HYPERFILTERING FOR STOCHASTIC SYSTEMS

BY

JAMES C. DAVIDSON

B.S., University of New Mexico, 2002

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2007

Urbana, Illinois

Adviser:

Professor Seth Hutchinson

ABSTRACT

Information-feedback control schemes (more specifically, sensor-based control schemes) select an action at each stage based on the sensory data provided at that stage. Since it is impossible to know future sensor readings in advance, predicting the future behavior of a system becomes necessary. Hyperfiltering is a sequential method that enables probabilistic evaluation of future system performance in the face of this uncertainty. Rather than evaluating individual sample paths or relying on point estimates of state, hyperfiltering maintains at each stage an approximation of the full probability density function over the belief space (i.e., the space of possible posterior densities for the state estimate).

Hyperfiltering has potential applications ranging from simulating a policy to garner insight on its performance, to being used to aid in generating nearly optimal control policies. In many problems, the policy is given for a system for which the performance is to be analyzed. In other problems policy search is used as a tool to find nearly optimal policies. Hyperfiltering can be used in such cases to simulate the effect of policies to determine their performance and, when optimizing, selecting the policy that performs best. Moreover, as a representation of all future uncertainty at a given stage, hyperfiltering may be used to evaluate the ability of the information space to be utilized to achieve a given objective.

To alleviate the inherent difficulty of generating the exact hyperfilter, especially for discrete state and observation space problems, the hyper-particle filtering algorithm is presented. The hyper-particle filter is built on the concept of the particle filter. However, the hyper-particle filter evolves a system forward one future stage to the next

for unknown observations and unactualized controls, unlike the particle filter, which evolves the system from a previous stage to the current stage for an actualized control and observation.

Numerous simulations of the hyper-particle filtering approximation method are performed and presented, demonstrating the performance of the method over a variety of parameters. The results indicate that the method converges quickly, in both mean and standard deviation, as the number of samples representing the probability function over the belief space increases. It appears that very few hyper-particles are needed, relative to the number of observations and iterations, to achieve an accurate analysis of the system. Additionally, the simulations verify the computational complexity as being $O(Kqn)$, where K, is the desired time horizon, n is the number of samples representing the probability function over the belief space, and q is the number of samples representing the approximated belief.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$\mathbf{w}$: a random variable

w: an event of $\mathbf{w}$

$p(w)$: $p_{\mathbf{w}}(w)$

k: stage of the process

K: time horizon

$\mathbf{x}_k$: state r.v. at stage k

$\mathcal{X}$: state space (or sample space) of $\mathbf{x}_k \in \mathcal{X},\ \forall k$

$\mathbf{y}_k$: observation r.v. at stage k

$\mathcal{Y}$: observation space (or sample space) of $\mathbf{y}_k \in \mathcal{Y},\ \forall k$

u_k: control action at stage k

$\mathcal{U}$: control space

I_k: information state at stage k

$\mathcal{I}_k$: information space, for which $I_k \in \mathcal{I}_k,\ \forall k$

Δ^n: n-dimensional simplex

b_k: belief state at stage k, whereby $b_k \triangleq p_{x_k|I_k}$

$\mathcal{P}_b$: belief space, for which $b_k \in \mathcal{P}_b,\ \forall k$

$B(\cdot)$: belief transition function

π: control policy

β_k: hyperbelief at stage k, whereby the hyperbelief denotes a probability function over $\Delta^{|\mathcal{X}|-1}$

$\mathcal{P}_\beta$: hyperbelief space, for which $\beta_k \in \mathcal{P}_\beta, \ \forall k$

$\Phi(\cdot)$: the hyperbelief transition function

s_k^i: the i'th particle sample at stage k

$\mathcal{S}_k$: the set of particles, $\mathcal{S}_k = \{s_k^i\}$ at stage k

z_k^j: the j'th hyper-particle sample at stage k

$\mathcal{Z}_k$: the set of hyper-particles, $\mathcal{Z}_k = \{z_k^j\}$ at stage k

1 INTRODUCTION

As the field of robotics developed in the 1980s, it branched from the studying the physical description of systems (via kinematics, inverse-kinematics, and dynamical descriptions; e.g., [1–3]) to that of automated systems. Automation endows physical systems with the capability to perceive their environment and reason about their own actions (e.g., [2–7]). Soon after, it became apparent that uncertainty would be a fundamental barrier to the effective operation of autonomous robot systems.

There are essentially three kinds of uncertainty that afflict a robot: uncertainty in representation of the robot's environment, uncertainty in the robot's knowledge of its own state, and uncertainty in the effects of the robot's actions. These sources of uncertainty are particularly relevant when the robot attempts to do the following

- Estimate its current state – State Estimation or Localization
- Construct a representation of its environment – Mapping
- Construct a map, and localize itself in the map – Simultaneous Localization and Mapping (SLAM)
- Determine the best action or sequence of actions – Control or Planning

One of the most researched areas for systems subject to uncertainty is the localization problem. Early work focused on just passive localization. Within the robotics community many approaches to passive localization were developed for both deterministic and nondeterministic uncertainty including [8–12]. Realizing the influence actions

may have on localization, passive localization inspired active localization, wherein motion strategies are chosen to better localize the robot [13–17]

While significant research was being performed in the area of localization, several researchers realized the assumption of a known environment was unreasonable in many cases. The importance of both mapping and SLAM became apparent and work began on these more complicated problems. Early work (e.g., [18–21]) was based on a linear Gaussian assumption. An explosion of research on SLAM began in the late 1990s (e.g., [22–31]). Many of these methods focus on finding more efficient implementations assuming linear Gaussian systems and using Kalman (or Kalman-like) filtering or the extended Kalman filter.

For applications like localization and SLAM, as well as others, researchers were often concerned with feasibility and optimality in the face of uncertainty. Well before robotics became concerned with uncertainty, researchers in control theory and communications had already begun investigating its effects. Some of the early work pertained to the regulation of linear quadratic systems subject to Gaussian noise (as outlined in [32]). Analysis of these systems was quickly expanded to both discrete and continuous time systems subject to bilinear noise (e.g., [33–43]).

One weakness of such control theoretic approaches was the assumption of an unconstrained space. Constraints, either workspace, configuration, semantic, or state space, are typical in most robotic applications. As such, robotic researchers approached analysis of uncertain systems with these constraints in mind. Not surprisingly, the earliest robotics research to consider uncertainty was done in AI labs using symbolic constraint manipulation techniques originally developed for manipulating systems with constrained variables (e.g., [44, 45]) to evaluate the effectiveness of robot plans under worst case scenarios (e.g., [46, 47]). Within a short time, a number of additional tools were introduced to cope with uncertainty in a robot's representation of its own state including probabilistic methods, such as the Kalman filter (e.g., [18, 48–50]) and geo-

metric methods based on bounded uncertainties (e.g., [51–55]). These tools brought to bear the power of sensing by explicitly defining models for the uncertainty in sensor data and by developing process models that quantified the effects on uncertainty in the robot's actions. Most recently, Bayesian methods have been fused with computational tools such as particle filters (e.g., [56–67]) to provide a general framework for dealing with uncertainty in a wide range of robotics applications.

Ultimately, each of the above approaches attempts to answer the question of how to predict the effect of uncertainty on the future behavior of a stochastic system. To address this question the hyperfiltering method is introduced. Hyperfiltering is a forward sequential technique, whereby the predicted behavior and its uncertainty at one stage is used to evaluate the predicted behavior and its uncertainty at the next stage. In this way the predicted behavior and its uncertainty are propagated between stages.

Current methods either perform sample path simulation (as used in [68–75]), whereby a series of sample paths are generated and the ensemble average is taken to estimate the behavior, or the observation process is discarded entirely as in [76–78]. In such approaches, the behavior of the system is predicted without taking the effect of observations into consideration. Alternatively, the behavior is just predicted one stage into the future using forward projection techniques (refer to [79]). None of these techniques are adequate, especially when attempting to fully evaluate, for more than one stage into the future, how the robot behaves from one future stage to the next. In such situations the observations can have a dramatic impact on the evolution on a robotic system. When considering information-feedback policies, observations play a direct role in determining applied actions. Moreover, the observation itself molds the probability function over the state space. It is therefore critical that the full effect of the future observations be considered. Hyper-particle filtering has been developed to not only predict the behavior of a system but also to be used in the planning process.

Hyperfiltering is a technique melding the concept of forward projection with the concept of filtering. Filtering propagates the behavior of the system and its uncertainty to the *current stage* for some known observations. Forward projection, on the other hand, propagates the predicted behavior of a system and its uncertainty forward from the current stage to the next *future stage*. Unfortunately, forward projection for partially observed stochastic systems for more than the next stage has not been well defined. This limitation is overcome by the formulation of the hyperfilter and the hyper-particle filter approximation method.

By maintaining a complete representation of the behavior from one stage to the next, hyperfiltering has the capability to evaluate the complete effect of a policy from one stage to the next, an ability sample path simulation lacks. Such functionality may prove useful in generating policies as the effect on the performance from one stage to the next can be explicitly evaluated using hyperfiltering. Hyperfiltering may be particularly useful when utilizing local planners to determine event-driven control policies. In such scenarios, the effect of a policy is evaluated at each stage to determine if the policy succeeds and, thus, hyperfiltering can used to verify whether the policy is capable of directing the robot to trigger one event from another.

The hyperfiltering method will be introduced in Chapter 4. However, first the background concepts relating to hyperfiltering and the potential applications are outlined in Chapter 2. Filtering, in the traditional sense, will be reviewed in Chapter 3 to build the basis for the hyperfiltering method. Because of burdens such as the exact evaluation of the hyperfilter at each stage and its computation complexity (or worst case running time), the hyper-particle filter, a hyperfiltering approximation method, is also introduced in Chapter 4. Examples of the hyper-particle filter will be demonstrated in Chapter 5. The document will then conclude with some final remarks and comments in Chapter 6.

2 BACKGROUND

To aid in the development of the formulation and explanation of the concept of hyperfiltering, relevant background material will be presented before hyperfiltering is formally introduced in Chapter 4. Hyperfiltering is a method for representing the effect of uncertainty in robotic systems. Uncertainty is present in many real-world robotic systems, from motion noise to sensor noise, and the presence of these uncertainties can plague the motion and sensing of many real-world robotic systems. By modeling the uncertainty and considering robotic systems as stochastic processes, it is possible to better design and simulate the evolution of these systems in an attempt to understand, alleviate, and/or anticipate the effect of noise in such systems. First, the probabilistic formulation of the uncertainty in the process and sensing model will be presented in Section 2.1 followed by the description of the specific class of system of interest in Section 2.2.

2.1 Probabilistic Description

Consider a system modeled as a stochastic process so that the state of a robot at a given stage k is represented by the random variable (r.v.) $\mathbf{x}_k \in \mathcal{X}$, where the finite set $\mathcal{X}$ is the state space of the system. The system evolution from stage $k-1$ to the next stage k is influenced by a control input, or action, $u_{k-1} \in \mathcal{U}$, where the control space $\mathcal{U}$ is a finite set of possible controls. The probability of the system being in state x_k at the current stage k given the previous state x_{k-1} and applied action u_{k-1} is given by $p(x_k|x_{k-1}, u_{k-1})$ and is referred to as the *transition probability function.*

Oftentimes the state of a robotic system is not directly known, or knowable, therefore sensing is performed to to elicit information that may indirectly pertain to the state of the system. Sensors are used to make an observation in such cases. The observation is subject to stochastic noise, so that $y_k \in \mathcal{Y}$ at stage k is a r.v., with $\mathcal{Y}$ the observation space. The probability of a given observation y_k occurring given the system is in state x_k can be evaluated via the *observation probability function*: $p(y_k|x_k)$. Since the state cannot be uniquely recovered from the observation y_k, such systems are referred to as partially observable.

One inherent property of processes defined with the transition probability function and observation probability function, as given above, is that these systems are Markovian. A sequence of r.v.'s is Markovian when the next stage depends only on the current state and not on the past.

Definition 2.1. *A sequence of random variables* $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ *defined over the sample space* $\mathcal{X}$ *is a Markov process if*

$$p(x_k|x_1, x_2, \cdots, x_{k-1}) = p(x_k|x_{k-1}),\ \forall k \text{ and } \forall x_k, x_{k-1} \in \mathcal{X}.$$

A Markov process may be conditioned on other variables. For instance $p(x_k|A, x_{k-1})$ $= p(x_k|A, x_1, x_2, \cdots, x_{k-1})$ if x_k is independent of $x_{k-2}, \cdots, x_1$, when conditioned on x_{k-1} and A. Note that a discrete time Markov process is referred to as a *Markov chain* (MC). The attention of this research will be restricted to a specific subclass of Markov Chains: those that are *time-invariant* or *time-homogeneous*. A time-invariant MC implies that $p_{\mathbf{x}_{k+1}|\mathbf{x}_k}(x|x_k) = p_{\mathbf{x}_k|\mathbf{x}_{k-1}}(x|x_{k-1})$, for all $x_k = x_{k-1}$ and for all k.

MCs represent the bulk of the stochastic systems of interest within the field of robotics. By choosing a state space representation, the Markov property can be invoked. This is the case for the transition probability function and the observation probability function as defined above. As can be seen from the definition of these func-

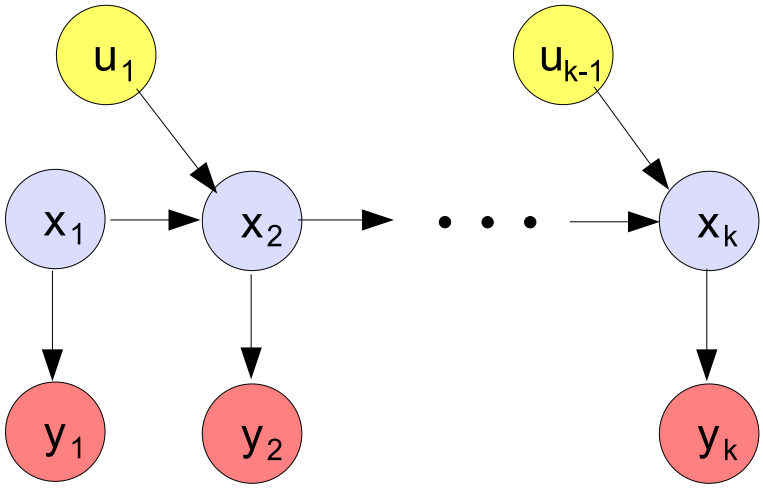


Figure 2.1: Dependency graph of a partially observed system defined by the observation probability function and the transition probability function

tions (and further illustrated by Figure 2.1), the probability function over the state for the current stage depends only on the control and the state of the previous stage, while the probability function over the observations depend only on the current state. Fortunately, Markov processes have many desirable properties that will be taken advantage of when deriving the hyperfilter and the hyper-particle filtering approximation technique in Chapter 4.

As a system evolves from one stage to the next, it generates a sequence of r.v.'s $\{\mathbf{x}_k\}_{k=1}^{K}$. Because the state is only indirectly observed through the observations, the system state is expressed as a probability function conditioned on the set of observations and past actions.

