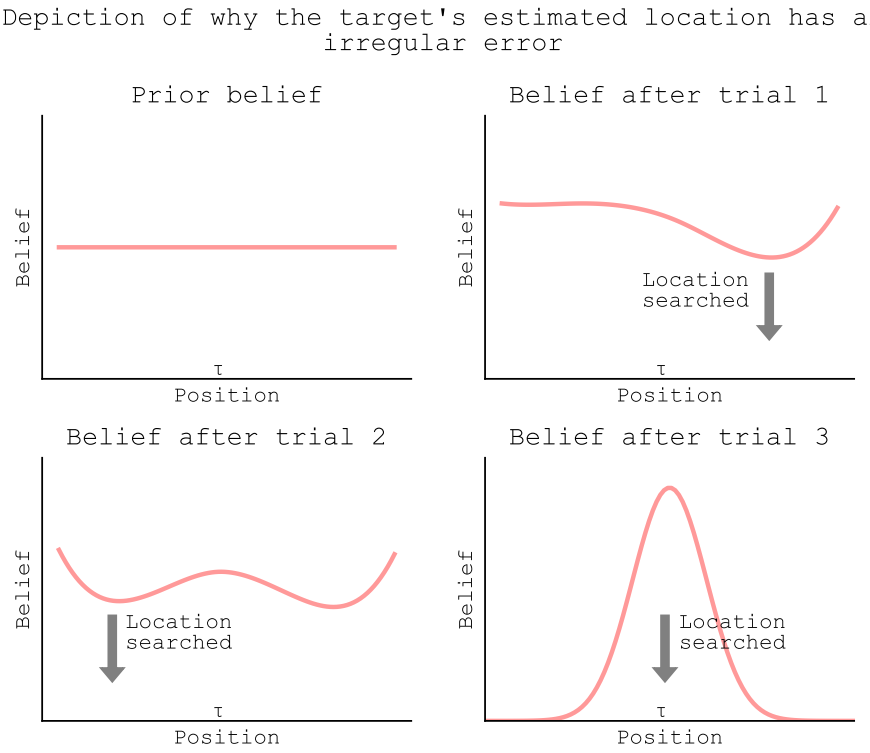


Figure B.1: Here, τ represents the target, which is stationary for the length of the episode. Given a limited number of search trials, the estimate of the target's location may not have a Gaussian error. After the first two search trials, no signal of the target was observed. After the third search trial, a strong signal of the target was observed. In this example, fewer than three trials may be insufficient to localize the target. Given more trials, the drone (guided by its learning algorithm) detects the target and the estimate has a Gaussian error.

Appendix C

The computational efficiency of Bayesian Optimization and the Wonham Filter

Given the framing of the problem described in Section 3.2, the drone's learning algorithm maintains a belief of where the target is, and updates this belief using the noisy signals it observes after searching for the target. The process of updating its belief is called filtering, which may be computationally intensive for a large domain.

Hence, if one could somehow reduce the amount of filtering required, then the computational load of the solution algorithm would be reduced. This would make the algorithm run quicker, which would allow the drone to find the target in a shorter space of time.

In the proposed solution to the research problem (see Chapter 4), solution algorithms follow two steps, namely, search for the target using Bayesian Optimization (described in Section 2.3), and then predict where the target will move to next by using the predict step of the Wonham Filter (shown in Equation 2.10). One may initially think that after the target is searched for using Bayesian Optimization, a filtering step using the Wonham Filter would be required. In other words, one may think that the Wonham Filter's filtering component is used to filter what is observed during the search before predicting where the target will move to next. However, here we show that the filtering step of the Wonham Filter is not required as it is carried out in the Bayesian Optimization step.

Below, we show that only one explicit filtering step is required, which limits the computational load born by filtering. This means that the choices of Bayesian Optimization and the Wonham Filter limit the computational load borne by the drone's learning algorithm. Hence, one may describe them as efficient when used together to address this research problem.

We now show that only one filtering step is required between the Bayesian Optimization step and the Wonham Filter's filtering step. Recall Equation 2.9, which shows that by the Wonham Filter, the time t filtered belief of the target's

location, $\mathbf{Pr(L)}_{t^+}$, is given as:

$$\mathbf{Pr(L)}_{t^+} = \frac{\mathbf{B}(*, \sigma) \odot \mathbf{Pr(L)}_{t^-}}{\mathbf{B}(*, \sigma)^T \mathbf{Pr(L)}_{t^-}}$$

assuming an observed signal σ, an observation model $\mathbf{B}$, and a prior belief $\mathbf{Pr(L)}_{t^-}$. Now let C represent a normalizing constant. Using the above equation, the posterior belief for each location $j \in \mathbf{L}$ is calculated as:

$$\begin{aligned} Pr(l = j)_{t^+} &= \frac{B(j, \sigma) \odot Pr(l = j)_{t^-}}{\mathbf{B}(*, \sigma)^T \mathbf{Pr(L)}_{t^-}} \\ &= \frac{B(j, \sigma) \odot Pr(l = j)_{t^-}}{C} \\ &= \frac{B(j, \sigma).Pr(l = j)_{t^-}}{C} \\ &= \frac{Pr(\sigma|j).Pr(j)}{C} \\ &= Pr(j|\sigma) \end{aligned}$$

The last two lines of the above equation can be recognized from Bayes' Theorem, given in Equation 2.1. Recall that Bayes' Theorem is the equation used in the filtering step of Bayesian Optimization. This means that the explicit filtering step from the Wonham Filter is unnecessary because it is done in the Bayesian Optimization step.

By limiting the amount of filtering required, the computational load is reduced. Hence, one can motivate that the use of Bayesian Optimization and the predict step of the Wonham Filter are computationally efficient techniques for addressing this research problem.

Appendix D

Modeling the target's movement after hitting a boundary

In the experiments in Chapter 5, the target moves in a bounded, spatial grid domain. Here, it is shown how the target's movement is affected when it hits the bounds of the domain.

The reaction of the target after reaching a boundary is best illustrated by way of example. Consider a target moving according to a constant pseudo-velocity model. Across each coordinate of motion, the target maintains its pseudo-velocity with probability 0.7. With probabilities 0.15 and 0.15, it increases or decreases its pseudo-velocity by ± 1 location per episode, respectively. As discussed in Section 2.4, independent motion is assumed across each coordinate.

Now assume that this target was initially moving with a pseudo-velocity of 3 locations per episode, to the right. When it reaches the boundary, it would be slowed down, and its pseudo-velocity would be reduced to 0. Thereafter, it would maintain a pseudo-velocity of 0 with probability 0.7, or it would increase its pseudo-velocity by ± 1 location in the opposite direction, and move to another location in the domain. This is illustrated in Figure D.1.

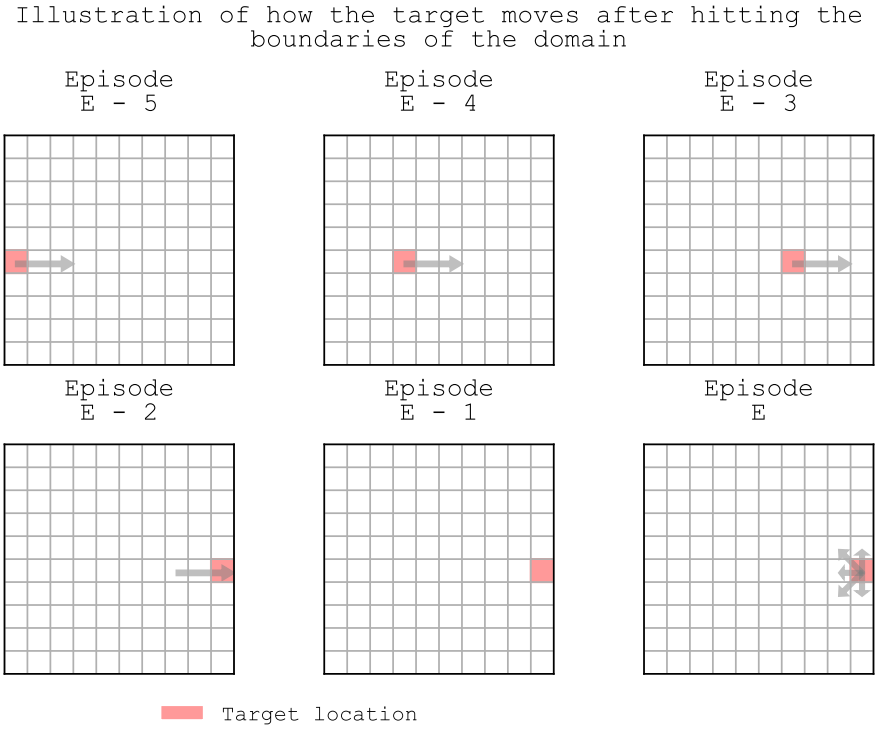


Figure D.1: In the first four episodes, the target maintains a pseudo-velocity of 3 locations per episode, moving to the right. After hitting a boundary, the target's pseudo-velocity, for motion in the respective coordinate, is reduced to 0. Thereafter, it would maintain a pseudo-velocity of 0 with probability 0.7, or it would move to one of the adjacent locations, with probability 0.3.