Definition 2.2. *The information state, I_k, at time k is the set of known information from the initial stage up to stage k. More precisely,*

$$I_k = \{y_1, u_1, y_2, u_2, y_3, \cdots, u_{k-1}, y_k\},$$

where $u_1, \cdots, u_{k-1}$ are the sequenced set of actions executed up to time k, and $y_1, \ldots, y_k$ are the sequenced set of observations made. The total information known up to time k is thus encapsulated in the information state.

The *information space* $\mathcal{I}_k$ is the set of all possible information states at time k. The initial probability function over the state, as represented by $p_{\mathbf{x}_1}$, is assumed to be given. This is necessary as filtering propagates the probability from one stage to the next and, therefore, the initial probability function must be given.

At every stage k, $p_{\mathbf{x}_k|I_k}$ is is the conditional probability function for the state given the information state I_k. As a notational device, the concept of a *belief* is used to represent this conditional probability function. Relying on $\mathbb{R}_+ = \{x \in \mathbb{R} | x \geq 0\}$, the belief can be define as follows:

Definition 2.3. *The belief b_k at stage k is function such that $b_k : \mathcal{X} \to \mathbb{R}_+$, whereby*

$$b_k \triangleq p_{\mathbf{x}_k|I_k}. \tag{2.1}$$

This notation is pervasive in the literature (refer to [79]) and is adopted because, later in Chapter 4, the formulation of the hyperfilter will be made more clear by using this notation because the hyperfilter is a functional (a functional's domain is a space of functions) of the belief.

The *belief space* $\mathcal{P}_b$ is the set of all possible beliefs for a given system. For discrete state space systems, where the belief state has $|\mathcal{X}|$ states, the belief space can be represented by a $|\mathcal{X}| - 1$ dimensional simplex $\Delta^{|\mathcal{X}|-1}$.

Definition 2.4. *An n-simplex is a subset of $\mathbb{R}^{n+1}$ given by*

$$\Delta^n = \{(x_1, x_2, \ldots, x_{n+1}) \ : \ \sum_{i=1}^{n+1} x_i = 1 \ \textit{and} \ x_i \geq 0 \ \forall i\}.$$

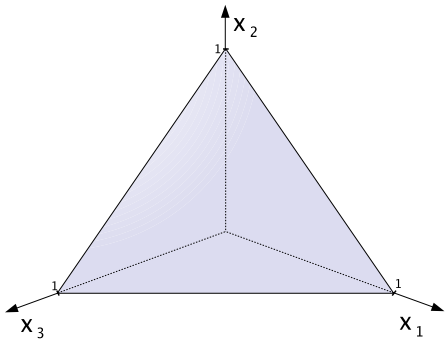


Figure 2.2: The 2-simplex

Table 2.1: Types of stochastic systems

	Uncontrolled	Controlled
Directly Observable	MC	MDP
Partially Observable	HMM	POMDP

An n-simplex is essentially a polytope that is an n-dimensional analogue of a triangle. The 2-simplex is illustrated in Figure 2.2. The belief, being the conditional probability function over the state space given the information state, is a representation of the uncertainty of the state of the system. Finding the belief, or an approximation of the belief, at every stage is the focus of most filtering methods (e.g., [48, 80–87]). By evaluating the belief at each stage as observations occur, filtering determines the behavior of the system as it evolves from the first stage to the current stage. However, filtering is applied as observations are received; filtering is not a method to predict the behavior into future stages for unknown observations.

There exist a number of possible types of Markov models depending on whether or not the system is fully observable or partially observable and whether the system has control inputs. The taxonomy of stochastic systems is as described in Table 2.1. The uncontrolled and fully observed processes are referred to generically as Markov Chains. However, partially observed uncontrolled Markov processes are known as hidden Markov models (HMM). Markov decision processes (MDPs), or controlled Markov chains, describe Markov processes where the state is fully observable and the process

has control inputs. For MDP systems the state is immediately known at the beginning of each stage and only the probability function that describes the evolution from one state to the next for a given control is needed to describe the evolution of the system. Markov processes with control inputs where the state is only partially observable are commonly referred to as partially observable Markov decision processes (POMDP). All the various types of Markov processes can be derived as special cases of the most general model: the POMDP, which is the model of interest of this research.

2.2 POMDPs

2.2.1 POMDP formulation

As the most general model, POMDPs incorporate the possibility of incomplete and uncertain knowledge when mapping states to observations. Such a representation enables the modeling and analysis of systems where sensing is limited and imperfect.

POMDPs include at least the following components:

- The state space representing the finite set of states of the world: $\mathcal{X}$.
- The finite set of control actions that can be executed: $\mathcal{U}$.
- The transition probability function: $p_{\mathbf{x}_k|\mathbf{x}_{k-1},u_{k-1}}$. This represents the likelihood of the system being in one state and transferring into another state at stage k given the applied action at stage $k-1$.
- The set of all possible observations: $\mathcal{Y}$. The observations represent the information received by the sensors.
- The observation probability function: $p_{\mathbf{y_k}|\mathbf{x}_k}$. The observation probability function represents the likelihood of a particular observation occurring given the system is in a specified state.

In addition, a POMDP may be specified with a reward function $r(\cdot)$, which defines the objective to be optimized for the POMDP.

Understanding the evolution of POMDP systems requires a control policy or a set of applied actions. The control policy essentially is the function that directs the motion of the robot. Evaluating the behavior is the subject of filtering methods. For sequential stochastic systems, filtering refers to the process where the probability function over the state is estimated at each stage. By evaluating the probability function from one stage to the next, the likelihood of a state is determined while at the same time the unlikely states are "filtered" out. Filtering will be discussed in more detail in Chapter 3. While the work presented does not specifically deal with planning, a control policy is needed to evolve the system forward into future stages, which is the primary goal of the hyperfilter. A brief introduction to the determination of control policies and an outline of various methods to determine control policies is therefore given below in Section 2.2.2.

2.2.2 Control policies for POMDPs

Ultimately, the goal of much of robotics research is to engineer autonomous or nearly autonomous systems. Within the context robotics, control theory, and AI this concept of autonomy comes to fruition via the control policy $\pi(\cdot)$. A control policy maps input to control actions. If the control policy depends only on time, or $\pi : \{i\}_1^K \rightarrow \mathcal{U}$, it is referred to as *open-loop policy*. Alternately, a closed-loop/feedback policy takes information acquired as the system evolves, such as an estimate of the state, the previously applied actions, and so forth when determining which control action to apply. When the control policy takes as input the entire information state, or $\pi : \mathcal{I}_k \rightarrow \mathcal{U}$, it is referred to as an *information-feedback policy*.

In fact, the belief b_k is a sufficient statistic for the information state as shown in [88] (and subsequently in [32]). The belief b_k is a sufficient statistic in that control policies that depend on b_k are as capable of obtaining the same solution as those that depend on the information state. Most optimization techniques rely directly on b_k because it encapsulates the information state and has a finite and constant dimensional representation, whereas the dimension of information state grows at each stage. Because of the ability to sequentially evaluate the belief, b_k, for the current stage given b_{k-1} of the previous stage, the concept of the belief fits naturally into a dynamic programming (DP) framework (see [32] for an in-depth discussion about DP).

Finding a control policy is often a difficult task when attempting to achieve a specific objective. There are two types of objectives: feasibility and optimality. A feasibility-based objective is a binary function that specifies whether or not a given goal has been satisfied, such as when the robot has reached some specified location or region in the state space. An optimality-based objective quantifies the quality of a solution. For systems with feasibility-based objectives, any policies that satisfy the feasibility-based objective functions are acceptable, whereas only the optimal policies are desired for systems with optimality-based objectives.

Methods to solve systems with feasibility-based objectives are often techniques where the control policy is given and the method evaluates the control policy to determine whether or not it satisfies the objective. In other methods, a control policy is generated via an exploratory technique by searching a subset of the possibilities. The subset of possibilities may comprise a subset of the control policies or a subset of intermediate goal states that the policies attempt to reach. In such scenarios, the policy is alternately modified and evaluated until the objective is met. Feasibility-based methods are rarely pointwise methods. Pointwise methods require that a control policy be evaluated and determined for every point in the space.

The two dominant methods for finding policies associated with optimality-based methods are value iteration, a DP approach, and policy iteration. Both of these techniques are pointwise techniques that evaluate over the entire set of possibilities. Value iteration operates backwards from the terminal time to the initial time. At each stage value iteration finds the optimal policy from the current stage to the next stage. This repeats until the initial stage is reached and the optimal solution is found. Policy iteration, on the other hand, operates by starting with an initial policy. Then at each iteration the method searches for any change to the policy from the previous iteration that improves the performance. Because both MDPs and POMDPs have only a finite number of policies, policy iteration will terminate after a finite amount of time.

Finding the optimal solution for MDPs is computationally reasonable for systems with a finite number of states and control actions; the solution is $O(K|\mathcal{U}||\mathcal{X}|^2)$, where K is the number of stages to be evaluated. The number of stages to be evaluated is typically referred to as the *time horizon*. Infinite horizon problems have a computational time complexity that generally polynomial in the desired error, number of states, and number of actions for infinite horizon problems [89, 90]. Partially observable systems, on the other hand, do not fare so well; they are intractable, with a best known computational time complexity that is exponential in both the time horizon and the number of observations [91]. However, finding the optimal policy for partially observed systems is desired for many real-world problems. Thus, many researchers have focused on finding efficient methods to solve POMDP models.

The typical objective is to find the optimal policy that maximizes the expected sum of rewards. Without any special structure, finding the optimal solution for a POMDP can be impractical, if not impossible, considering that the objective function can be a nonlinear function of the belief space. Fortunately, Smallwood and Sondik [92] established that POMDPs have a special structure when the cost is the expected sum of rewards; still, POMDPS are intractable even with this special structure.

The majority of work relating to POMDPs maximizes the expected sum of rewards for two classes of problems: finite time horizon and discounted infinite time horizon. The formulation for the discounted infinite time problem includes a strict discount factor applied at each stage that reduces the effect of the future cost on the system. Because of strict discounting, it is possible to establish a bound on the the approximation error for a given time horizon [91]. Many techniques were developed to find more efficient exact solutions to the POMDP problem, such as [91, 93–98]. However, since each of these techniques are intractable, many researchers turned their focus to finding approximate solutions.

Approximation techniques can focus on reducing the complexity in one or both of the terms, causing the exponential growth in computational time complexity: the dimension of the belief space and the length of the time horizon. Most approximation methods focus on the reduction of the complexity arising from the dimension of the belief state, b_k. One area of research attempts to mitigate the effect of the belief state dimension by factorizing the system as done in [68,69,99]. As done in [73, 100], the belief space can be compressed into a representative subspace. Another class of approximation methods focuses on sampling the reachable beliefs and limiting the complexity of the representation in cost at each stage (e.g., [70–72, 74, 75, 101]). The methods described in [100, 102–105] simplify the problem by representing the policy as a finite state machine. The problem may also be simplified through the use of limited information, such as is explored in [106].

Each of these approximation techniques has at least one thing in common: the set possibilities that are evaluated is a subset of the complete set of possibilities that must be analyzed to find an exact solution. This is the case whether a limited subset of sample beliefs are evaluated or a limited subset of policies are searched. In such situations, hyperfiltering may offer a tangible benefit in predicting the evolution of the system for

approaches where local policies are used to plan between the sets of possibilities. This will be discussed more thoroughly in Chapter 6.

3 STOCHASTIC FILTERING

For systems subject to uncertainty, *filtering* describes the process whereby an estimate of the state and its uncertainty are propagated from one stage to the next. Filtering is a sequential or recursive method whereby an estimate from the previous stage is used to determine an estimate for the current stage. *Hyperfiltering* is also a sequential method, but, while filtering evolves an estimate of the state and its uncertainty from previous stages to the current stage, hyperfiltering propagates the uncertainty and estimate of a system forward into future stages. The concept of filtering is presented as well as the relevant background material needed to motivate the development of hyperfiltering. Filtering is formulated in Section 3.1 and is followed, in Section 3.2, by a taxonomy of filtering approximation methods. The particle filter, the approximation technique on which the hyper-particle filter is based is presented in Section 3.3.

3.1 Formulation

Filtering is a general term for processing sequential systems that are either causal or noncausal, whereby the likelihood of the system being in a particular state is estimated or "filtered" from one stage to the next. In this way the unlikely states are "filtered" out. For stochastic systems, filtering refers to a sequential technique that generates an estimate of the state and uncertainty from one stage to the next. Conveniently, filtering minimizes the amount of information that must be retained from previous stages because only the previous stage is used to estimate the current stage. Having an estimate of the state and a representation of the uncertainty from one stage to the

next is immensely useful in finding and implementing robust and even optimal control policies.

When dealing with partially observable systems, a construct is needed to encapsulate the known information. Unfortunately, the state is only indirectly known through the information state I_k. The *information state* is an accumulation of all of the information about a system that is directly known. This includes the set of actions performed and the set of observations collected up to time k. The question is how to incorporate all of the information into an estimate of the state of the system and its uncertainty. While some filtering methods take advantage of certain properties of r.v.'s, the essential nature of filtering Markovian systems is best described by the precise method known as the Bayesian filter.

3.1.1 Bayesian filter

Because of Markov properties, Bayes rule can be applied to perform sequential filtering. This method, known as Bayesian filtering was first introduced in [88]. Bayesian filtering estimates the belief at the current stage k from the previous belief at stage $k-1$. The evolution of the belief can be split into two stages: prediction and update. The prediction stage takes the previous belief, $p_{\mathbf{x}_{k-1}|I_{k-1}}$, and pushes it through the transition probability function to obtain the predicted current belief, $p_{\mathbf{x}_k|I_{k-1},u_k}$. The prediction step evaluates the effect of an action on the belief of the system. This is achieved by simply marginalizing $p(x_k|I_{k-1},u_{k-1})$ on x_{k-1}, which becomes

$$p(x_k|I_{k-1},u_{k-1}) = \sum_{x_{k-1}\in\mathcal{X}} p(x_k|x_{k-1},I_{k-1},u_{k-1})p(x_{k-1}|I_{k-1},u_{k-1}) \quad (3.1)$$

$$= \sum_{x_{k-1}\in\mathcal{X}} p(x_k|x_{k-1},u_{k-1})p(x_{k-1}|I_{k-1}). \quad (3.2)$$

In (3.1) the current belief is marginalized on x_{k-1}. The transition probability function is independent of I_{k-1} given x_{k-1}. Therefore, (3.1) reduces to (3.2). Likewise, because the previous belief is conditionally independent of u_{k-1}, $p(x_{k-1}|I_{k-1}, u_{k-1})$ becomes $p(x_{k-1}|I_{k-1})$ as shown in (3.2). The current belief is now represented in terms of the previous belief and the transition probability function.