Appendix E

Sample traces of the target's track

The experiments in Chapter 5 are carried out on targets which follow either a fixed motion path or a stochastic motion path. The fixed motion path is a circular track shown in Figure E.1.

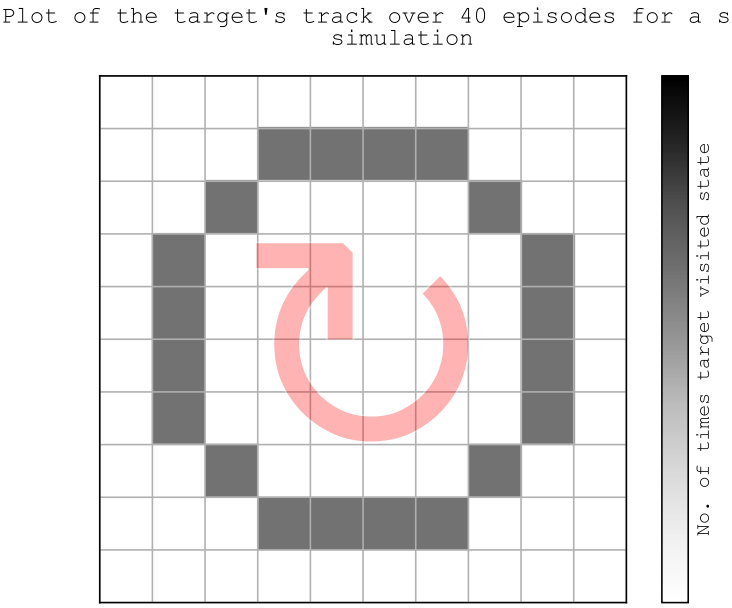


Figure E.1: The target maintains a fixed, clockwise motion path. Over 40 episodes, the target goes around the track twice.

The stochastic motion path is simulated from a constant pseudo-velocity model. In this model, the target maintains a constant pseudo-velocity with probability 0.7. The target increases or decreases its pseudo-velocity by ± 1

location per episode with probabilities 0.15 and 0.15, respectively. Sample traces of the target's track, simulated from the stochastic model, are shown in Figures E.2, E.3 and E.4.

More granularly, the stochastic model is simulated as follows. An initial location in the domain is randomly chosen. Across each coordinate, the target's initial pseudo-velocity is either -1, 0 or 1 with equal probability. Thereafter, the constant pseudo-velocity model described above is applied, for motion in each coordinate.

Notably, under the stochastic model, when the target hits the bounds of the domain it slows down, as described in Appendix D. After slowing down, it tends to dwell near the bounds of the domain due to it losing speed after hitting a boundary.

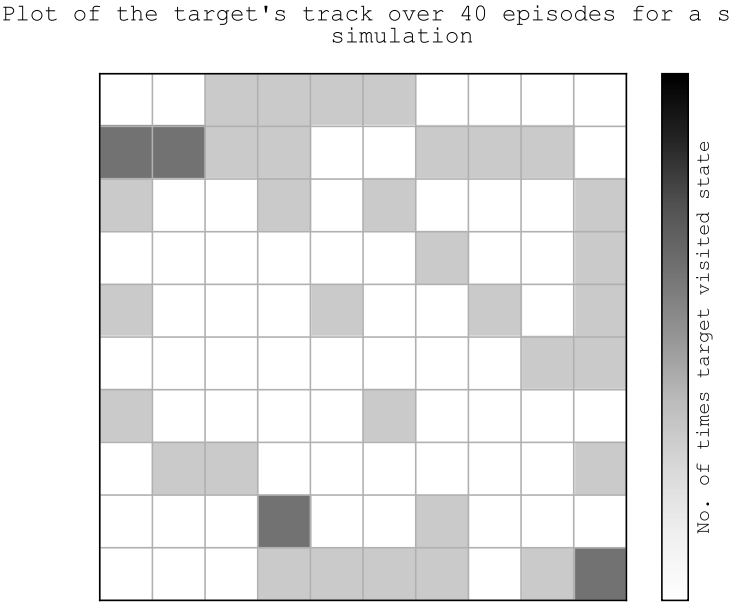


Figure E.2: The target moves according to a constant pseudo-velocity model. This is number 1 of 3 sample traces from the constant pseudo-velocity model.

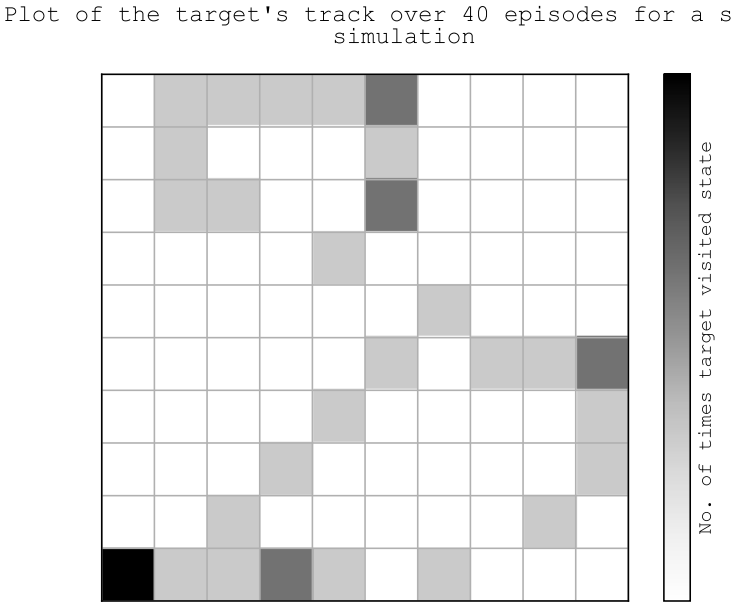


Figure E.3: The target moves according to a constant pseudo-velocity model. This is number 2 of 3 sample traces from the constant pseudo-velocity model.

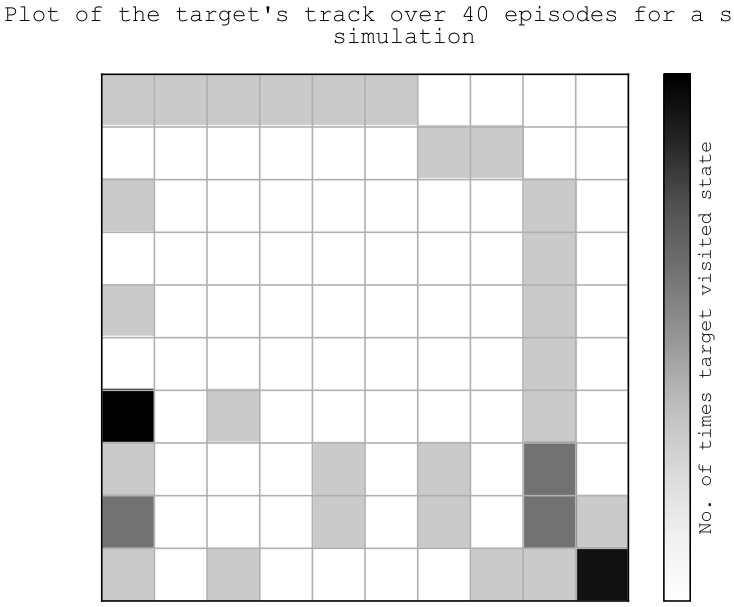


Figure E.4: The target moves according to a constant pseudo-velocity model. This is number 3 of 3 sample traces from the constant pseudo-velocity model.

Appendix F

Experimental results with increased signal noise

The experiments in Chapter 5 are carried out with moderate signal noise. Here, the results of selected experiments are shown when the signal noise is increased. The revised signal noise model, $\mathbf{B}^{'}$, is given below:

$$\mathbf{B}^{'} = \begin{bmatrix} B^{'}(0,0) & B^{'}(0,0.5) & B^{'}(0,1) \\ B^{'}(0.5,0) & B^{'}(0.5,0.5) & B^{'}(0.5,1) \\ B^{'}(1,0) & B^{'}(1,0.5) & B^{'}(1,1) \end{bmatrix} = \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.3 & 0.6 \end{bmatrix}$$

where $B^{'}(\sigma, \sigma^{'})$ is the probability of observing a noisy signal of $\sigma^{'}$, when the noiseless signal would be σ. Given the revised signal model, the drone only has a 0.6 probability of receiving an accurate signal, whereas in Chapter 5, the probability of receiving an accurate signal was 0.95.

Figures F.1, and F.2 show the results for selected experiments in Chapter 5, with increased signal noise. The results are intuitive: when the signal noise is increased, all of the algorithms become increasingly ineffective at finding and tracking the target.

Experiment 1.1 with increased signal noise

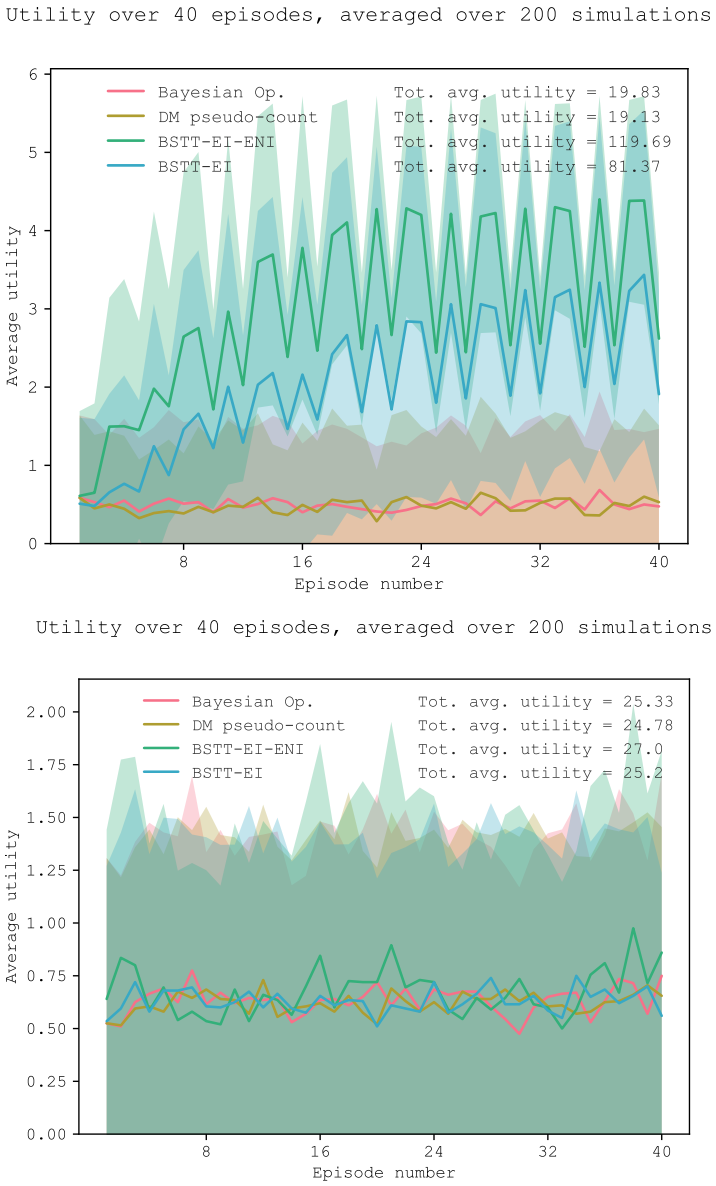


Figure F.1: Results for Experiment 1.1 with increased signal noise. In the topmost frame, the results of the original experiment are shown with the probability of receiving an accurate signal set to 0.95. In the lower frame, the signal noise is increased, and the probability of receiving an accurate signal is reduced to 0.6. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

Experiment 1.2 with increased signal noise

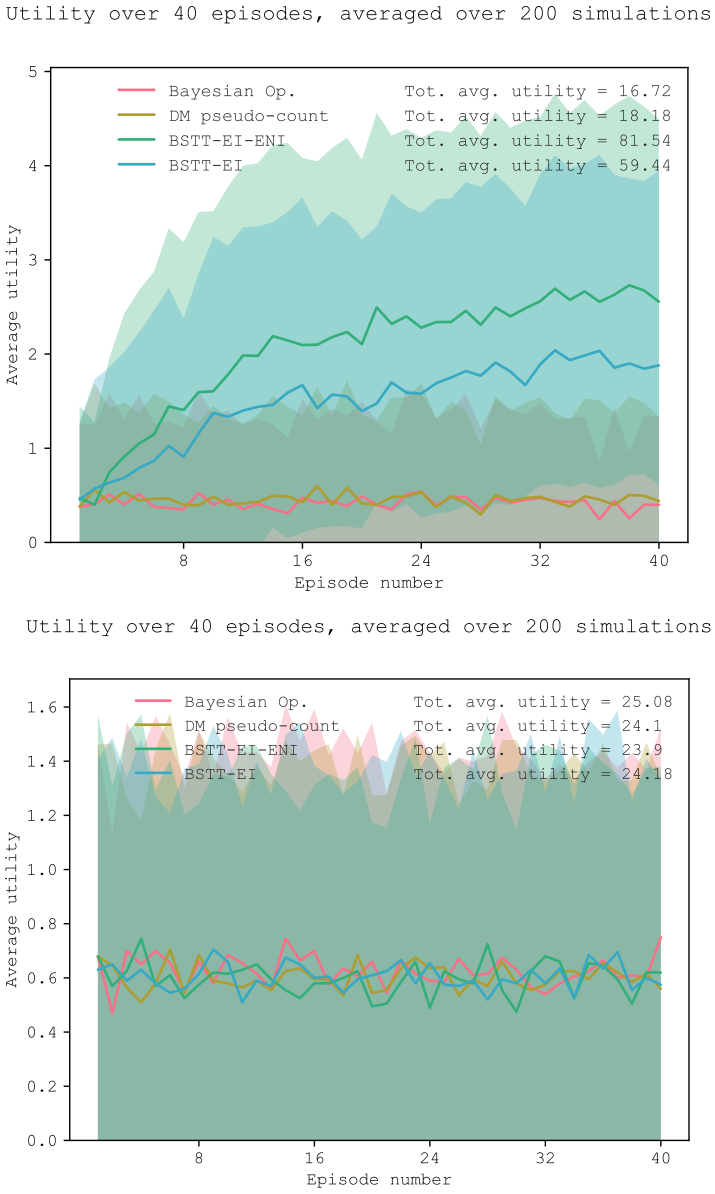


Figure F.2: Results for Experiment 1.2 with increased signal noise. In the topmost frame, the results of the original experiment are shown with the probability of receiving an accurate signal set to 0.95. In the lower frame, the signal noise is increased, and the probability of receiving an accurate signal is reduced to 0.6. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

Appendix G

Experimental results with a misspecified motion model

The experiments in Chapter 5 are carried out under the assumption that the learning algorithm's motion model accurately reflects the motion model used by the target. Here, the results of selected experiments are shown when the motion model assumed by the drone's learning algorithm is different to the model used by the target.

Here, the target follows the same motion model as in Chapter 5. For motion in each coordinate, the target maintains a constant pseudo-velocity with probability 0.7, and changes its pseudo-velocity by ± 1 location per episode, with probability 0.3. If it changes pseudo-velocity, it increases its pseudo-velocity or decreases its pseudo-velocity with equal probability.

In Chapter 5, it was assumed that the drone's learning algorithm knew of this motion model. However, here we assume that the drone's learning algorithm has an inaccurate model. For motion in each coordinate, the drone's learning algorithm assumes that the target maintains a constant pseudo-velocity with probability 0.4. It assumes that the target increases its pseudo-velocity by 1 location per episode, with probability 0.25. It also assumes that the target increases its pseudo-velocity by 2 locations per episode with probability 0.05. The probabilities of the target decreasing its pseudo-velocity are the same as the probabilities of the target increasing its pseudo-velocity.

The learning algorithm's assumed model of the target's pseudo-velocity has the following implications. The target is assumed to have a more erratic path than it actually does. Alternatively, the drone's learning algorithm may specify such a motion model if it is unsure of the target's actual motion model. If the drone's learning algorithm only knows that the target moves according to a constant pseudo-velocity model, then the misspecified model described above may represent the learning algorithm's uncertainty of how frequently the target changes its pseudo-velocity.

Figures G.1 and G.2 show the revised results for selected experiments in Chapter 5. The results show that BSTT-EI and BSTT-EI-ENI are robust to a motion model with inaccurately specified process noise. It is most important that the drone's learning algorithm knows that the target's pseudo-velocity characterizes its motion. For as long as the drone's learning algorithm knows that

any changes in the target's pseudo-velocity are small, the algorithms may be applied effectively. This holds even when the small changes in pseudo-velocity happen frequently.