After the prediction stage, the update stage is executed. The update stage incorporates an observation to condition the belief on new information to generate $p_{\mathbf{x}_k|I_k}$. After applying Bayes rule, the system of interest becomes

$$\begin{aligned} p(x_k|I_k) &= p(x_k|y_k, I_{k-1}, u_{k-1}) && (3.3)\\ &= \frac{p(y_k|x_k, I_{k-1}, u_{k-1})p(x_k|I_{k-1}, u_{k-1})}{p(y_k|I_{k-1}, u_{k-1})} && (3.4)\\ &= \eta_k p(y_k|x_k)p(x_k|I_{k-1}, u_{k-1}) && (3.5) \end{aligned}$$

where η_k is the normalizing constant

$$\frac{1}{\eta_k} = \sum_{x_k \in \mathcal{X}} p(y_k|x_k)p(x_k|I_{k-1}, u_{k-1}). \tag{3.6}$$

In (3.3), $p(x_k|I_k)$ is rewritten so that the y_k and u_{k-1} are pulled out of I_k. As shown in (3.4), Bayes rule is applied so that the probability of y_k is conditioned on x_k. The probability of y_k is independent of other terms when conditioned on x_k, thus the other terms are eliminated in (3.5). Also in (3.5), $p(y_k|u_{k-1}, I_{k-1})$ is a normalizing constant that can be determined as the sum of the probability of y_k over all possible x_k as is shown in (3.6).

By combining both the prediction (3.2) and update (3.5), the Bayesian filter is obtained:

$$p(x_k|I_k) = \eta_k p(y_k|x_k) \sum_{x_{k-1} \in \mathcal{X}} p(x_k|x_{k-1}, u_{k-1})p(x_{k-1}|I_{k-1}), \tag{3.7}$$

where η_k is as defined by (3.6). From (3.7), it is shown that $p_{\mathbf{x}_k|I_k}$ can be evaluated directly from $p_{\mathbf{x}_{k-1}|I_{k-1}}$ and u_{k-1} using both the transition and observation probability functions. Likewise, if Bayes rule is applied to $p_{\mathbf{x}_{k-1}|I_{k-1}}$, the belief at stage $k-1$ requires only the belief and action at stage $k-2$. By continuing to expand (3.7), it becomes apparent that the Bayesian filter can be recursively applied up to the initial belief. This inductive step is crucial in understanding that the Bayesian filter can be sequentially applied to continuously evaluate the current belief given the previous belief.

Interestingly, by analyzing filtering from the belief perspective, the problem reduces to a deterministic function that transitions one belief into another. Thus, the belief at stage k can be determined from the belief at stage $k-1$.

Definition 3.1. *The belief transition function $B(\cdot)$, where $B : \mathcal{P}_b \times \mathcal{U} \times \mathcal{Y} \to \mathcal{P}_b$, transfers one belief b_{k-1} into another b_k given some particular action u_{k-1} and observation y_k,* or

$$b_k = B(b_{k-1}, u_{k-1}, y_k),$$

where, with $x_k(i) \in \mathcal{X}$ *for* $i = 1, \ldots, |\mathcal{X}|$ *representing the set of states,*

$$B(b_{k-1}, u_{k-1}, y_k) \triangleq \begin{bmatrix} \eta_k p(y_k|x_k(1)) \sum_{x_{k-1}\in\mathcal{X}} p(x(1)|x_{k-1}, u_{k-1}) b_{k-1}(x_{k-1}) \\ \eta_k p(y_k|x_k(2)) \sum_{x_{k-1}\in\mathcal{X}} p(x_k(2)|x_{k-1}, u_{k-1}) b_{k-1}(x_{k-1}) \\ \vdots \\ \eta_k p(y_k|x_k(|\mathcal{X}|)) \sum_{x_{k-1}\in\mathcal{X}} p(x_k(|\mathcal{X}|)|x_{k-1}, u_{k-1}) b_{k-1}(x_{k-1}) \end{bmatrix}$$

and

$$\frac{1}{\eta_k} = \sum_{x_k\in\mathcal{X}} \sum_{x_{k-1}\in\mathcal{X}} p(y_k|x_k) p(x_k|x_{k-1}, u_{k-1}) b_{k-1}(x_{k-1}).$$

The Bayesian filtering description derived above was for discrete space systems. The continuous state analog is generated by replacing the summations with integrals. Many robotic systems evolve over continuous spaces. The difficulty with extending from dis-

crete to continuous spaces is the lack of closed form solutions to the integral equivalent of (3.2). It becomes necessary to find tractable approximations to the Bayesian filter to proceed. Fortunately, there exist a plethora of techniques focused on approximating the Bayesian filter for both continuous and discrete systems.

3.2 Filtering Approximation Methods

Most general filtering problems have no known analytical solution or an unacceptable running time, also known as computational time complexity. Because of inherent difficulties in filtering for general systems, approximation methods are typically the only feasible choice. For discrete systems difficulties arise from the $O(|\mathcal{X}|^2)$ computational time complexity in the evolution from one stage to the next, where $|\mathcal{X}|$ is the number of states. This means the worst case running time grows quadratically in the size of the state space. Often for realistic systems, there can be millions of states, and the square of this quantity can make the exact derivation of the next belief state prohibitively expensive. Reducing the running time to linear or sublinear computational time complexity is often desired for such systems.

While computational time complexity can be an issue with continuous time systems, it is more often the lack of a known closed form solution that makes approximation techniques necessary. As an exception, the Kalman filter (KF) [48], the most popular method for continuous state systems, is an exact filter for linear Gaussian systems. Taking advantage of the property that a random variable that is the superposition of jointly Gaussian random variables is itself a Gaussian random variable, Kalman derived a formulation to evolve the mean and covariance describing the probability function. This method's popularity holds even to this day.

The extended Kalman filter (EKF) was developed in an attempt to expand the KF method to general nonlinear systems [80]. The EKF linearizes the system around

the current estimate and then applies the KF on the linearized system to update the estimated parameters. The EKF has become especially popular in SLAM applications.

Numerous alternative methods, based directly on Bayesian filtering, have been developed. To expand past the Gaussian limitation of the KF, the Gaussian mixture method [81] was developed for linear systems subject to non-Gaussian noise. The mixture method works by approximating a non-Gaussian probability function by a sum of Gaussian probability functions. A major drawback to this method is the possible exponential growth, in the time horizon, of the number of Gaussians representing the belief.

Researchers in [82] tackle the problem of nonlinear Gaussian systems with the unscented filter. The unscented filter samples a Gaussian to estimate the belief and then passes the samples through the transition probability function. Once completed, a new Gaussian probability function is generated to fit the newly evolved samples. Set-theoretic methods are used to filter systems when no model besides the support of the noise is known. These systems are evaluated using forward projection techniques [107]. The approach of Hanebeck [83] and Stump et al. [84] is to employ sequential elliptical approximations to evaluate the uncertainty sets.

While some research has occurred for the class of problems listed above, the majority of work has focused on solutions to general nonlinear, non-Gaussian systems. The vast majority of this work has been for parametrized family solutions, including [85–87, 108]. Several researchers have researched mixture methods for parametrized families (e.g., [109, 110]) that meld the parametrized solution with the Gaussian mixture method.

More recently, sampling-based methods have become popular. Sampling methods take a computational approach to solving the filtering problem, whereby at each stage a finite set of points is used to evaluate an approximation to the exact filtering outcome. A grid-based method introduced in [111] deterministically samples the sample space by using a grid-based approximation over the state space. For continuous systems, using

this approach makes it possible to reduce a possibly unknown analytical solution to an approximate computational solution. For discrete systems, sampling can reduce the computational time complexity significantly by considering only a limited number of the states in the state space. A major drawback of this deterministic approach is that the computational burden grows exponentially with the dimension of the state space. To alleviate this drawback, motivated by the convergence properties of random sampling, an alternative random sampling method known as particle filtering has become the defacto standard when filtering nonlinear, non-Gaussian systems. Particle filtering is used as a basis for an approximation for the hyperfilter and, for this reason, particle filtering is described in detail below.

3.3 The Particle Filter

Unlike deterministic sampling, e.g. grid based methods, the particle filter [56–67] randomly samples the space. Particle filtering is a sequential method that is simple to implement and has many desirable properties. Particle filtering is based on Monte Carlo Markov chain (MCMC) simulation, which is influenced by Monte Carlo integration. Unlike particle filtering, MCMC is an iterative method requiring the re-evaluation of all information at every stage [56]. At each stage k, MCMC methods sample a series of states from the initial stage to the current stage. The probability of the sample is then evaluated by simulating the system forward from the initial stage until the current stage k. At the next stage $k+1$, the MCMC methods, again, generate a new set of state samples for each stage from the initial stage to stage $k+1$. This process is an iterative technique whereby no information from the previous stage is retained. Iterative techniques have worst-case running times that are geometric in the number of states at each stage. Because of this, the entire computational time complexity burden for estimating the system from an initial stage to a given time horizon results in a computational

time complexity that is exponential in the time horizon. Particle filtering, on the other hand, is sequential in nature. At each stage the approximation of the current belief is evaluated from the approximation of the previous belief.

Particle filtering approximates the probability function by a finite set of samples instead of performing exact filtering. Each sample consists of a two elements: a weight (probability) and a point in the state space. Because the representation is discrete, the evolution through the Bayesian filtering equations becomes computational in nature. Hence, particle filtering allows for the evaluation of a broad class of nonlinear, non-Gaussian systems.

Particle filtering is known as bootstrap filtering [58], condensation [63], sequential Monte Carlo [64], interacting particle approximations [65], and survival of the fittest [66]. Regardless of the name, one of the benefits of particle filtering is that, under general conditions, the convergence or the error is defined in the number of samples, not the dimension of the state space. Furthermore, as was shown in [67] (and subsequently in [56]), particle filtering converges under fairly weak assumptions.

Unlike most derivations, the formulation of the particle filtering algorithm, to be presented in this section, is specifically separated into a prediction and an update step. The reason for this is to facilitate the adaptation of the particle filtering method into the hyper-particle filtering approach (see Section 4.3). Particle filtering can also be applied to discrete state systems as will be done when exploring the hyper-particle filter. The following derivation is based on the sequential importance resampling method (SIR) (see [112] for further description). There are numerous other adaptations to particle filtering that can be applied; however, the fundamental approach in all of these methods is the same.

Particle filtering approximates the probability function of a system with a finite set of particles $\mathcal{S} = \{s^i\}$. Each particle $s^i = (x_k^i, w_k^i)$ comprises as a point x^i in the state space $\mathcal{X}$ and a scalar weight w^i, where $0 < w^i \leq 1$ and $\sum_i w^i = 1$. At any given stage,

k, the belief of the system is approximated by the set of particles $\mathcal{S}_k = \{s_k^i\}$ as

$$p(x_k|I_k) \approx \sum_{i=1}^{|\mathcal{S}_k|} w_k^i \delta(x_k - x_k^i).$$

Particle filtering begins by taking m random samples of the initial belief and giving them equal weight of $1/m$. The primary portion of the method, repeated at each iteration, performs the approximated Bayesian filtering on the set of particles $\mathcal{S}_k$ for each stage k. Instead of calculating each possible future state from each current state, particle filtering works by randomly sampling a set of next states using an importance sampling function $q(\cdot)$. Because state and observation spaces are finite and the observation y_k is already known, it is possible to sample from the optimal choice of importance sampling function, which is

$$q(x_k|x_{k-1}^i, y_k, u_{k-1}) \triangleq p(x_k|x_{k-1}^i, y_k, u_{k-1}), \tag{3.8}$$

as was shown in [64]. The importance sampling function (3.8) can be expanded by applying Bayes rule to become

$$\begin{aligned} q(x_k|x_{k-1}^i, y_k, u_{k-1}) &= p(x_k|x_{k-1}^i, y_k, u_{k-1}) \\ &= \frac{p(y_k|x_k, x_{k-1}^i, u_{k-1})p(x_k|x_{k-1}^i, u_{k-1})}{p(y_k|x_{k-1}^i, u_{k-1})} \\ &= \frac{p(y_k|x_k)p(x_k|x_{k-1}^i, u_{k-1})}{p(y_k|x_{k-1}^i, u_{k-1})}, \end{aligned}$$

where the later equation follows from the conditional independence of y_k on previous states (i.e., x_{k-1}^i). Often $q(x_k|x_{k-1,}u_{k-1})$ is chosen as the importance sampling function because it is often simple to generate samples from x_{k-1} and u_{k-1}.

From the previous set of particle samples, $\mathcal{S}_{k-1} = \{(x_{k-1}^i, w_{k-1}^i)\}_{i=1}^m$, a set of new state samples $\{x_k^j\}_{j=1}^m$ are randomly generated using $q(x_k|x_{k-1,}u_{k-1})$ as the importance sampling function, where one sample x_k^j is generated for each x_{k-1}^i in $\mathcal{S}_{k-1}$. The weight,

$\hat{w}_k^j$, representing the probability of each new sampled state, x_k^j, is then determined to generate the new particle set $\mathcal{S}_k = \{(x_k^j, \hat{w}_k^j)\}_{j=1}^m$. The probability of *any* state $x_k \in \mathcal{X}$ at time k is obtained from the previous belief at stage $k-1$ and the transition probability function as

$$p(x_k|I_{k-1}, u_{k-1}) = \sum_{x_{k-1} \in \mathcal{X}} p(x_k|x_{k-1}, u_{k-1}) p(x_{k-1}|I_{k-1}) \tag{3.9}$$

$$\approx \sum_{i=1}^{m} p(x_k|x_{k-1}^i, u_{k-1}) w_{k-1}^i. \tag{3.10}$$

In (3.10), the exact predicted belief is approximated from the particle set at stage $k-1$. The weight w_{k-1}^i is the approximated probability of $p(x_{k-1}^i|I_{k-1})$ for each x_{k-1}^i in $\mathcal{S}_{k-1}$.

To simplify the analysis, particle filtering methods assume that the transition probability of each particle x_k^j is nonzero for only the sample x_{k-1}^i used to generate it. For discrete systems this approximation reduces the computational time complexity significantly. Taking this approximation into account, (3.10) reduces to

$$p(x_k^j|I_{k-1}, u_{k-1}) \approx \eta_p p(x_k^j|x_{k-1}^i, u_{k-1}) w_{k-1}^i,$$

where η_p is a normalizing constant. Because each sample x_k^j was sampled randomly from an importance sampling function, an adverse effect is introduced. Because the samples are generated from an importance sampling function and not the actual probability transition function, the random samples are not representative of the random samples that would be generated if the probability transition function was sampled directly and therefore the representation of the posterior probability function is skewed. Without taking into account this effect, the result can quickly become erroneous.

The issue is that, on one hand, if one samples from the transition function the weight should be identical for each sample, as the sampled set will approximate the probability function itself. However, if one sampled the space uniformly, each sample should be

weighted according to the probability of each sample. The question is then how to take into account the adverse effect when performing quasi-random or random sampling from a probability function other than the transition probability function. If each sample is just given equal weight, the approximation becomes that of the importance sampling function and not the transition probability function. However, if each sample is weighted only according to the transition probability function, the approximated probability function becomes erroneous. Imagine sampling from an importance sampling function that is a Gaussian centered in the state space. If the transition probability function has a low probability of occurring around the center of the state space so that each sample receives a low weight, the samples get assigned a higher weight when normalized and the set of samples are focused in the center of the space. The result is an inaccurate representation of the true probability function.