Experiment 1.1 with a misspecified motion model

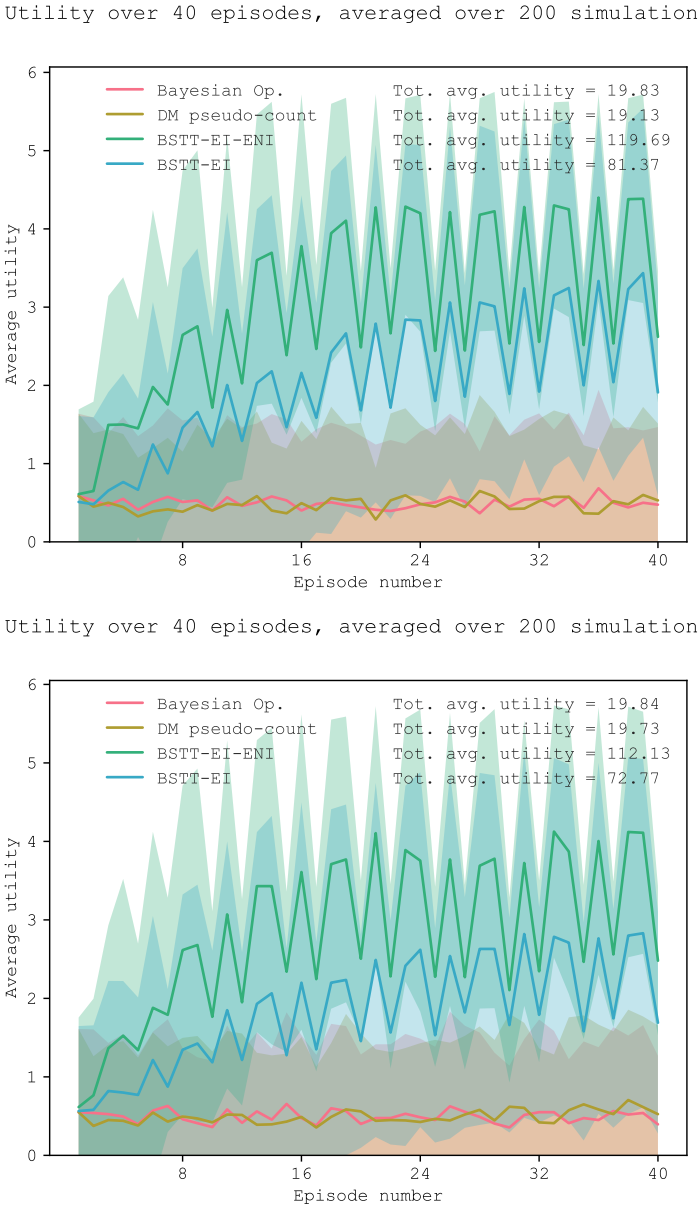


Figure G.1: Results for Experiment 1.1 for a learning algorithm with a misspecified motion model. In the topmost frame, the results of the original experiment are shown, when the drone's learning algorithm has an accurately specified motion model. In the lower frame, the learning algorithm's motion model assumes that the target is highly likely to change its pseudo-velocity. Here, the target follows a fixed, circular motion path. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

Experiment 1.2 with a misspecified motion model

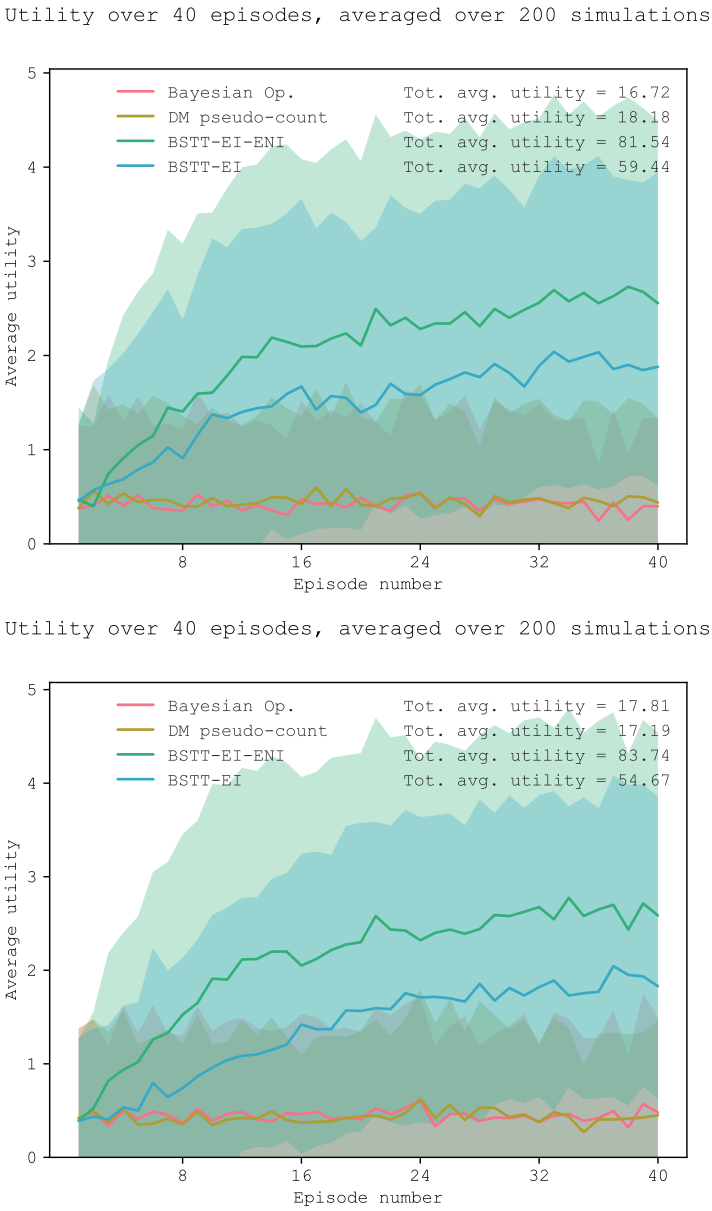


Figure G.2: Results for Experiment 1.2 for a learning algorithm with a misspecified motion model. In the topmost frame, the results of the original experiment are shown, when the drone's learning algorithm has an accurately specified motion model. In the lower frame, the learning algorithm's motion model assumes that the target is highly likely to change its pseudo-velocity. Here, the target follows a stochastic motion path. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

Appendix H

Sample trace of the drone's searches

Here, we show a sample trace of the drone's search path, for all the algorithms. In this sample trace, the conditions are the same as in the Experiment in Section 5.2.1. The target follows a fixed, circular motion path. However, the pseudo-velocity of the target is unknown to the drone's learning algorithm at the outset, and neither are the bounds of the target's pseudo-velocity. The sample trace is shown in Figures H.1 to H.10.

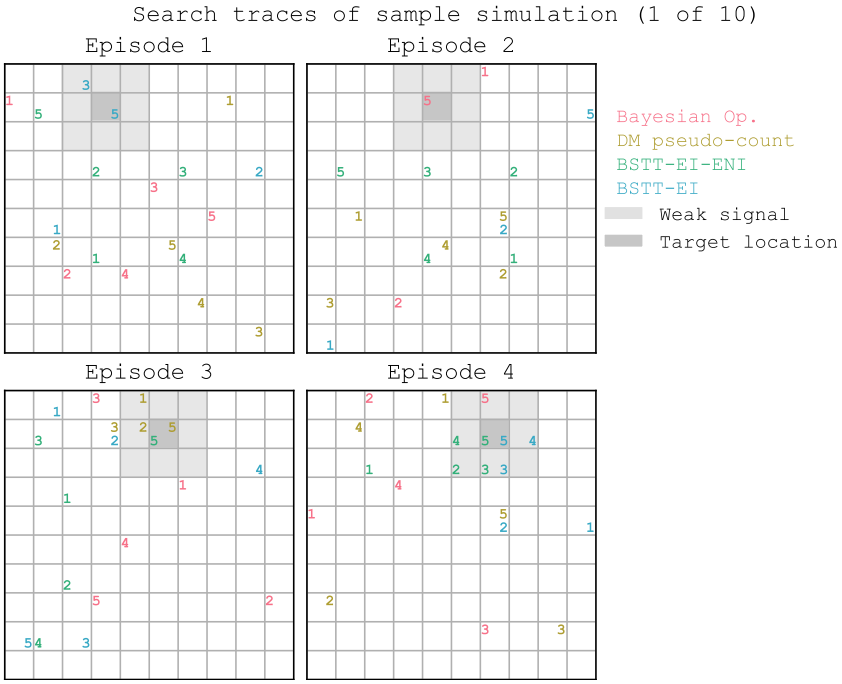


Figure H.1: A sample trace of the drone's search footprints are shown, for all the algorithms. Here, the target moves in a fixed, circular motion path, as in the experiment of Section 5.2.1. In this figure, if multiple searches are carried out in the same location by the same algorithm, then only the trial number of the last search is shown. Bayesian Optimization is shown in the top left corner of each location; the Augmented DM pseudo-count is shown in the top right corner of each location; BSTT-EI-ENI is shown in the bottom left corner of each location; and BSTT-EI is shown in the bottom right corner of each location.

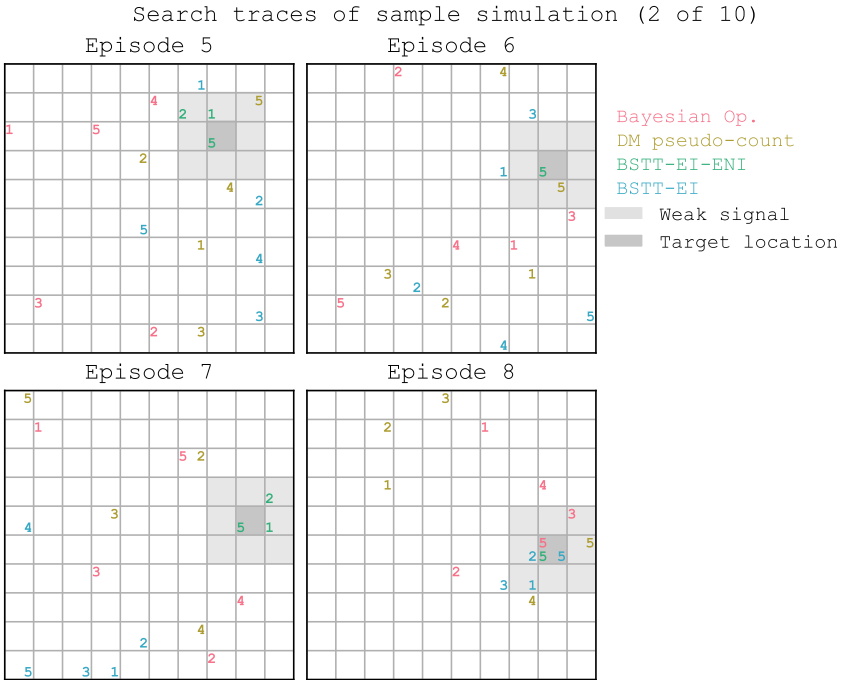


Figure H.2: A sample trace of the drone's search footprints are shown, for all the algorithms. Here, the target moves in a fixed, circular motion path, as in the experiment of Section 5.2.1. In this figure, if multiple searches are carried out in the same location by the same algorithm, then only the trial number of the last search is shown. Bayesian Optimization is shown in the top left corner of each location; the Augmented DM pseudo-count is shown in the top right corner of each location; BSTT-EI-ENI is shown in the bottom left corner of each location; and BSTT-EI is shown in the bottom right corner of each location.

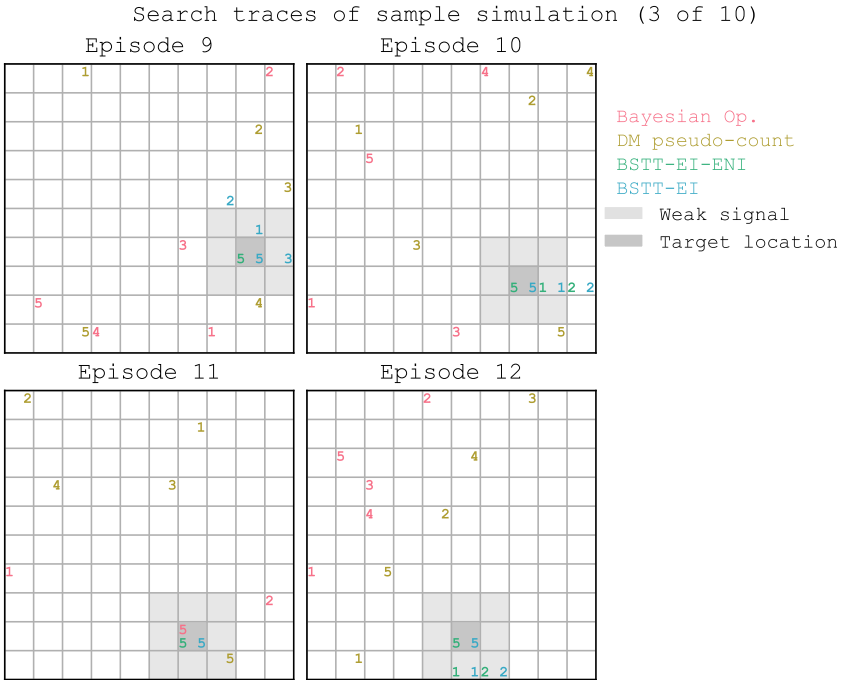


Figure H.3: A sample trace of the drone's search footprints are shown, for all the algorithms. Here, the target moves in a fixed, circular motion path, as in the experiment of Section 5.2.1. In this figure, if multiple searches are carried out in the same location by the same algorithm, then only the trial number of the last search is shown. Bayesian Optimization is shown in the top left corner of each location; the Augmented DM pseudo-count is shown in the top right corner of each location; BSTT-EI-ENI is shown in the bottom left corner of each location; and BSTT-EI is shown in the bottom right corner of each location.

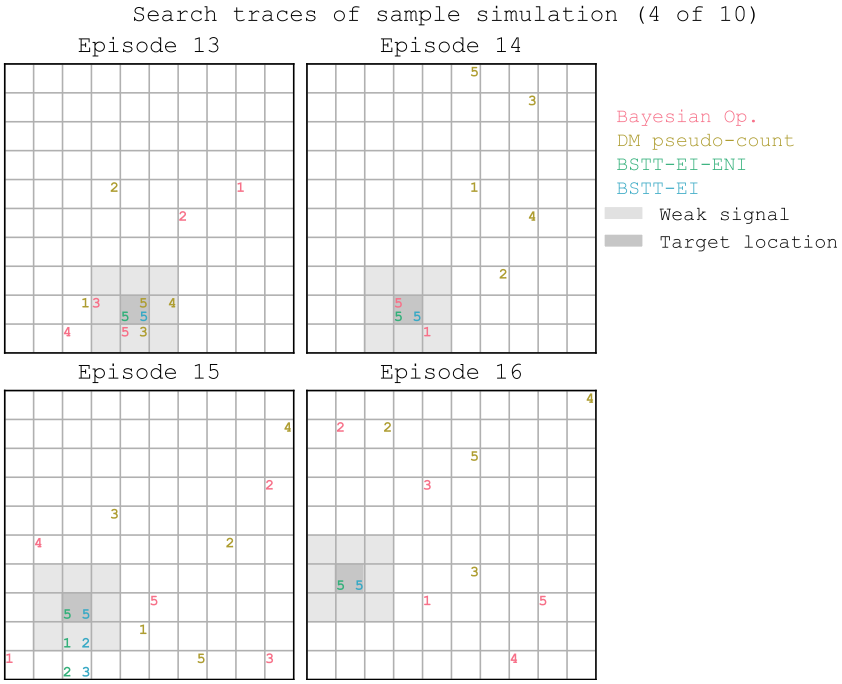


Figure H.4: A sample trace of the drone's search footprints are shown, for all the algorithms. Here, the target moves in a fixed, circular motion path, as in the experiment of Section 5.2.1. In this figure, if multiple searches are carried out in the same location by the same algorithm, then only the trial number of the last search is shown. Bayesian Optimization is shown in the top left corner of each location; the Augmented DM pseudo-count is shown in the top right corner of each location; BSTT-EI-ENI is shown in the bottom left corner of each location; and BSTT-EI is shown in the bottom right corner of each location.

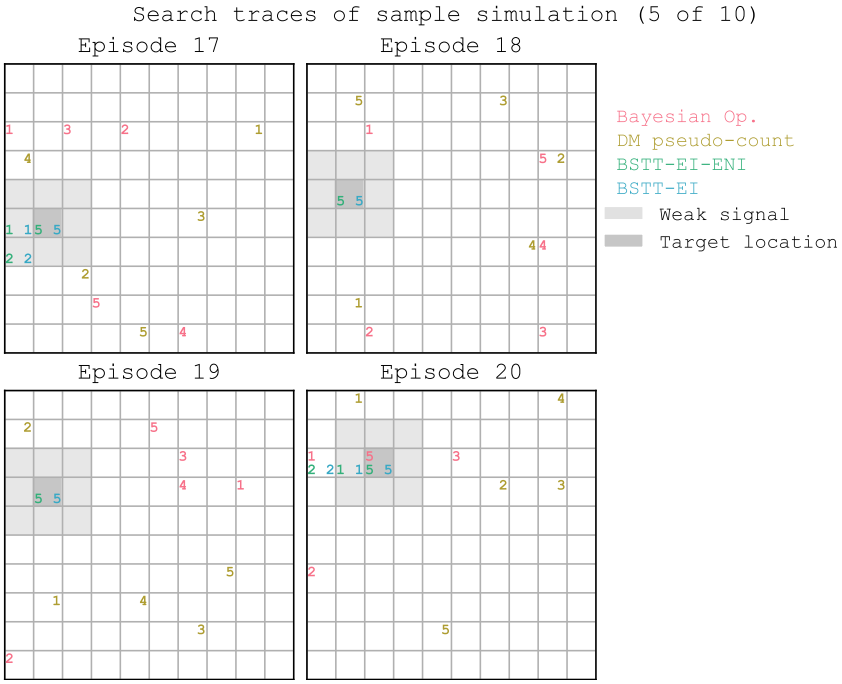


Figure H.5: A sample trace of the drone's search footprints are shown, for all the algorithms. Here, the target moves in a fixed, circular motion path, as in the experiment of Section 5.2.1. In this figure, if multiple searches are carried out in the same location by the same algorithm, then only the trial number of the last search is shown. Bayesian Optimization is shown in the top left corner of each location; the Augmented DM pseudo-count is shown in the top right corner of each location; BSTT-EI-ENI is shown in the bottom left corner of each location; and BSTT-EI is shown in the bottom right corner of each location.

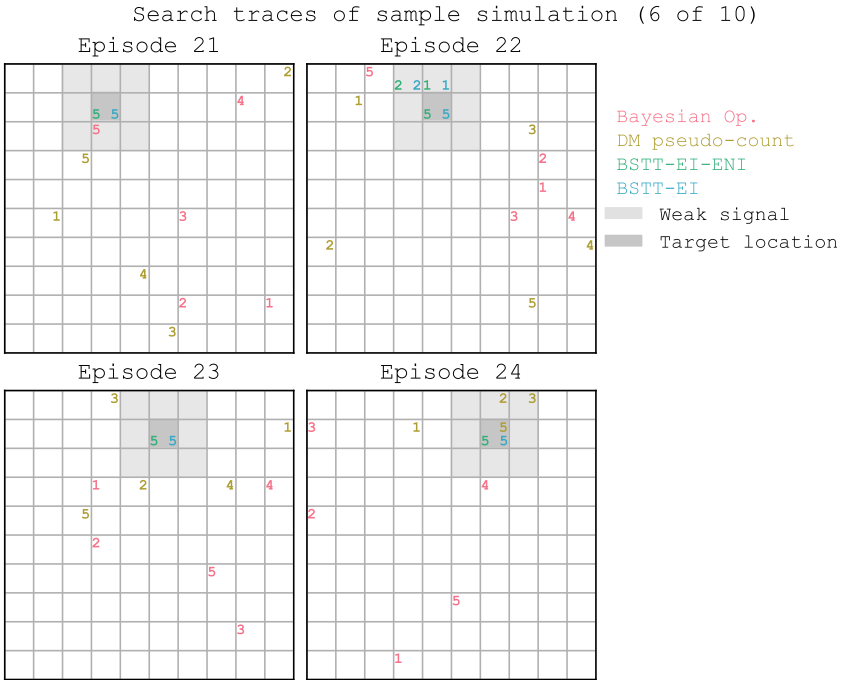


Figure H.6: A sample trace of the drone's search footprints are shown, for all the algorithms. Here, the target moves in a fixed, circular motion path, as in the experiment of Section 5.2.1. In this figure, if multiple searches are carried out in the same location by the same algorithm, then only the trial number of the last search is shown. Bayesian Optimization is shown in the top left corner of each location; the Augmented DM pseudo-count is shown in the top right corner of each location; BSTT-EI-ENI is shown in the bottom left corner of each location; and BSTT-EI is shown in the bottom right corner of each location.