Particle filtering researchers rely on insights from Monte Carlo integration to deal with this issue. When approximating the expectation of some bounded function $c(\cdot)$ relative to some probability function $p(\cdot)$ by a randomly generated, finite set of samples, the set of samples generated can adversely effect the result. It turns out that this adverse effect can be eliminated by weighting each sample by the ratio of the probability of the sample being generated by the transition probability function divided by the probability of the sample being generated by the importance sampling function. More precisely, as observed in Monte Carlo integration, for some function $c(\cdot)$,

$$E[c(x_k)] = \sum_{x \in \mathcal{X}} c(x)p(x) = \sum_{x \in \mathcal{X}} c(x)\frac{p(x)}{q(x)}q(x).$$

The expectation of a r.v. with a probability function $p(x)$ can be represented as the expectation of another r.v. with the probability function $q(x)$ by weighting $c(x)$ by the ratio of $p(x)$ and $q(x)$ for each $x \in \mathcal{X}$. Thus, as can be seen, the adverse effect of the importance sampling on the expected value of any bounded function $c(\cdot)$ is

Algorithm 1 Particle filter prediction

1: $\hat{\mathcal{S}}$= PF Predict($\mathcal{S}$,u,y), where $\mathcal{S} = \{w^i, x^i\}_{i=1}^m$
2: **for** $i = 1, \cdots, m$ **do**
3: sample $\hat{x}^i$ from $q(\cdot \mid x^i, u, y)$
4: $\hat{w}^i \leftarrow w^i \frac{p(\hat{x}^i|x^i,u,y)}{q(\hat{x}^i|x^i,u,y)}$
5: **end for**
6: $\frac{1}{\eta_p} \leftarrow \sum_{i=1}^m \hat{w}^i$
7: **for** $i = 1, \cdots, m$ **do**
8: $\hat{w}^i \leftarrow \eta_p \hat{w}^i$
9: **end for**
10: $\hat{S} \leftarrow \{\hat{w}^i, \hat{x}^i\}_{i=1}^m$
11: $\hat{S} \leftarrow$PF_update($\hat{S}$,y)
12: return $\hat{S}$

eliminated. As the effect of the bias relative to any $c(\cdot)$ is attenuated, any measure over the probability function has the effect of the bias attenuated. By dividing by $p_{\mathbf{x}_k|\mathbf{x}_{k-1},u_{k-1}}$ by $q_{\mathbf{x}_k|\mathbf{x}_{k-1},u_{k-1},y_k}$, the expected adverse effect in (3.11) relative to any bounded function $c(\cdot)$ is therefore attenuated and the weight $\hat{w}_k^j$ associated with x_k^j becomes

$$p(x_k^j|I_{k-1}, u_{k-1}) \approx \eta_p \frac{p(x_k^j|x_{k-1}^i, u_{k-1})}{q(x_k^j|x_{k-1}^i, u_{k-1}, y_k)} w_{k-1}^i \tag{3.11}$$

$$= \hat{w}_k^j. \tag{3.12}$$

When the transition probability function is selected as the importance sampling function and is substituted into (3.11), the updated weight becomes $\hat{w}_k^i = w_{k-1}^i$. Thus the adverse effect is eliminated so that weight for each new particle is just the weight with the previous particle. The precise particle filtering prediction algorithm is given in Algorithm 1. Because the systems of interest are discrete, it is always possible to sample from $p_{x_k|x_{k-1},u_{k-1}}$ directly. Thus, the effect of sampling from an importance sampling function becomes moot. Each of the examples in Chapter 5 use the transition probability function directly as the importance sampling function.

The update procedure weights of each particles according to the probability of each sample given the observation y_k. With η_u, a normalizing constant, the new weight

Algorithm 2 Particle filter update

1: $\mathcal{S}$= PF_update($\hat{\mathcal{S}}$,y), where $\hat{\mathcal{S}} = \{\hat{w}^i, \hat{x}^i\}_{i=1}^m$
2: **for** $i = 1, \cdots, m$ **do**
3: $w^i \leftarrow \hat{w}^i p(y \mid \hat{x}^i)$
4: **end for**
5: $\frac{1}{\eta_u} \leftarrow \sum_{i=1}^m w^i$
6: **for** $i = 1, \cdots, m$ **do**
7: $w^i \leftarrow \eta_u w^i$
8: **end for**
9: $\mathcal{S} \leftarrow \{w^i, x^i\}_{i=1}^m$
10: estimate particle divergence of S
11: **if** particle divergence greater than threshold **then**
12: $\mathcal{S} \leftarrow$PF_resample(S)
13: **end if**
14: return S

becomes

$$\begin{aligned} p(x_k^i|I_k) &= \frac{p(y_k|x_k^i)p(x_k^i|I_{k-1}, u_{k-1})}{p(y_k|I_{k-1}, u_{k-1})} \\ &\approx \eta_u p(y_k|x_k^i)\hat{w}_k^i \\ &= w_k^i \end{aligned}$$

by using (3.12) as an approximation of $p(x_k^i|I_{k-1}, u_{k-1})$. The particle location does not change in the update step of the particle filtering algorithm. Instead, the weight is only amplified or attenuated. The update algorithm is described in Algorithm 2.

Once the new weight is generated, a resampling algorithm is executed. The resampling algorithm selects a set of m random samples from the belief approximated by the particle set. Resampling avoids particle degeneracy, which occurs when low weight, or low probability, particles continue to be utilized. Particle degeneracy can, and usually does, cause problems because higher probability regions are undersampled and lower probability regions are oversampled, resulting in a poor representation of the actual probability function. However, resampling has the potential side effect of causing particle impoverishment, whereby particles with high weights are selected many times. This

Algorithm 3 Particle filter resample

$\bar{S}$= PF_resample($\mathcal{S}$), where $\mathcal{S} = \{w^i, x^i\}_{i=1}^m$
Initialize cdf c to zero
for $j = 1, \cdots$,m **do**
  $c_{j+1} \leftarrow c_j + w^j$
end for
draw initial sample u_0 from uniform density over $[0, \frac{1}{m}]$
$i \leftarrow 2$
for $j = 1, \cdots, m$ **do**
  $u_j \leftarrow u_j + \frac{j-1}{m}$
  while $u_j > c_i$ **do**
    $i \leftarrow i + 1$
  end while
  $\bar{w}^j \leftarrow \frac{1}{m}$
  $\bar{x}^j \leftarrow x^{i-1}$
end for
$\bar{S} \leftarrow \{\bar{w}^j, \bar{x}^j\}_{j=1}^m$
return $\bar{\mathcal{S}}$

is especially a concern in cases when the system is subject to a small process noise [112]. The resampling algorithm is described in Algorithm 3.

The prediction stage has an $O(lm)$ computational time complexity, where m is the number of samples, and l is the computational time complexity of drawing a random sample from $q_{\mathbf{x}_k|\mathbf{x}_{k-1},u_{k-1},y_k}$. The update stage has a computational time complexity of $O(m)$, which includes the $O(m)$ computational time complexity of the resampling procedure. The algorithmic complexity of the particle filter up to the time horizon K is therefore $O(Kml)$. This computational time complexity includes all stages including the prediction, update, and resampling. When samples are generated from $p_{\mathbf{x}_k|\mathbf{x}_{k-1},u_{k-1}}$, the computational time complexity of the sampling procedure is $O(|\mathcal{X}|)$. This occurs because the sampling procedure requires that a sample be generated from an uniform probability function. The sample is then indexed into the cumulative distribution function generated from the transition probability function, which has $|\mathcal{X}|$ possibilities. This process is similar to the resampling procedure outlined in Algorithm 3. Thus, when the importance sampling function is chosen as the transition probability function, particle

filtering has a computational time complexity of $O(Km|\mathcal{X}|)$. While particle filtering was originally derived for continuous space systems, it can be applied as an approximation to discrete systems to reduce the computational burden of finding the exact solution. The exact solution for the discrete case is $O(K|\mathcal{X}|^2)$. In many robotics applications there can be tens of millions of states. Evaluation of such a system is not practical with today's computational means and the $O(Kml)$ time complexity of the particle filter is preferable to the exact solution.

4 HYPERFILTERING

Hyperfiltering is a method, for systems modeled by POMDPs, to propagate the estimate of the belief, b_k at stage k, and its uncertainty forward into future stages for unseen observations and unactualized control inputs. By choosing the probability function over the beliefs, hyperfiltering is able to sequentially evaluate the estimate of the system and its uncertainty forward from one stage to the next. Moreover, by adopting the complete representation of the uncertainty, instead of just some statistics of the belief, a more accurate representation of the evolution of the system is obtained.

In this chapter, both the motivation (Section 4.1) and formulation (Section 4.2) of the hyperfilter are explored. Additionally, the hyper-particle filtering method that approximates the hyperfilter is introduced in Section 4.3.

4.1 Hyperfilter Motivation

When dealing with stochastic processes, a method is needed to understand how a process evolves over time. Closed form solutions representing the behavior of a system as it evolves are elusive and, for most systems, are assumed to not exist. Simulations are therefore performed to predict the evolution of such processes. There are two objectives desired for predicting the evolution of stochastic systems. The first is maintaining a representation of the system and its uncertainty at a given stage that can be used to estimate the system and its uncertainty at a future stage. Such a representation would enable the sequential evaluation of the system forward into future stages. The second is a representation that enables the system's performance to be measured, such as the

likelihood of a robot achieving a certain objective, the quality of achieving an objective, or merely to understand the effects the noise on a system.

For stochastic processes, the expected value as well as higher-order moments are typically evaluated as a method to understand the evolution of a system. For instance, if the system is executing a policy within the class of feedback policies that depend on the information state (or any equivalent sufficient statistic such as the belief), the representation of the expectation of evolution of x_k over the set of all possible observations becomes

$$E_{\mathbf{I}_k}[p(x_k|\mathbf{I}_k)] = \sum_{I_k \in \mathcal{I}_k} p(x_k|I_k)p(I_k) \tag{4.1}$$

$$= \sum_{y_1 \in \mathcal{Y}} \sum_{y_2 \in \mathcal{Y}} \cdots \sum_{y_k \in \mathcal{Y}} p(x_k|I_{k-1}, \pi(I_{k-1}), y_k)p(I_{k-1}, \pi(I_{k-1}), y_k) \tag{4.2}$$

$$= \sum_{y_1,\ldots,y_k \in \mathcal{Y}} p(x_k|I_{k-1}, \pi(I_{k-1}), y_k)p(y_k|I_{k-1}, \pi(I_{k-1}))p(I_{k-1}, \pi(I_{k-1})) \tag{4.3}$$

$$\vdots$$

$$= \sum_{y_1,\ldots,y_k \in \mathcal{Y}} p(x_k|\pi(I_1), \pi(I_2), \ldots, \ \pi(I_{k-1}), y_1, \ldots, y_k) \cdot$$
$$p(y_k|I_{k-1}, \pi(I_{k-1})) \cdots p(y_1|I_1, \pi(I_1))p(y_1|x_1)p(x_1), \tag{4.4}$$

where (4.2) is obtained from (4.1) by pulling u_{k-1} and y_k out of I_k and substituting $u_{k-1} = \pi(I_{k-1})$. Because $p(I_k, u_{k-1}, y_k) = p(y_k|I_{k-1}, u_{k-1})p(I_{k-1}, u_{k-1})$, (4.3) is obtained from (4.2). The result in (4.4) is obtained by repeating the process performed in (4.2) and (4.3) from stage $k-1$ to the initial stage.

Because the policy maps from the random information state at each stage to a control action, the effect of each observation must be considered. This clearly demonstrates that for controlled systems subject to an information-feedback policy, the effect of the observation on the system's evolution cannot just be eliminated, and hence, the

posterior over the observation space at each stage plays a critical role in the evaluation of the evolution of the system.

Propagating only the expected belief from stage to stage has little value as the uncertainty in the system is not evaluated. Knowledge of the uncertainty of the system from stage to stage is needed to fully understand the evolution of the system when the system is subject to noise. Higher-order moments of the probability over the set of possible observations in the observation space can be determined to estimate the uncertainty of the system. Just as for finding the expected value, the posterior over the set of possible observations is essential in evaluating higher-order moments. Other moments or measures may also be considered. However, in any case where the policy is a function of the information state (or a sufficient statistic), the posterior over the observation space is required to estimate the performance of the system forward to future stages.

While some filtering methods exist that propagate an estimate of the moments like [113], they are for a specific class of problems. The majority of methods for filtering of nonlinear non-Gaussian systems approximate the probability function and evaluate the approximated probability function between stages. Even other methods that estimate moments are just approximating moments used to fit to an assumed probability function as [80, 82]. In the general case, when no special structure is known, it is not obvious how to propagate a finite set of moments from one stage to the next. It would then seem reasonable to pursue a method to approximate the probability function over the belief rather than just a finite set of moments.

There are two possibilities to predict the behavior of a system, either by a backward-based method or by a forward-based method. In the former, a representation of behavior of a system at stage k is attained via a representation of the behavior of the system at stage $k+1$, whereas in the latter a representation of the behavior of a system at stage $k+1$ is determined from the representation of the behavior of the system at stage k.

The weakness of backward-based approaches is that the representation of the behavior of the system must be known for all possible representations at stage $k+1$ in order for the behavior at the preceding stage k to be determined because the preceding stage k indexes into the set of possible representations at stage $k+1$. To overcome this handicap, a set of feasible representations can be sampled to reduce the set of possibilities considered, but the generation of this set is typically done through online forward-based methods (e.g., [70–72, 101]) or offline learning methods (e.g., [73]), which use forward-based simulation techniques to determine the feasible set. Even so, these techniques either are iterative in nature, or fail to propagate the estimate of the behavior and its uncertainty forward and instead just sample a set of feasible representations of the behavior. Thus forward-based methods are preferable over backward-based methods, especially considering many backward-based approximation methods use forward-based methods to aid in the approximation.

4.2 Hyperfiltering Formulation

To generate a forward-based sequential method, a construct that encapsulates the effect of the unobserved random observations must first be found. Interestingly, expanding the focus and attention of the search onto what happens in the space of probability functions over the belief space, the evolution of the probability function over the beliefs can be observed to be the precise construct needed, enabling the examination of the evolution of the stochastic system forward into future stages for unknown observations. This is fortuitous as the goal is to evaluate the behavior of the system based not only on the state but also on the belief. Hyperfiltering is then the application of filtering on the belief space. The conceptual difference between filtering and hyperfiltering is that filtering encapsulates all past observations and controls into the estimate of the current belief, whereas hyperfiltering takes into account all possible values of future

observations and control actions when making an estimate of the probability function over the belief.

When filtering, in the traditional sense, an observation is made so the belief transition function Definition 3.1 is deterministic:

$$b_k = B(b_{k-1}, u_{k-1}, y_k).$$

However, when predicting future behavior, the observations are unknown and stochastic in nature. The future belief therefore becomes a random variable (refer to Definition 4.1 below) defined by the stochastic process

$$\mathbf{b}_{k+1} = B(\mathbf{b}_k, \pi(\mathbf{b}_k), \mathbf{y}_{k+1},). \tag{4.5}$$

The evolution from one stage to the next via the stochastic process (4.5) generates a random variable and, thus, a representation of the probability function over the belief is needed to proceed.