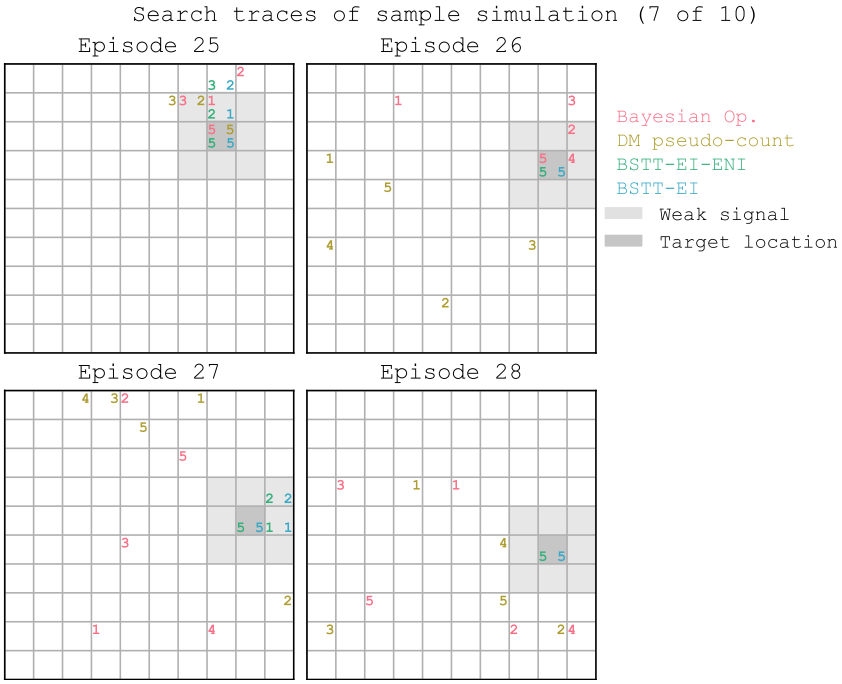


Figure H.7: A sample trace of the drone's search footprints are shown, for all the algorithms. Here, the target moves in a fixed, circular motion path, as in the experiment of Section 5.2.1. In this figure, if multiple searches are carried out in the same location by the same algorithm, then only the trial number of the last search is shown. Bayesian Optimization is shown in the top left corner of each location; the Augmented DM pseudo-count is shown in the top right corner of each location; BSTT-EI-ENI is shown in the bottom left corner of each location; and BSTT-EI is shown in the bottom right corner of each location.

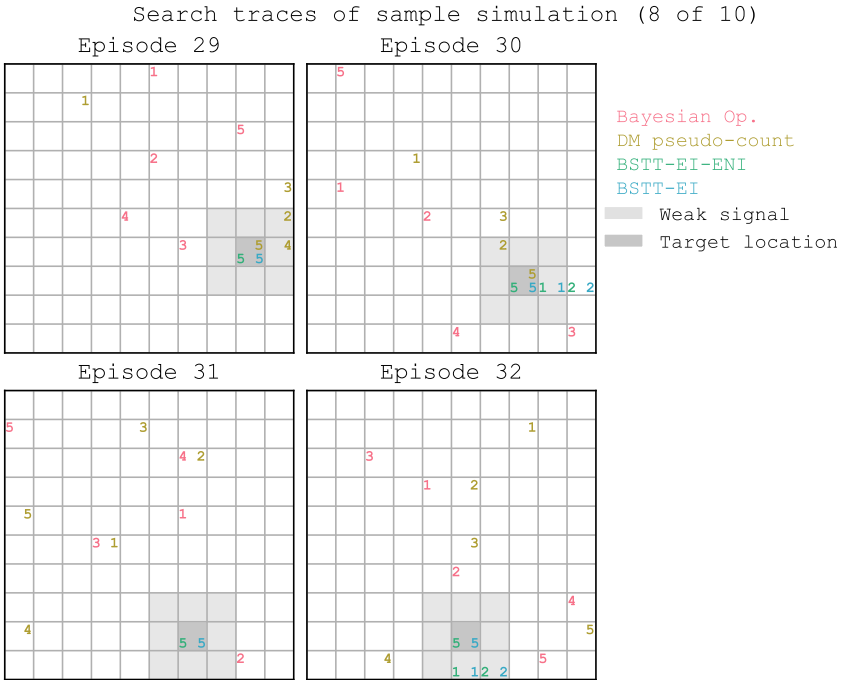


Figure H.8: A sample trace of the drone's search footprints are shown, for all the algorithms. Here, the target moves in a fixed, circular motion path, as in the experiment of Section 5.2.1. In this figure, if multiple searches are carried out in the same location by the same algorithm, then only the trial number of the last search is shown. Bayesian Optimization is shown in the top left corner of each location; the Augmented DM pseudo-count is shown in the top right corner of each location; BSTT-EI-ENI is shown in the bottom left corner of each location; and BSTT-EI is shown in the bottom right corner of each location.

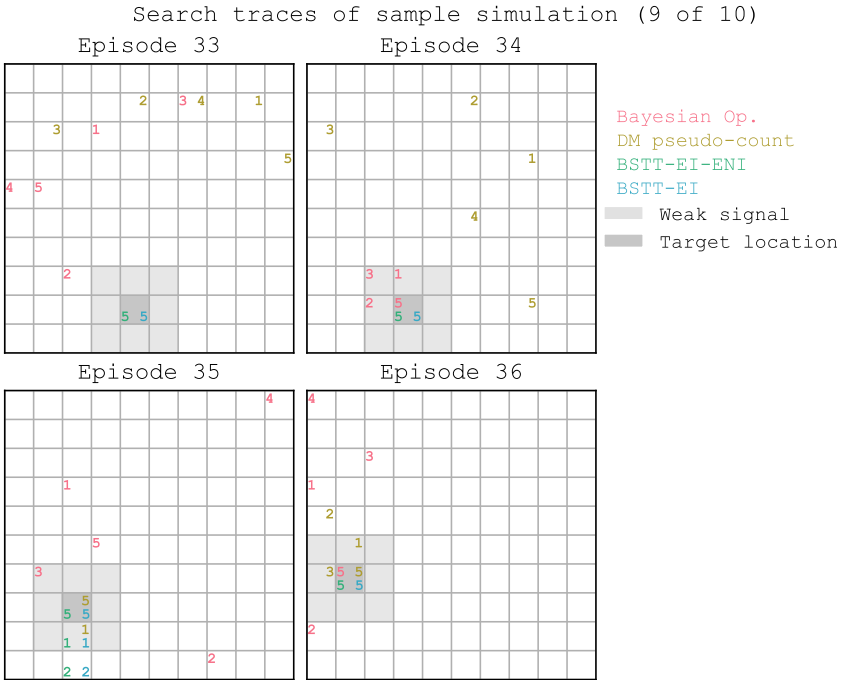


Figure H.9: A sample trace of the drone's search footprints are shown, for all the algorithms. Here, the target moves in a fixed, circular motion path, as in the experiment of Section 5.2.1. In this figure, if multiple searches are carried out in the same location by the same algorithm, then only the trial number of the last search is shown. Bayesian Optimization is shown in the top left corner of each location; the Augmented DM pseudo-count is shown in the top right corner of each location; BSTT-EI-ENI is shown in the bottom left corner of each location; and BSTT-EI is shown in the bottom right corner of each location.