4.2.1 The hyperbelief

To incorporate the future unseen observations, hyperfiltering maintains a probability function over the belief space. Just as the belief gave rise to the ability to filter over a state space, the probability function over the belief, known as the hyperbelief, enables the forward sequential prediction of the evolution of POMDPs.

Definition 4.1. *For a POMDP with a discrete state space and applied control policy π, the hyperbelief β_k at stage k is a functional, such that $\beta_k : \mathcal{P}_b \to \mathbb{R}_+$. The hyperbelief is a probability function over the belief space at each stage. The initial hyperbelief β_1 at*

stage $k = 1$ is given; for $k > 1$, the hyperbelief is defined as

$$\beta_k \triangleq p_{b_k|\beta_1,\pi}.$$

Note that $\mathbb{R}_+$ denotes the set of non-negative real numbers. By representing the probability function over the belief at each stage given the initial hyperbelief and control policy π, the hyperbelief is a representation of the predicted behavior of a system at future stages. Each β_k is contained in the *hyperbelief space* $\mathcal{P}_\beta$. The hyperbelief space $\mathcal{P}_\beta$ is defined as the set of all probability measures over or $\mathbb{B}(\mathcal{P}_b)$, the Borel σ-algebra defined over the belief space $\mathcal{P}_b$.

For discrete state space POMDP systems, each of the state space, belief space, and hyperbelief space are well defined. The belief space is represented as an n-1 dimensional simplex. The Borel σ-algebra $\mathbb{B}(\Delta^{n-1})$ exists, and thus, the hyperbelief space is well defined.

An illustration of the relationship between the state space, the belief space, and the hyperbelief space is depicted in Figure 4.1. In the depicted example, the state space $\mathcal{X}$ at Figure 4.1(a) contains 10 states. A probability function over the state at Figure 4.1(b) maps into a point b at Figure 4.1(c) in the belief space $\mathcal{P}_b$, which is an $|\mathcal{X}| - 1$ dimensional simplex. In a similar manner at Figure 4.1(d), a probability function over the belief space $\mathcal{P}_b$, maps to a point β in the hyperbelief space $\mathcal{P}_\beta$ at Figure 4.1(e).

4.2.2 Evolution of the hyperbelief: The hyperfilter

By focusing on the evolution of the hyperbelief from future stage to future stage, it is possible to propagate the complete estimate of the belief and its uncertainty, in a similar manner to how the Bayesian filter propagates the belief, and not merely some statistics of the belief, from past stages to the current stage. Moreover, it is possible to derive a sequential method, whereby starting from the initial stage and progressing

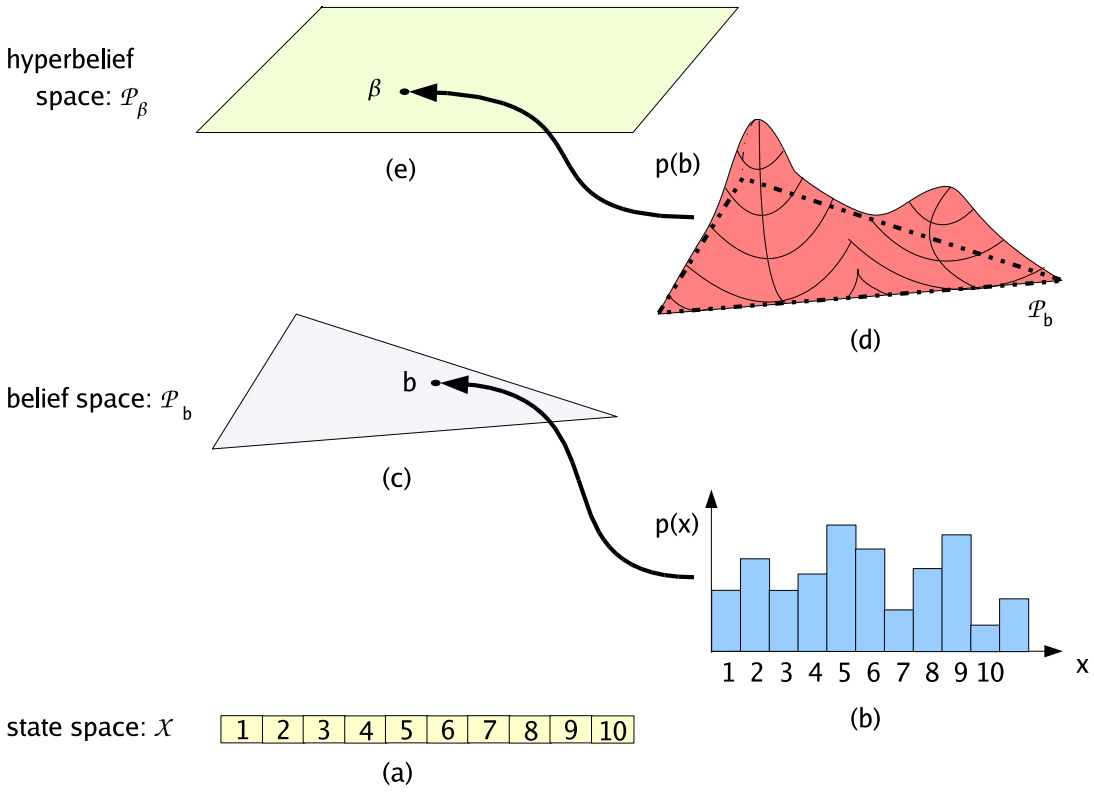


Figure 4.1: The hyperbelief space (e) is the set of probability functions over the belief space (c), such as (d), whereas the belief space (c) is the set of probability functions over the state space (a), such as (b)

forward, the hyperbelief β_{k+1} at future stage $k+1$ is calculated from the hyperbelief β_k at future stage k.

Definition 4.2. *The belief transition probability function $p(b_{k+1}|b_k, u_k)$, represents the probability of the outcome b_{k+1} of the stochastic process $B(\mathbf{b}_k, u_k, \mathbf{y}_{k+1})$ given b_k and the applied control input u_k.*

Since both π and b_k are known, the probability function over the the set of possible observations can be inferred. This induces a probability function on the set of actions, which is used to determine the probability of b_{k+1} occurring. The class of control policies explored are restricted to information-feedback polices; thus, the observation determines which control action is applied. The state space and, therefore, the belief space can be augmented to include a stage number, making it possible to define time varying feedback policies.

The hyperbelief β_{k+1} can be marginalized on the previous hyperbelief to represent the β_{k+1} as the integral of the belief transition probability function and the previous hyperbelief β_k at stage k. In turn, β_k can be represented as the integral of the belief transition probability function and the previous hyperbelief β_{k-1} at stage $k-1$. Thus, a sequential formulation of the hyperbelief can be formulated.

Defining Π to be the set of all information feedback policies and $M(\mathcal{P}_b)$ as the set of all $\mathbb{B}(\mathcal{P}_b)$-measurable functions defined over $\mathcal{P}_b$, it is possible to establish a sequential formulation of the hyperbelief.

Theorem 4.1. *For a system modeled as a POMDP with a discrete state space and with a given control policy $\pi \in \Pi$, the hyperbelief $\beta_k \in \mathcal{P}_\beta$ at stage k given the initial hyperbelief $\beta_1 \in \mathcal{P}_\beta$, can be evaluated via the sequential application of the belief transition probability function from stage k to the initial stage. This holds if the belief transition function is defined such that $p_{b_{k+1}|b_k,u_k}(\cdot|\ b_k, u_k) \in \mathcal{P}_\beta$ for all $b_k \in \mathcal{P}_b$, $u_k \in \mathcal{U}$ and $p(b_{k+1}|\cdot, u_k) \in \mathcal{M}(\mathcal{P}_b)$ for all $b_{k+1} \in \mathbb{B}(\mathcal{P}_b)$, $u_k \in \mathcal{U}$.*

Proof. The proof follows by induction on the application of the belief transition probability function. First note that belief transition function, as a function of a random belief and a random observation, is Markovian; the future probability of a belief depends only on the previous belief, and the policy, which depends on the previous belief. At the second stage, $k = 2$, the hyperbelief can be formulated by marginalizing the hyperbelief β_2 on b_1 and substituting $u_1 = \pi(b_1)$ to obtain

$$\begin{aligned} \beta_2(b_2) = p(b_2|\beta_1, \pi) &= \int_{b_1 \in \mathcal{P}_b} p(b_2|b_1, \beta_1, \pi(b_1)) p(b_1|\beta_1) db_1 \\ &= \int_{b_1 \in \mathcal{P}_b} p(b_2|b_1, \pi(b_1)) \beta_1(b_1) db_1. \end{aligned} \tag{4.6}$$

Because the state space is finite, the belief space is represented as a finite dimensional simplex. The integration therefore is performed over the simplex. In the second equation, $p(b_2|b_1, \beta_1, \pi(b_1))$ is conditionally independent of β_1 when conditioned on b_1. Because $p(b_1|\beta_1)$ is the probability of b_1 conditioned on the hyperbelief β_1, by definition it is equivalent to $\beta_1(b_1)$. As β_1 is the initial hyperbelief, it is assumed to be given. The result is $p(b_2|b_1, \pi(b_1))$, which is just the belief transition probability function. The assumption that $\beta_1 \in \mathcal{P}_\beta$ implies that β_1 is an integrable function. Moreover, by the assumption that $p(b_{k+1}|\cdot, u_k) \in \mathcal{M}(\mathcal{P}_b)$ for all $b_{k+1} \in \mathbb{B}(\mathcal{P}_b)$, $u_k \in \mathcal{U}$, implies (4.6) is integrable as the product of two integrable functions is also integrable. Moreover, because $p_{b_{k+1}|b_k,u_k}(\cdot|\ b_k, u_k) \in \mathcal{P}_\beta$ for all $b_k \in \mathcal{P}_b$, $u_k \in \mathcal{U}$ is integrable and satisfies the properties to be a probability measure, then, by definition of $\mathcal{P}_\beta$, the hyperbelief β_2 belongs to $\mathcal{P}_\beta$.

Assuming that $\beta_{k-1} \in \mathcal{P}_\beta$ and that β_{k-1} can be evaluated by integrating over the belief transition probability function the hyperbelief at stage $k-2$, the hyperbelief β_k

at stage k is expressed as

$$\begin{aligned}\beta_k(b_k) = p(b_k|\beta_1,\pi) &= \int_{b_{k-1}\in\mathcal{P}_b} p(b_k|b_{k-1},\beta_1,\pi(b_{k-1}))p(b_{k-1}|\beta_1,\pi)db_{k-1} \\ &= \int_{b_{k-1}\in\mathcal{P}_b} p(b_k|b_{k-1},\pi(b_{k-1}))\beta_{k-1}(b_{k-1})db_{k-1}. \qquad (4.7)\end{aligned}$$

The first equation follows from the marginalization of the probability of $p(b_k|\beta_1,\pi)$ on b_{k-1}. The second equation follows from the belief transition probability function at stage k being conditionally independent of the initial hyperbelief β_1 given the belief b_{k-1} at stage $k-1$ and by substituting $u_{k-1} = \pi(b_{k-1})$ into the belief transition probability function. Again, because of the assumed form of the belief transition probability function and that $\beta_{k-1} \in \mathcal{P}_\beta$, (4.7) is integrable and $\beta_k \in \mathcal{P}_\beta$. □

An analogous proof can be established for control policies that are time varying or for open-loop policies. Notice the analog between the belief transition probability function and that of completely observable MDP type processes. By abstracting into the belief space, the analysis reduces to evaluating a fully observable equivalent of the evolution of the system, where the observation is treated as a noise term in the belief space, similar to the process noise in an MDP system. From this insight, the hyperbelief transition function naturally follows.

Definition 4.3. *The function that transfers a hyperbelief $\beta_k \in \mathcal{P}_\beta$ into the hyperbelief $\beta_{k+1} \in \mathcal{P}_\beta$ given a feedback policy that depends on the belief is denoted as the hyperbelief transition function Υ, such that $\Upsilon : \mathcal{P}_\beta \times \Pi \to \mathcal{P}_\beta$, where Π is the set of all feedback policies. The hyperbelief transition function is represented as*

$$\beta_{k+1} = \Upsilon(\beta_k, \pi),$$

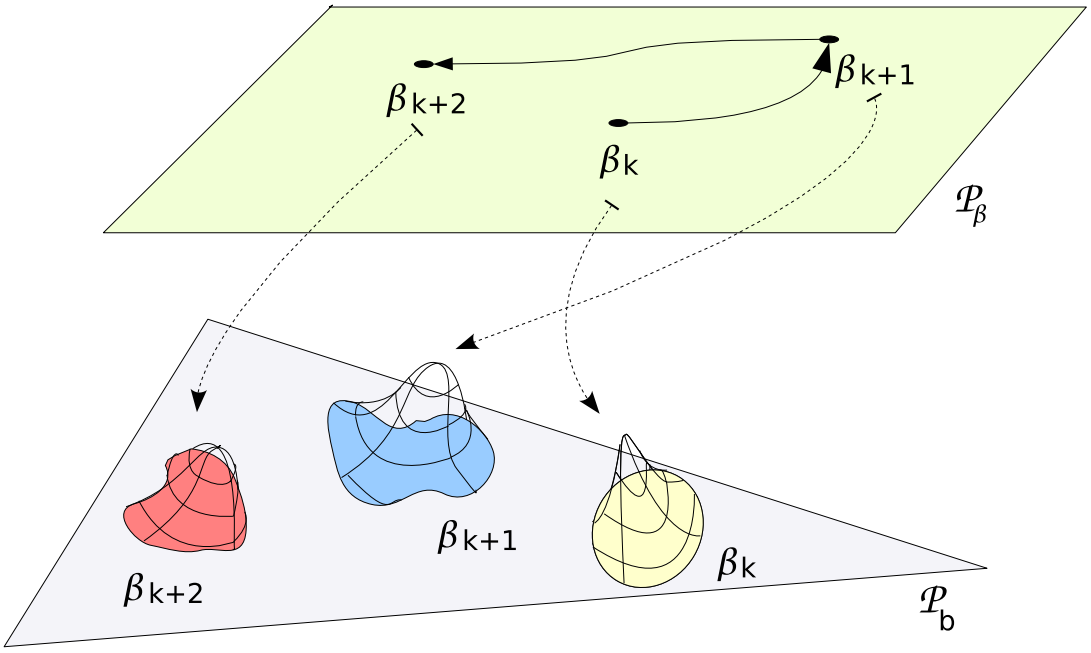


Figure 4.2: The evolution of the hyperbelief

where, for each $b_{k+1} \in \mathcal{P}_b$,

$$\Upsilon(\beta_k, \pi)(b_{k+1}) \triangleq \int_{b_k \in \mathcal{P}_b} p\left(b_{k+1} | b_k, \pi\left(b_k\right)\right) \beta_k(b_k) db_k.$$

Because the output of the hyperbelief transition function is defined over the belief space $\mathcal{P}_b$, the notation $\Upsilon(\cdot)(b_{k+1})$ is adopted to represent the resulting function evaluated at a specific belief $b_{k+1} \in \mathcal{P}_b$.

The hyperbelief transition function can be nested as a set of compositions up to any given stage. In this way, it is possible to preserve the result of one stage as the input for the next stage. Hence, the hyperbelief encapsulates all the information needed to predict the evolution of a partially observed system into future stages. The concept of the hyperfilter is illustrated in Figure 4.2. By applying the hyperbelief transition function to the hyperbelief β_k at stage k using the policy π, the hyperbelief β_{k+1} is

generated. Likewise, by sequentially applying the hyperbelief transition function to the resultant β_{k+1} the hyperbelief β_{k+2} is generated. In each case the evolution of the hyperbelief in $\mathcal{P}_\beta$ corresponds to a probability function over $\mathcal{P}_b$ as is illustrated in the figure.

4.2.3 Generation of the belief transition probability function

To generate the belief transition probability function, several properties will need to be discussed to provide insight into the hyperfiltering method. As discussed above, the evolution of the system into future stages produces a hyperbelief via the stochastic equivalent of the belief transition function, whereby the observation is a random variable instead of being an observed outcome.

As discussed in Section 3.1.1, the belief transition function is a composition of two steps: the prediction step and the update step, both of which are deterministic processes that transition a belief into another belief in the belief space. From Definition 3.1, with the evaluation of η_k substituted into the equation, the belief transition function for a belief b_k, control action u_k, and observation y_{k+1} is defined as

$$B(b_k, u_k, y_{k+1})(x_{k+1}) = \frac{p(y_{k+1}|x_{k+1}) \sum_{x_k \in \mathcal{X}} p(x_{k+1}|x_k, u_k) b_k(x_k)}{\sum_{x_{k+1} \in \mathcal{X}} p(y_{k+1}|x_{k+1}) \sum_{x_k \in \mathcal{X}} p(x_{k+1}|x_k, u_k) b_k(x_k)}, \tag{4.8}$$

where the notation $B(\cdot)(x_{k+1})$ is adopted to represent the resulting function evaluated at a specific state x_{k+1}.

The prediction step updates the probability function based on an applied control action; more precisely, the prediction step transitions a belief b_k at stage k to a belief $\hat{b}_{k+1}$ at stage $k+1$ for some control action u_k, and can be represented by $\hat{b}_{k+1} = \hat{B}(b_k, u_k)$, where for all $\hat{x}_{k+1}$ in $\mathcal{X}$,

$$\hat{B}(b_k, u_k)(\hat{x}_{k+1}) = \sum_{x_k \in \mathcal{X}} p(\hat{x}_{k+1}|x_k, u_k) b_k(x_k), \tag{4.9}$$

and thus, $\hat{b}_{k+1} = p_{x_{k+1}|I_k,u_k}$.

By substituting $\hat{b}_{k+1}$ into (4.8), the update can be formulated, which reweights the probability function based on an observation. At stage $k+1$, the update step $\bar{B}(\cdot)$ transitions the belief $\hat{b}_{k+1}$ to belief b_{k+1} at stage $k+1$ for an observation y_{k+1}. The update step is give as

$$\bar{B}(\hat{b}_{k+1}, y_{k+1})(x_{k+1}) = \frac{p(y_{k+1}|x_{k+1})\hat{b}_{k+1}(x_{k+1})}{\sum_{x_{k+1}\in\mathcal{X}} p(y_{k+1}|x_{k+1})\hat{b}_{k+1}(x_{k+1})}, \tag{4.10}$$

for all x_{k+1} in $\mathcal{X}$.

The composition of both (4.9) and (4.10) becomes the belief transition function:

$$B(b_k, u_k, y_{k+1}) = \bar{B}(\hat{B}(b_k, u_k), y_{k+1}).$$

In this way, the belief transition function can be split into two steps.

When the belief of the previous stage is random and the observation is random, the evolution of the system from one stage to the next still proceeds in two steps. The first step is the predicted transition of a random belief under a given policy. This is related to the prediction stage of traditional filtering methods. When conditioned on a specific belief, $\hat{B}$ condenses to a single point indicating the single, unique outcome. The probability of $\hat{b}_{k+1}$, therefore, is given as

$$p(\hat{b}_{k+1}|b_k, u_k) = \delta(\hat{b}_{k+1} - \hat{B}(b_k, u_k)). \tag{4.11}$$

However, if a random belief at stage k is given, the resulting hyperbelief is calculated for every possible belief in the previous hyperbelief, hence for some $\hat{b}_{k+1}$,

$$p(\hat{b}_{k+1}|\beta_k, \pi) = \int_{b_k\in\mathcal{P}_b} p(\hat{b}_{k+1}|b_k, \pi(b_k))\beta_k(b_k)db_k. \tag{4.12}$$

Like the prediction step $\hat{B}$, when the update $\bar{B}$ is conditioned on a specific observation the probability function over the belief space condenses to a single point in the belief space indicating the single, unique outcome:

$$p(b_{k+1} \mid \hat{b}_{k+1}, y_{k+1}) = \delta(b_{k+1} - \bar{B}(\hat{b}_{k+1}, y_{k+1})). \tag{4.13}$$

However, the observation is unknown when predicting the future, so the observation $\mathbf{y}_{k+1}$ acts like a noise term in the update $\bar{B}(\hat{b}_{k+1}, \mathbf{y}_{k+1})$. When taking the random observation into account, the resulting probability for some belief b_{k+1} given some predicted belief $\hat{b}_{k+1}$ is given as

$$\begin{aligned} p(b_{k+1}|\hat{b}_{k+1}) &= \sum_{y_{k+1}\in\mathcal{Y}} p(b_{k+1}|\hat{b}_{k+1}, y_{k+1}) p(y_{k+1}|\hat{b}_{k+1}) & (4.14) \\ &= \sum_{y_{k+1}\in\mathcal{Y}} p(b_{k+1}|\hat{b}_{k+1}, y_{k+1}) \sum_{x_{k+1}\in\mathcal{X}} p(y_{k+1}|x_{k+1}) \hat{b}_{k+1}(x_{k+1}), & (4.15) \end{aligned}$$

where in (4.14) y_{k+1} is introduced as a marginalizing term. The probability function $p(y_k|\hat{b}_{k+1})$ represents the posterior over the observation given the probability function $\hat{b}_{k+1}$ defined over the state space, which is evaluated as $p(y_k|\hat{b}_{k+1}) = \sum_{x_{k+1}} p(y_{k+1}|x_{k+1})\hat{b}(x_{k+1})$. This property is used to obtain (4.15) from (4.14).

If the composition of both (4.11) and (4.13) is taken, it is possible to represent the belief transition probability function as

$$\begin{aligned} p(b_{k+1}|b_k, u_k) &= \int_{\hat{b}_{k+1}\in\mathcal{P}_b} p(b_{k+1}|\hat{b}_{k+1}, u_k) p(\hat{b}_{k+1}|b_k, u_k) d\hat{b}_{k+1} & (4.16) \\ &= \int_{\hat{b}_{k+1}\in\mathcal{P}_b} p(b_{k+1}|\hat{b}_{k+1}) p(\hat{b}_{k+1}|b_k, u_k) d\hat{b}_{k+1} & (4.17) \\ &= \int_{\hat{b}_{k+1}\in\mathcal{P}_b} p(b_{k+1}|\hat{b}_{k+1}) \delta(\hat{b}_{k+1} - \hat{B}(b_k, u_k)) d\hat{b}_{k+1} & (4.18) \\ &= p(b_{k+1}|\hat{B}(b_k, u_k)). & (4.19) \end{aligned}$$

By marginalizing the belief transition probability function on $\hat{b}_{k+1}$, (4.16) is obtained. The fact that the probability of b_{k+1} is conditionally independent of u_k given $\hat{b}_{k+1}$ is applied to derive (4.17) from (4.16). Next (4.11) is substituted into the equation to derive (4.18). Finally, the equation reduces to (4.19) because $\hat{b}_{k+1}$ is unique with probability one when conditioned on b_k. The integration reduces to the single point $\hat{b}_{k+1} \in \mathcal{P}_b$, which is obtained via $\hat{B}(b_k, u_k)$.

While (4.19) provides insight on the evolution of the system at the belief level, it does not represent the complete evolution from the state space perspective. By expanding (4.19) to be in terms of the transition probability function and the observation probability function, the belief transition probability function becomes

$$\begin{aligned}
p(b_{k+1}|b_k, u_k) &= p(b_{k+1}|\hat{B}(b_k, u_k)) \\
&= \sum_{y_{k+1} \in \mathcal{Y}} p(b_{k+1}|\hat{b}_{k+1}, y_{k+1}) p(y_{k+1}|\hat{B}(b_k, u_k)) \\
&= \sum_{y_{k+1} \in \mathcal{Y}} p(b_{k+1}|\hat{b}_{k+1}, y_{k+1}) \sum_{\hat{x}_{k+1} \in \mathcal{X}} p(y_{k+1}|\hat{x}_{k+1}) \hat{B}(b_k, u_k)(\hat{x}_{k+1}) \\
&= \sum_{y_{k+1} \in \mathcal{Y}} p(b_{k+1}|\hat{b}_{k+1}, y_{k+1}) \sum_{\hat{x}_{k+1} \in \mathcal{X}} p(y_{k+1}|\hat{x}_{k+1}) \sum_{x_k \in \mathcal{X}} p(\hat{x}_{k+1}|x_k, u_k) b_k(x_k) \\
&= \sum_{y_{k+1} \in \mathcal{Y}_b(b_{k+1}, b_k, u_k)} 1 \sum_{\hat{x}_{k+1} \in \mathcal{X}} p(y_{k+1}|\hat{x}_{k+1}) \sum_{x_k \in \mathcal{X}} p(\hat{x}_{k+1}|x_k, u_k) b_k(x_k) \quad (4.20)
\end{aligned}$$

where

$$\mathcal{Y}_b(b_{k+1}, b_k, u_k) = \{y \ : \ b_{k+1} = B(b_k, u_k, y), \forall y \in \mathcal{Y}\}.$$

The second and third equations just perform the steps of (4.14) and (4.15), respectively, where $\hat{b}_{k+1}$ is substituted with $\hat{B}(b_k, u_k)$. The third equation follows from the application of (4.9). In (4.20), because (4.13) is a delta function, all observations outside of $\mathcal{Y}_b(b_{k+1}, b_k, u_k)$ are evaluated with a weight of zero and all the observations in the set

are assigned a weight of one. The summation over $y_{k+1} \in \mathcal{Y}$ therefore can be reduced to the summation over $y_{k+1} \in \mathcal{Y}_b(b_{k+1}, b_k, u_k)$.

As can be seen from (4.20), the hyperfiltering equations are likely to be highly nonlinear. Moreover, the set $\mathcal{Y}_b$ may not be easily computed. However, for specific systems, insight into the evolution of the system may provide alternative formulations of the hyperfiltering equations that eliminate these issues. As an example, if there are a finite number of beliefs with nonzero probability in the in initial hyperbelief, each future hyperbelief comprises only a finite number of beliefs with nonzero probability because there are only a finite number of possible observations and states. Thus, even though the system evolves in a continuous space, only a finite set of possibilities may occur at each stage.

4.3 Hyper-Particle Filtering: A Hyperfiltering Approximation

While filtering is a difficult problem, the issues with hyperfiltering grow exponentially (quite literally in the case of discrete systems); the number of possible beliefs can grow exponentially with the time horizon. Another issue is the high nonlinearity of the belief transition probability function as shown in Section 4.2.3. Because of these issues, an approximation method is needed. Based on particle filtering, hyper-particle filtering is introduced to approximate hyperfiltering in the following section.

4.3.1 Hyper-particle filter formulation

The hyperfiltering technique determines the hyperbelief β_{k+1} from the hyperbelief β_k. However, the hyperfiltering algorithm may be difficult or impossible to implement precisely, so an approximation may be required. To achieve this approximation, the hyper-particle filtering method is developed based on the particle filtering technique. Whereas

traditional particle filtering represents the evolution over just a single known observation at each stage, hyper-particle filtering represents the evolution of the system over all possible unknown observations at each future stage. Based on Monte Carlo integration, particle filtering approximates the probability function of a random vector by a set of samples and associated weights. The weights are normalized values that represent the approximated probability of the system being in the state represented by each one of the samples. Each particle is sampled according to an importance sampling function. Often the sampling strategy is random, typically either the true transition probability function or a simplified approximation of the transition probability function. Under very general conditions it has been shown that the particle filter converges to the true probability function for both continuous and discrete random variables [56, 67]. One aspect that sets hyper-particle filtering apart from sample path simulation, besides the sequential nature, is that a set of beliefs is sampled from the hyperbelief at each stage, not just a single observation from the single belief as done in sample path simulation.

Hyper-particle filtering takes as input a set of hyper-particles (which approximate the hyperbelief) and a control policy, and outputs a new set of hyper-particles via the hyperbelief transition probability function (as defined at Definition 4.3). The hyper-particle filter is a two-tiered approach. At the lower level, a traditional particle filter is used to approximate one possible belief over the state by a set of particles. At the upper level, the hyperbelief is approximated by a set of hyper-particles. Each hyper-particle has both a sample and a weight associated to it. The sample is just an approximated belief represented by a particle set. The weight approximates the probability of the associated sample. Besides being a two-tiered approach, the hyperfilter proceeds in two steps, in a manner similar to the particle filter, with a prediction step and an update step.

To illustrate the notion of hyper-particle filtering, only basic particle filtering techniques are employed (i.e., the prototypical sequential importance resampling method

[112]). Of course, there are more compelling developments that could be incorporated into the hyper-particle filtering framework such as those referenced in Section 3.3.

At the upper level, the hyper-particle filter at stage k consists of a set of R hyper-particles. These hyper-particles are denoted as $\mathcal{Z}_k = \{z_k^i\}_{i=1}^R$. Each z_k^i is a pair where $z_k^i = (\alpha_k^i, b_k^i)$, with α_k^i a nonnegative scalar weight and $b_k^i \in \mathcal{P}_b$. Each b_k^i represents a point in belief space, and α_k^i represents a probability mass for that point. The set $\mathcal{Z}_k$ approximates the hyperbelief: $\beta_k(b_k) \approx p(b_k|\mathcal{Z}_k) = \sum_{i=1}^R \alpha_k^i \delta(b_k - b_k^i)$. At the lower tier, the belief point b_k^i is associated with a traditional particle set, where each b_k^i comprises a set of pairs of scalar weights w_k^q and samples x_k^q in the state space $\mathcal{X}$, giving: $b_k^i = \{w_k^q, x_k^q\}_{q=1}^Q$, where Q is the number of particle samples. Thus, each b_k^i approximates the belief at stage k as $b_k(x_k) \approx \sum_{q=1}^Q w_k^q \delta(x_k - x_k^q)$.