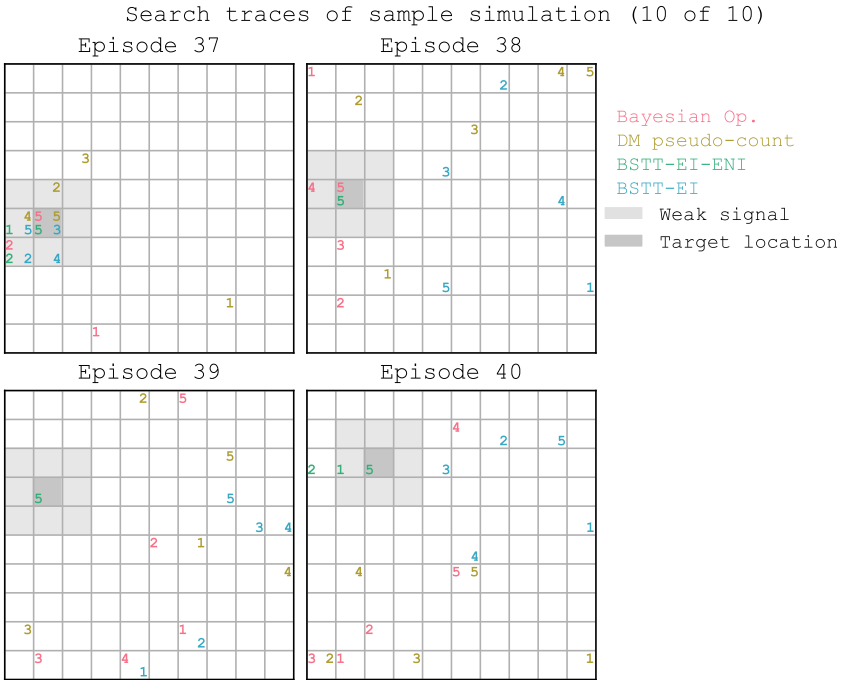


Figure H.10: A sample trace of the drone's search footprints are shown, for all the algorithms. Here, the target moves in a fixed, circular motion path, as in the experiment of Section 5.2.1. In this figure, if multiple searches are carried out in the same location by the same algorithm, then only the trial number of the last search is shown. Bayesian Optimization is shown in the top left corner of each location; the Augmented DM pseudo-count is shown in the top right corner of each location; BSTT-EI-ENI is shown in the bottom left corner of each location; and BSTT-EI is shown in the bottom right corner of each location.

Appendix I

Experimental results on a larger grid, with a linear motion path

The experiments in Chapter 5 are carried on a target which follows either a fixed circular motion path or a stochastic motion path. Here, the results of the experiments are shown when the grid size is 20×20, and the target follows a fixed linear motion path (shown in Figure I.1). Similar linear motion paths are considered in [3][5][7]. In this experiment, the target is initially located in the topmost row of the grid, in the central column. It then moves at a pseudo-velocity of 1 location per episode down, until it reaches the bottom of the grid. Thereafter, it moves at 1 location per episode back up to the top of the grid.

Figure I.2 shows the performance of the algorithms when the target follows a fixed linear motion path on a 20×20 grid. Both BSTT-EI and BSTT-EI-ENI incorporate the assumption that the target maintains a constant pseudo-velocity with probability 0.95, and changes its pseudo-velocity by ± 1 location per episode with probability 0.05. All other assumptions are the same as in Chapter 5. The drone (guided by its learning algorithm) is given $n = 5$ search trials in each episode, and the target follows the exact same motion paths as in Chapter 5.

Here, the performance charts show that BSTT-EI still relatively outperforms the baseline and benchmark algorithms. One can clearly see that BSTT-EI-ENI does not perform as well as BSTT-EI as the domain gets larger, even for a fixed motion path. Hence one can see that the way we have used negative information may only be effective in smaller domains.

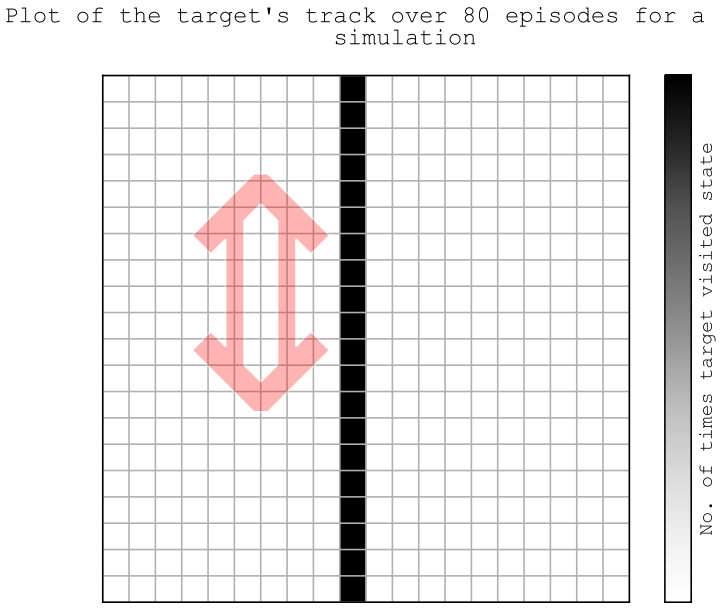


Figure I.1: This figure shows a sample trace for a target moving with a fixed linear motion path. Here, the target follows a fixed, linear motion path. The target begins at the top of the grid, and then moves down to the bottom of the grid at a pseudo-velocity of 1 location per episode. Thereafter, it moves back to the top of the grid at a pseudo-velocity of 1 location per episode. The target repeats this cycle until the end of the simulation.

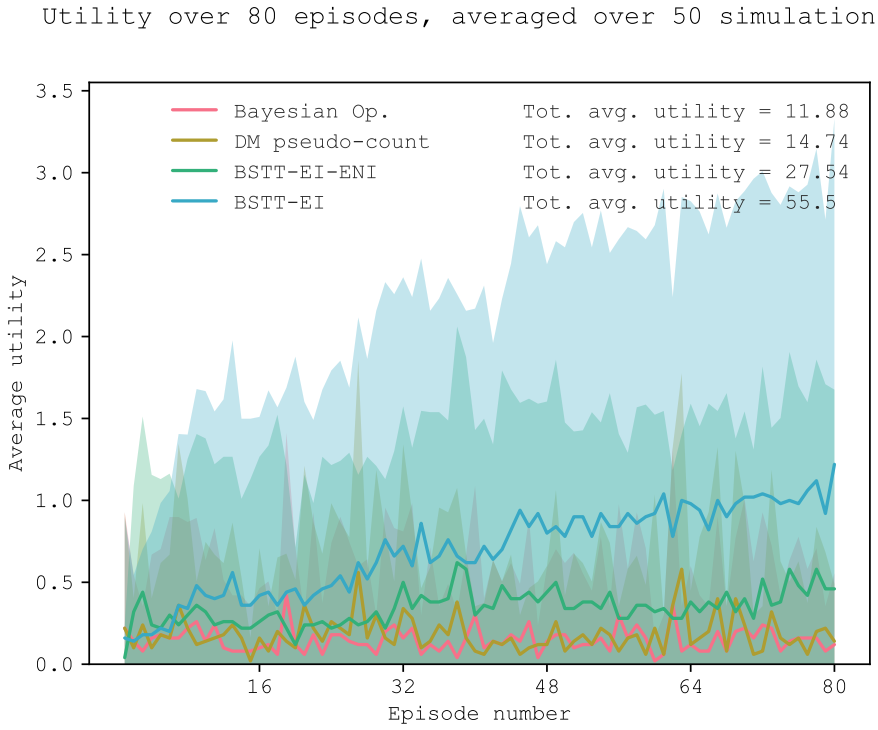


Figure I.2: Here, the target follows a fixed, linear motion path on a 20×20 grid. Shaded regions represent ± 1 standard deviation from the average utility. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

Appendix J

Experimental results with a reduced positive information threshold

The experiments in Section 5.2 are carried out when BSTT-EI and BSTT-EI-ENI have reduced positive information thresholds. Here, the positive information threshold is reduced from 0.5 to $\frac{1}{100} = 0.01$. When the positive information threshold is reduced to 0.01, then the drone will always observe positive information after a search. Both BSTT-EI and BSTT-EI-ENI will infer the target to be in the location with the greatest belief over it. If there are multiple locations with the greatest belief, then one of them will be selected randomly.

Figures J.1 and J.2 show the results of the experiments in Section 5.2, when the positive information threshold is reduced. The first point to note is that BSTT-EI and BSTT-EI-ENI show similar performance. The reason is because when the positive information threshold is reduced to 0.01, every search trial will result in positive information being observed, hence the mechanics of BSTT-EI and BSTT-EI-ENI are the same. Recall from Section 4.3 that BSTT-EI-ENI only differs from BSTT-EI when there is no positive information observed after a search.

The second result to note is that without the use of the positive information threshold, BSTT-EI shows superior performance when compared to the results in Section 5.2. The implications are that the use of thresholds to infer the target's location (as described in Section 3.3.5) may be unnecessary. When the results of this section are considered in conjunction with the results of Sections 5.4 and 5.5, which show that BSTT-EI-ENI is only effective in very contrived scenarios, one can see that the use of BSTT-EI with a low positive information threshold gives the best performance.

Note how the reduction of the positive information threshold to 0.01 results in the method for inferring the location being the same as the maximum a posteriori method (see Section 3.3.5), which is more ubiquitous [23]. Hence the results indicate that BSTT-EI is effective when using the maximum a posteriori method to infer the target's location from the belief.