The concept is to approximate the hyperbelief at stage $k+1$ from the approximated hyperbelief at stage k as represented by $\mathcal{Z}_k$ by randomly sampling set of beliefs $\{b_{k+1}^l\}$ and then adjusting their weights to generate the approximated hyperbelief $\mathcal{Z}_{k+1}$. In this way, a finite number of beliefs are generated to represent the hyperbelief at stage $k+1$ from the hyperbelief at stage k. The hyperbelief transition function is split into two steps because sampling from the belief transition probability function, $p(b_{k+1}|\ b_k, u_k)$, directly may not be practical or feasible. However, as shown in (4.19), $p(b_{k+1} \mid b_k, u_k) = p(b_{k+1}|\ \hat{B}(b_k, u_k))$, which can be used to generate samples in two steps. In the prediction step $\hat{B}$ generates the predicted sample. Then the update step is performed by generating a set of samples from an importance sampling function $q_{b_{k+1}|\hat{b}_{k+1}}$, which is conditioned on the sample generated during the prediction step.

The evolution of the hyper-particle filtering proceeds as follows. At stage k, $\mathcal{Z}_k$ is an approximation of the hyperbelief β_k. The predicted next stage hyperbelief under the control policy π is approximated by using traditional particle filtering (as described in Section 3.3) to approximate the predicted hyperbelief by sampling a set of hyper-particles from $\hat{B}$ for each hyper-particle in $\mathcal{Z}_k$. This reduces the computational burden

of the exact prediction: the exact prediction via $\hat{B}$ takes $O(|\mathcal{X}|^2)$ time when there are $|\mathcal{X}|$ states and, thus, to be useful, the particle filtering approximation of the actual transition is used to reduce the computational time complexity by generating only a fixed number Q of particle samples from stage to stage.

The outcome is a new hyper-particle $\hat{z}^j_{k+1} = \{\hat{\alpha}^j_{k+1}, \hat{b}^j_{k+1}\}$ generated for each $z^i_k \in \mathcal{Z}_k$, whereby each $\hat{b}^j_{k+1}$ approximates the deterministic outcome of the prediction step $\hat{b}^j_{k+1} \approx \hat{B}(b^i_k, u_k)$. As $\hat{B}$ is a deterministic function of the beliefs and no diffusion of probability function over the belief space occurs, all the weight of the previous hyperbelief, $z^i_k \in \mathcal{Z}_k$, that generated $\hat{b}^j_{k+1}$ should be assigned to $\hat{z}^j_k$, therefore, $\hat{\alpha}^j_{k+1} = \alpha^i_k$.

Once the predicted set of hyper-particles is generated, a set of at most T beliefs for each belief in $\hat{\mathcal{Z}}_{k+1}$ is randomly sampled to create a new set of hyper-particle samples: $\mathcal{Z}_{k+1} = \{\alpha^l_{k+1,}, b^l_{k+1}\}_{l=1}^{RT}$ at stage $k+1$. These beliefs are sampled according to an importance sampling function $q_{b_{k+1}|\hat{b}_{k+1}}(\cdot|\hat{b}^j_{k+1})$, for each j. The importance sampling function $q_{b_{k+1}|\hat{b}_{k+1}}$ is a random, or quasi-random, function that generates a sample in the belief space given $\hat{b}_{k+1}$. The importance sampling function can be biased based on a variety of desired attributes from emphasizing sampling of certain regions to forcing quasi-uniform sampling. The preferred importance sampling function is $p_{b_{k+1}|\hat{b}_{k+1}}$ in the cases where it is possible to sample this probability function directly.

For each b^l_{k+1}, the new updated weight must be calculated, where the new weight is based on the previous weight and the probability of the new belief. For hyper-particle

z_{k+1}^l the probability is approximated by the weight α_{k+1}^l, which is found by

$$\begin{aligned} p(b_{k+1}^l|\beta_k,\pi) &= \int_{b_k\in\mathcal{P}_b} p(b_{k+1}^l|b_k,\pi(b_k))p(b_k|\beta_k,\pi)db_k && (4.21)\\ &= \int_{b_k\in\mathcal{P}_b} p(b_{k+1}^l|\hat{B}(b_k,\pi(b_k)))\beta_k(b_k)db_k && (4.22)\\ &\approx \sum_{i=1}^{R} p(b_{k+1}^l|\hat{B}(b_k,\pi(b_k^i)))\alpha_k^i && (4.23)\\ &\approx \sum_{j=1}^{R} p(b_{k+1}^l|\hat{b}_{k+1})\hat{\alpha}_{k+1}^j && (4.24)\\ &\approx \eta_{k+1}\frac{p(b_{k+1}^l|\hat{b}_{k+1}^j)}{q(b_{k+1}^l|\hat{b}_{k+1}^j)}\hat{\alpha}_{k+1}^j = \alpha_{k+1}^l. && (4.25) \end{aligned}$$

Equation (4.21) follows from marginalizing $p(b_{k+1}^l|\beta_k,\pi)$ on b_k, reducing to $p(b_{k+1}^l|b_k,\pi)$. Because $p(b_k|\beta_k)=\beta_k(b_k)$ and the fact that $p(b_{k+1}|b_k,u_k)=p(b_{k+1}|\hat{B}(b_k,u_k))$, as was shown in (4.19), (4.22) is obtained from (4.21). At each stage, the hyper-particle set $\mathcal{Z}_k$ is an approximation of the hyperbelief β_k, where the approximated probability of a belief is given as

$$\beta_k(b_k)\approx p(b_k|\mathcal{Z}_k)=\sum_{i=1}^{R}\alpha_k^i\delta(b_k-b_k^i).$$

As the belief space is a closed and bounded continuous space, one would expect that the approximation of any function over this space by a set of delta functions would be a poor choice, in such a senario each of the belief points have zero measure. However, it was shown in [56, 67] that under some weak assumptions the error of the expected value of the particle filtering method taken with respect to any continuous and bounded function $c(\cdot)$ for n samples is bounded by

$$E_{\mathbf{I}_k}\left[\left(\int_{b_k\in\mathcal{P}_b} c(b_k)(\sum_{i=1}^{n}\alpha_k^i\delta(b_k-b_k^i))db_k-\int_{b_k\in\mathcal{P}_b} c(b_k)p^n(b_k|\mathbf{I}_k)db_k\right)^2\right]=\frac{m_k}{\sqrt{n}}\|c\|$$

where m_k is some constant independent of n. Moreover, when considering discrete systems, there are only a finite set of belief points at each stage, each a delta function in the belief space. Because hyper-particle filtering is a forward-based method, whereby each b_k^i is a sample in the set of feasible beliefs, the approximation given above is reasonable and represents an approximation of the finite set of possible belief points at each stage. The quality of this approach will be demonstrated in Chapter 5. Thus, (4.22) can be approximated as the summation over all the belief samples in $\mathcal{Z}_k$ in (4.23). The set $\hat{\mathcal{Z}}_{k+1}$ was generated to approximate the output of $\hat{B}$ for each sample in $\mathcal{Z}_k$ and, thus, (4.24) is obtained by substituting $\hat{\mathcal{Z}}_{k+1}$ into (4.23), where $\alpha_k^i = \hat{\alpha}_{k+1}^j$. To reduce the effect of the bias of the importance sampling function $q_{b_{k+1}|\hat{b}_{k+1}}$ (as was done for the particle filter), (4.25) is obtained from (4.24), while also satisfying the requirement that the total weight must be normalized such that $\frac{1}{\eta_{k+1}} = \sum_{l=1}^{RT} \alpha_{k+1}^l$. As was shown with regard to the particle filter, dividing by the probability of the sample being generated by the importance sampling function reduces or eliminates the impact of the bias on the expected value. Additionally, (4.25) follows from the assumption that each particle b_{k+1}^l has zero probability of occurring from any belief sample other than the belief sample $\hat{b}_{k+1}^j$ that generated it, so the summation reduces to the evaluation over a single belief. This stage is analogous to (3.11) as performed by the particle filter.

Alternatively, the samples can be drawn from an importance sampling function that considers all of $\hat{\mathcal{Z}}_{k+1}$ instead of just $\hat{b}_{k+1}^j$. Extending the class of importance sampling functions to $q_{b_{k+1}|\hat{\mathcal{Z}}_{k+1}}$ enables the probability of each $\hat{b}_{k+1}^j$ to be taken into account when sampling. This can have a significant impact on the sampling as beliefs with low probability can be sampled fewer times than those with greater probability. If $q_{b_{k+1}|\hat{\mathcal{Z}}_{k+1}}$

is chosen, the weight becomes

$$\begin{aligned} p(b_{k+1}^l|\beta_k,\pi) &\approx \sum_{j=1}^{R} p(b_{k+1}^l|\hat{b}_{k+1})\hat{\alpha}_{k+1}^j \\ &\approx \eta_{k+1}\sum_{j=1}^{R}\frac{p(b_{k+1}^l|\hat{b}_{k+1}^j)}{q(b_{k+1}^l|\hat{\mathcal{Z}}_{k+1})}\hat{\alpha}_{k+1}^j = \alpha_{k+1}^l, \end{aligned} \tag{4.26}$$

The first equation is just (4.24) restated. In (4.26) the probability of each belief sample b_{k+1}^l for each belief sample $\hat{b}_{k+1}^j$ is considered, regardless of whether $\hat{b}_{k+1}^j$ generated b_{k+1}^l. The evaluation of weight α_{k+1}^l in the equation above deviates from traditional particle filtering (refer to (4.24)). The contribution of each prior belief may have a significant impact on the posterior probability of a belief and therefore the posterior distribution. Sampling from the the posterior may then increase the quality of the result significantly by better representing the actual resulting probability function. When sampling from the posterior directly and not some other importance sampling function, there is no bias in the sampling so the weight of each sample is the same. In particular, from (4.24),

$$p(b_{k+1}^l|\hat{Z}_{k+1}) = \sum_{j=1}^{R} p(b_{k+1}^l|\hat{b}_{k+1}^j)\hat{\alpha}_{k+1}^j,$$

when $q_{b_{k+1}|\hat{Z}_{k+1}} = p_{b_{k+1}|\hat{Z}_{k+1}}$, (4.26) reduces to

$$\alpha_{k+1}^l = \eta_{k+1} = \frac{1}{RT}.$$

This can be thought of just sampling RT samples from a given probability function. However, it just so happens that the probability function being sampled is generated from a previous set of samples. The computational burden of reweighting therefore is eliminated. This, unfortunately, does not alleviate the computational cost associated with generating $p_{b_{k+1}|\hat{\mathcal{Z}}_{k+1}}$. Because of the increased computational burden in the sam-

pling and reweighting, the examples illustrated in Chapter 5 use the former approach of sampling from $q_{b_{k+1}|\hat{b}_{k+1}}$ instead.

In extremely large problems where $|\mathcal{Y}|$ is prohibitively large, so that $p_{b_{k+1}|\hat{b}_{k+1}}$ is too costly to evaluate, the probability of b_{k+1} can be approximated by a finite set of possible observations, which are generated via an importance sampling function $q_{y_{k+1}|\hat{b}_{k+1}}$. For a set of M sampled observations, $p(b^l_{k+1}|\hat{b}^j_{k+1})$ can be approximated by

$$\begin{aligned} p(b_{k+1}|\hat{b}_{k+1}) &\propto \sum_{y_{k+1}\in\mathcal{Y}} p(b_{k+1}|\hat{b}_{k+1}, y_{k+1})p(y_{k+1}|\hat{b}_{k+1}) \\ &\approx \sum_{m=1}^{M} p(b_{k+1}|\hat{b}_{k+1}, y^m_{k+1})p(y^m_{k+1}|\hat{b}_{k+1}) \\ &\approx \sum_{m=1}^{M} \frac{p(b^l_{k+1}|\hat{b}^j_{k+1}, y^m_{k+1})p(y^m_{k+1}|\hat{b}^j_{k+1})}{q(y^m_{k+1}|\hat{b}^j_{k+1})} \end{aligned} \tag{4.27}$$

In the first equation $p(b^l_{k+1}|\hat{b}^j_{k+1})$ is conditioned on y_{k+1}. The second equation is just the summation over the sampled set of observations $\{y^m_{k+1}\}_{m=1}^M$. Finally, (4.27) is obtained by reducing the effect of the bias introduced by sampling the set of observations as was done previously for the particle filter. The approximated weight for the particle b^l_{k+1} becomes

$$\begin{aligned} \alpha^l_{k+1} &= \eta_{k+1}\frac{p(b^l_{k+1}|\hat{b}^j_{k+1})}{q(b^l_{k+1}|\hat{b}^j_{k+1})}\hat{\alpha}^j_{k+1} \\ &\approx \eta_{k+1}\frac{\displaystyle\sum_{m=1}^{M}\frac{p(b^l_{k+1}|\hat{b}^j_{k+1}, y^m_{k+1})p(y^m_{k+1}|\hat{b}^j_{k+1})}{q(y^m_{k+1}|\hat{b}^j_{k+1})}}{q(b^l_{k+1}|\hat{b}^j_{k+1})}\hat{\alpha}^j_{k+1}, \end{aligned}$$

where (4.27) is substituted into the first equation to obtain the second equation. This is precisely the method used for the example in Section 5.1 where the observation space comprises over 328,000 possible observations. The algorithm describing the computa-

Algorithm 4 Hyper-particle filter

1: $\bar{\mathcal{Z}}$ = HPF($\mathcal{Z}$,n, T,π), where $\mathcal{Z} = \{\alpha^i, b^i\}_{i=1}^{R}$
2: $l \leftarrow 1$
3: **for** $j = 1, \cdots, R$ **do**
4: predict $\hat{b}^j$ using the particle filtering prediction step
5: $\hat{\alpha}^j \leftarrow \alpha^j$
6: **for** $t = 1, \cdots, T$ **do**
7: sample $\bar{b}^l$ from $q(\cdot \mid \hat{b}^j)$
8: $l \leftarrow l + 1$
9: **end for**
10: **end for**
11: **for** $l = 1, \ldots, RT$ **do**
12: $\bar{\alpha}^l \leftarrow \frac{p(\bar{b}^l \mid \hat{b}^j)}{q(\bar{b}^l \mid \hat{b}^j)} \hat{\alpha}^j$
13: **end for**
14: $\alpha_{tot} \leftarrow \sum_{l-1}^{RT} \bar{\alpha}^l$
15: normalize each $\bar{\alpha}^l$ by α_{tot}
16: $\bar{\mathcal{Z}} \leftarrow \{\bar{\alpha}^l, b^l\}_{l=1}^{RT}$
17: $\bar{\mathcal{Z}} \leftarrow$HPF_resample($\bar{\mathcal{Z}}$,$n$)
18: return $\bar{\mathcal{Z}}$

tion of one entire stage is given in Algorithm 4, where the algorithm for *HPF_resample* is given in Algorithm 5.