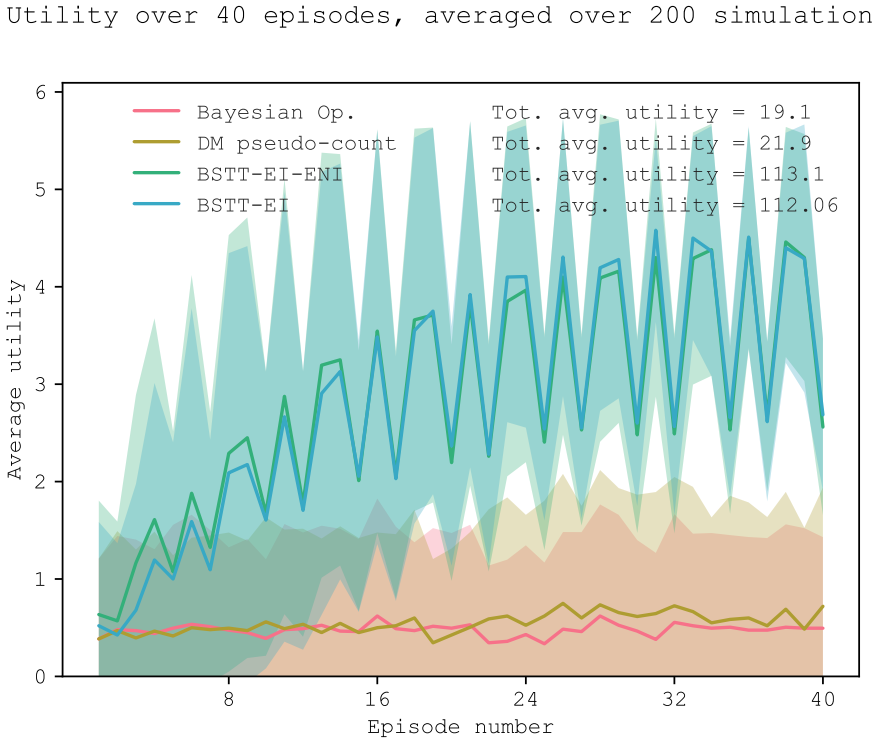


Figure J.1: The results of the experiment in Section 5.2.1 when the positive information threshold is reduced to 0.01. Shaded regions represent ± 1 standard deviation from the average utility. The maximum reward achievable in each episode is 5. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

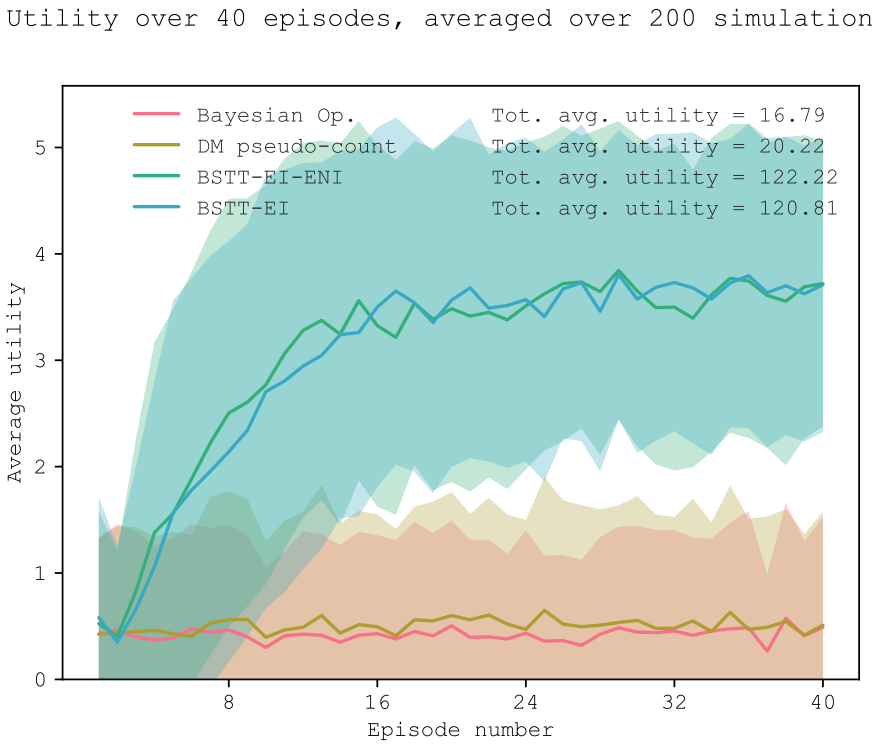


Figure J.2: The results of the experiment in Section 5.2.2 when the positive information threshold is reduced to 0.01. Shaded regions represent ± 1 standard deviation from the average utility. The maximum reward achievable in each episode is 5. Average utility is the average utility received in each episode, over all simulations. Total average utility is the sum of the average utility over all episodes.

Bibliography

[1] Stefano V Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.

[2] B Amutha and M Ponnavaikko. Energy efficient hidden markov model based target tracking mechanism in wireless sensor networks 1. 2009.

[3] Javed Aslam, Zack Butler, Florin Constantin, Valentino Crespi, George Cybenko, and Daniela Rus. Tracking a moving object with a binary sensor network. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 150–161. ACM, 2003.

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

[5] Tatiana Bokareva, Wen Hu, Salil Kanhere, Branko Ristic, Neil Gordon, Travis Bessell, Mark Rutten, and Sanjay Jha. Wireless sensor networks for battlefield surveillance. In *Proceedings of the land warfare conference*, pages 1–8, 2006.

[6] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[7] Richard R Brooks, Parameswaran Ramanathan, and Akbar M Sayeed. Distributed target classification and tracking in sensor networks. *Proceedings of the IEEE*, 91(8):1163–1171, 2003.

[8] Gavin E Crooks. Estimating probabilities from counts with a prior of uncertain reliability.

[9] Michael Duff and Andrew Barto. Optimal learning: Computational procedures for bayes-adaptive markov decision processes. *University of Massachusetts Amherst*, 2002.

[10] Asmaa Ez-Zaidi and Said Rakrak. A comparative study of target tracking approaches in wireless sensor networks. *Journal of Sensors*, 2016, 2016.

[11] Jason A Fuemmeler and Venugopal V Veeravalli. Smart sleeping policies for energy-efficient tracking in sensor networks. In *Networked Sensing Information and Control*, pages 267–287. Springer, 2008.

[12] U Haneback, O Schrempf, P Krauthausen, P Ruoff, I Gilitshenski, M Dolgov, and D Frisch. *Stochastische Informationsverarbeitung*. Karlsruhe Institute of Technology, 2017. Available upon request from Karlsruhe Institute of Technology.

[13] Majd Hawasly. Policy space abstraction for a lifelong learning agent. 2014.

[14] David Hsu, Wee Sun Lee, and Nan Rong. A point-based pomdp planner for target tracking. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2644–2650. IEEE, 2008.

[15] Daniel Jurafsky and James Martin. Hidden markov models, September 2018.

[16] Wooyoung Kim, Kirill Mechitov, J-Y Choi, and Soo Ham. On target tracking with binary proximity sensors. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 301–308. IEEE, 2005.

[17] Vikram Krishnamurthy. Algorithms for optimal scheduling and management of hidden markov model sensors. *IEEE Transactions on Signal Processing*, 50(6):1382–1397, 2002.

[18] X Rong Li and Vesselin P Jilkov. Survey of maneuvering target tracking: Iii. measurement models. In *Signal and Data Processing of Small Targets 2001*, volume 4473, pages 423–447. International Society for Optics and Photonics, 2001.

[19] Roman Marchant, Fabio Ramos, Scott Sanner, et al. Sequential bayesian optimisation for spatial-temporal monitoring. In *UAI*, pages 553–562, 2014.

[20] Roman Marchant Matus. Bayesian optimisation for planning in dynamic environments. 2015.

[21] Kevin P Murphy. A survey of pomdp solution techniques. *environment*, 2:X3, 2000.

[22] Benjamin Noack, Florian Pfaff, Marcus Baum, and Uwe D Hanebeck. State estimation considering negative information with switching kalman and ellipsoidal filtering. In *Information Fusion (FUSION), 2016 19th International Conference on*, pages 1945–1952. IEEE, 2016.

[23] Massachusetts Institute of Technology. Estimation techniques, March 2006.

[24] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[25] Benjamin Rosman, Majd Hawasly, and Subramanian Ramamoorthy. Bayesian policy reuse. *Machine Learning*, 104(1):99–127, 2016.

[26] Benjamin Rosman and Subramanian Ramamoorthy. What good are actions? accelerating learning using learned action priors. In *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*, pages 1–6. IEEE, 2012.

[27] Matthijs TJ Spaan. Partially observable markov decision processes. In *Reinforcement Learning*, pages 387–414. Springer, 2012.

[28] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[29] Gabriel A Terejanu. Discrete kalman filter tutorial. *University at Buffalo, Department of Computer Science and Engineering, NY*, 14260, 2013.

[30] Serdar Tugac and Murat Efe. Radar target detection using hidden markov models. *Progress In Electromagnetics Research*, 44:241–259, 2012.

[31] Ingmar Visser. Seven things to remember about hidden markov models: A tutorial on markovian models for time series. *Journal of Mathematical Psychology*, 55(6):403–415, 2011.

[32] David Wingate, Noah D Goodman, Daniel M Roy, Leslie P Kaelbling, and Joshua B Tenenbaum. Bayesian policy search with policy priors. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22(1), page 1565, 2011.

[33] W Murray Wonham. Some applications of stochastic differential equations to optimal nonlinear filtering. *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, 2(3):347–369, 1964.