At this point, all that remains is to resample the hyper-particle set. The resampling essentially represents a random sampling from the posterior over the beliefs. This endows hyper-particle filtering with the ability to sample based on the likelihood of the outcome over the entire approximated hyperbelief, not just the prior of a single sample (as is done in sample path simulation). Resampling performs two other functions: it reduces the hyper-particle degeneracy (refer to [64]) and limits the growth of the number of hyper-particles. Particle degeneracy occurs when, as time evolves, particles come to represent low probability samples. This is undesirable because it causes a skewing in the representation of the true probability function. Because there are a limited number of particles, these low probability samples take the place of some potential higher probability samples. Resampling, thus, is performed to overcome this particle filtering artifact. Additionally, because multiple beliefs for the hyper-particle set at

Algorithm 5 Hyper-particle filter resample

1: $\mathcal{Z}$=HPF_resample($\bar{\mathcal{Z}}$,n), where $\bar{\mathcal{Z}} = \{\bar{\alpha}^i, \bar{b}^i\}_{i=1}^{RT}$
2: Initialize cdf c to zero
3: **for** $j = 1, \cdots, RT$ **do**
4: $\quad c_{j+1} \leftarrow c_j + \bar{\alpha}^j$
5: **end for**
6: draw initial sample u_1 from uniform density over $[0, \frac{1}{RT}]$
7: $i \leftarrow 2$
8: **for** $j = 1, \cdots, n$ **do**
9: $\quad u_j \leftarrow u_j + \frac{j-1}{n}$
10: $\quad$ **while** $u_j > c_i$ **do**
11: $\quad\quad i \leftarrow i + 1$
12: $\quad$ **end while**
13: $\quad a^j \leftarrow \frac{1}{n}$
14: $\quad b^j \leftarrow b^{i-1}$
15: **end for**
16: $\mathcal{Z} \leftarrow \{\alpha^j, b^j\}_{j=1}^{n}$
17: return $\mathcal{Z}$

stage $k+1$ may be generated for each belief in the hyper-particle set at stage k, the resampling enables a fixed number of beliefs to be selected to represent the approximated hyperbelief.

Resampling is a procedure whereby a new set of samples is generated from the current set of samples. Resampling is performed by generating the cumulative distribution function (cdf) over the hyper-particle set that is represented by $\mathcal{Z}_k$ at stage k. A starting point in the cdf is randomly generated in the range $s = \frac{1}{n}$, where n is the number of hyper-particles desired. Then for n samples, starting from s and adding $\frac{1}{n}$ for each sample, the belief with the probability nearest the probability of the sample is chosen. Finally a weight of $\frac{1}{n}$ is assigned to each new hyper-particle. This procedure is outlined in Algorithm 5. The only major difference between the hyperfiltering resampling in Algorithm 5 and particle filtering resampling (as described in Algorithm 2), is that the number of resulting hyper-particle samples is specified as an input, whereas the particle filtering resampling algorithm generates the same number output samples as in the number of samples in the input particle set.

Hyper-particle filtering is more than just a sequential implementation of sample path simulation. Hyper-particle filtering samples from the posterior over the beliefs generated from the approximated hyperbelief. Sample path simulation, on the other hand, samples a single observation from each prior belief. By sampling from the posterior over the belief, a more accurate representation of the next stage's hyperbelief can be obtained. Moreover, by representing the evolution of the system as a probability function instead of a single sample, hyper-particle filtering outperforms sample path simulation, as will be demonstrated in Chapter 5. More surprising is that, depending on the importance sampling function, the hyper-particle filter achieves a more precise estimate at a lower computational time complexity or at worst with the same computational time complexity.

4.3.2 Hyper-particle filtering algorithmic complexity

The goal of this research is to create an efficient method for approximating future hyperbeliefs in a partially observed stochastic system. The method has been outlined and described, but it still remains to establish the algorithmic complexity of this approach. The method, as described, is quite flexible. In fact, it will be shown that the method varies, depending on the choice of performance parameters, from $O(KRQ)$ to $O(QRT^K)$, where K is the time horizon, R is the number of hyper-particles, and Q is the number of particles approximating the belief (via particle filtering).

The generation of new hyper-particles is nested in two "for" loops (as seen in Algorithm 4). Within the first loop there is the particle filtering prediction stage, which generates a new belief under the control policy taking $O(QL)$ time, where Q is the number of particles in the belief representation and the particle sampling takes $O(L)$ time for each sample. The sampling of new hyper-particles is performed next, in the second loop, for which there are T samples to be generated and weights to be updated

for each of the R initial hyper-particles. Assuming that the sampling takes $O(M)$, where M can vary depending on the importance sampling function used, the entire sampling process takes $O(R(QL+MT))$ time. The reweighting occurs next for each of the RT hyper-particles which takes $O(Q)$ time. Thus, to this point, the algorithm takes $O(RT(QL+M))$ time. The outcome of this step is then sent to the resampling algorithm. The resampling proceeds by first generating the cdf over over the hyper particles. A new set of new hyper-particles is then sampled from the cdf. This entire process takes $O(V)$ time where V is the number of hyper-particles. Because the input to the resampling method takes the new set of hyper-particles, $V = RT$. When combined, the total complexity per stage, including all of these elements, becomes $O(RT(QL+M))$.

This complexity described is for a single stage. The issue is how this method performs when nested for K stages. This result depends on the desired performance. Assuming that there are n_{k-1} hyper-particles at stage $k-1$, the computational time, $g_k(\cdot)$, taken to reach stage k, can be expressed in terms of $g_{k-1}(\cdot)$, the computational time taken to reach stage $k-1$, in addition to the computational time taken at stage k, or

$$g_k(L,R,T,Q,M) = c_1 n_{k-1} T(QL+M) + c_2 g_{k-1}(L,R,T,Q,M),$$

where c_1 and c_2 are just some constant terms. Assuming the algorithm starts with R hyper-particles, then there $\zeta_k = \prod_{i=1}^{k-1} n_i$ hyper-particles at stage k. Expanding $g_k(\cdot)$ from the first stage to the k'th stage, the total cost becomes

$$g_K(L,R,T,Q,M) = \sum_{k=1}^{K} \zeta_k T[c_3 T(QL+M)]. \tag{4.28}$$

If n_k is defined as always being a constant number regardless of the input, (4.28) reduces to

$$g_K^{const}(L,R,T,Q,M) = \sum_{k=1}^{K} c_3 RTQM = c_3 KRT(QL+M) = O(KR(QL+M)),$$

where c_3 is a constant. However, if n_k is allowed to grow unconstrained, or by T times the previous stage, then at stage k, $\zeta_k = RT^{k-1}$ so the growth becomes exponential:

$$g_K^{max}(L,R,T,Q,M) = \sum_{k=1}^{K} c_3 RT^{k-1} T(QL+M) = O((QL+M)(nT)^K).$$

When T stays constant and both L and M take constant time, T, L, and M can be eliminated from consideration to obtain a lower bound based on a constant number of hyper-particles of $O(KQR)$ and an upper bound of $O(QRT^K)$ when no restriction is placed on the growth of the number of hyper-particles.

When sampling from $q_{b_{k+1}|\hat{\mathcal{Z}}_{k+1}}$, the probabilities of a belief being generated from all other beliefs are evaluated. This should enhance the accuracy of the result. However, it does come with an increased computational complexity. Generating $\hat{\mathcal{Z}}_{k+1}$ takes $O(LRQ)$ time. Then a set of RT samples are drawn from $q_{b_{k+1}|\hat{\mathcal{Z}}_{k+1}}$, which takes $O(MRT)$ time, and the reweighting takes $O(LQR^2T)$ time, resulting in a worst case running time of $O(R^2TQ)$. For a given time horizon the algorithm takes $O(KLR^2Q)$–when the number of hyper-particles remains fixed at R. However, if $p_{b_{k+1}|\hat{\mathcal{Z}}_{k+1}}$ is used as the importance sampling function, the sampling takes $M = O(LR|\mathcal{Y}|Q))$ time, where $|\mathcal{Y}|$ is the number of possible observations. If the entire posterior over $p_{b_{k+1}|\hat{\mathcal{Z}}_{k+1}}$ is generated taking $O(LR|\mathcal{Y}|Q)$, sampling can be performed directly on the posterior taking $O(RT)$ time via a method similar to the resampling algorithm. The reweighting reduces to constant time as the weight of each particle is just $\frac{1}{RT}$ (as discussed above), thus, the method takes $O(LR|\mathcal{Y}|Q)$ at each stage. The total time complexity up to stage K

for a fixed R number of hyper-particles then results in a $O(KLR|\mathcal{Y}|Q)$ running time. Moreover, if the transition probability function is chosen as the importance sampling function used for the particle filtering algorithm, the sampling takes $L = O(|\mathcal{X}|)$ time. The entire process then has a $O(K|\mathcal{X}|R|\mathcal{Y}|Q)$ computational complexity.

As a comparison to sample path simulation, the exact solution takes, $O(M+|\mathcal{X}|^2)$ to generate a new sample. However, if particle filtering is performed, it takes $O(M+QL)$ time. If the system is simulated to stage k it takes for R samples, the complexity is $O(kR(M+QL))$ and simulating all stages up to a given horizon takes $O(K^2R(M+QL))$.

5 RESULTS

5.1 Prototypical Two-Dimensional Example

To demonstrate the concept of the hyperfilter and, in particular, the effectiveness of the hyper-particle filter, a representative, complex system is analyzed using the hyper-particle filter technique. This example comprises a set of 10 000 possible states, a set of eight possible actions, and a set of 328 420 possible observations. A substantial amount of uncertainty exists in both the transition probability function and the observation probability function. Such a problem is immense by the standards of the problems in the literature (e.g., [68–73, 99–104, 106]).

In this example, the continuous state space (a subset of $\mathbb{R}^2$) is approximated by a discrete 100×100 grid, whereby the system evolves in this discrete two-dimensional workspace $\mathcal{W}$, where the space $\mathcal{W}$ consists of two sets: the free space, $\mathcal{W}_{free}$, and the obstacle space, $\mathcal{W}_{obs}$. For this simple example, Figure 5.1 illustrates $\mathcal{W}$. The white portion of the grid specifies $\mathcal{W}_{free}$, which is also the state space of the system, whereas the black portion of the grid indicates $\mathcal{W}_{obs}$. It is assumed that $\mathcal{W}_{free} \cap \mathcal{W}_{obs} = \emptyset$. Moreover, the interior boundary of the free space is specified as $\partial\mathcal{W}_{free}$. Here, $\partial\mathcal{W}_{free}$ consists of the elements of the space that are adjacent to any point in $\mathcal{W}_{obs}$.

The system evolves from stage k to stage $k+1$ according to the transition probability function $p(x_{k+1}|\ x_k, u_k)$, where $x_{k+1} \in \mathcal{W}_{free}$ is the next state, $x_k \in \mathcal{W}_{free}$ is the current state, and $u_k \in \mathcal{U}$ is the control action. In particular, the control action set for the evolution comprises the quantum set of movements to any of the eight adjacent neighbors (i.e., north, northwest, west, and so forth). The transition function distributes

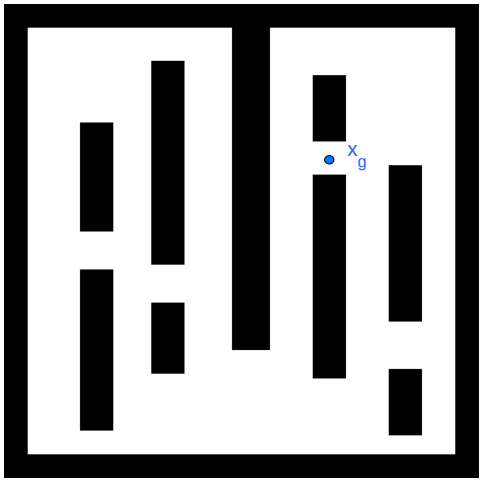


Figure 5.1: Example state space

the probability over a subset of a 5×5 grid around the current location. Taking the obstacles into account, the probability spreads out in the direction of the action and accumulates on the boundary of $\mathcal{W}_{free}$.

The robot is equipped with two types of sensors to perceive the world: contact and distance measurement. Additionally, it is assumed that the robot knows its orientation with respect to the global frame at every stage. This is assumed to be known through the implicit use of a compass. There is one contact sensor, c, indicating when the robot enters $\partial\mathcal{W}_{free}$. More precisely, if the robot is adjacent to an obstacle the contact sensor reports true with probability (w.p.) 0.8 and false w.p 0.2. Otherwise, if the robot is in $\partial\mathcal{W}_{free}$ it reports true w.p. 0.1 and false w.p. 0.9.

There are a set of four distance sensors $\mathcal{L} = \{north, south, east, west\}$ measuring the distance $d_l(x)$ in each of the cardinal directions. A distance sensor $l \in \mathcal{L}$ measures the shortest distance from the current location of the robot to the boundary of $\partial\mathcal{W}_{free}$ along some linear trajectory. However, the sensor has a finite range R_{max}. Just as the state is a discrete representation, the distance sensor has a finite set of values, each mapping a portion of the line segment to the boundary of the state space. The

distance measurement is corrupted by additive noise; the noise varies based on the distance along the senor's trajectory and hence, it is dependent on the state of the robot. The noise is modeled as a discrete approximated Gaussian. The variance of the noise is proportional to the distance to the boundary. The variance of $d_l(x)^{\frac{3}{5}}$ was chosen because it is a nonlinear function of the distance and the uncertainty of the measurement has moderate growth as the distance increases. Then the conditional probability mass function (pmf) for the distance measure y_l, given the state x and workspace $\mathcal{W}$, becomes

$$p_l(y_l|x) = \eta \frac{1}{\sqrt{2\pi d_l(x)^{\frac{3}{5}}}} e^{-\frac{(y_l-x)^2}{2d_l(x)^{3/5}}},$$

where

$$\eta = \sum_{y=0}^{D_{max}} \frac{1}{\sqrt{2\pi d_l(x)^{\frac{3}{5}}}} e^{-\frac{(y_l-x)^2}{2d_l(x)^{3/5}}}.$$

Essentially, each distance sensor is an approximation of ultrasonic sensors that are typically employed on most "can" type robots; and in this example it is assumed that the robot has a collection of four of distance sensors that encircle the robot. While typically there is some correlation with ultrasonics and IR sensors, it is assumed that each $l \in \mathcal{L}$ is independent of the other distance sensors conditioned on the state.

Together sensors c and $\mathcal{L}$ define the observation space of the robot. Because all the sensors are independent, the observation pmf of the combined readings $y_k = [y_c, y_{north}, \cdots y_{east}]$ is given as

$$p(y_k|x_k) = p(y_c|x_k)p(y_{north}|x_k)\cdots p(y_{east}|x_k).$$

Combined $p(y_k|x_k)$ and $p(x_k|x_{k-1}, u_{k-1})$ specify the probabilistic evolution of the system for each observation y_k, next state x_k, current state x_{k-1}, and control action u_{k-1}. This combinatorial representation comes with some significant drawbacks. Namely, the

number of cases that must be evaluated to generate the entire transition and observation models is more than a simple hindrance. Instead, hyper-particle filtering will be utilized to approximate the evolution of the system.

The policy employed for this simple example comprises an event-based controller. The events are defined over observation space and a subspace of $\mathbb{N}$ corresponding to a representation of a number of stages. The events represent some tangible observation set that has some physical meaning in the workspace $\mathcal{W}_{free}$. For instance, one event represents the probability of being in contact with the south wall by evaluating the probability that the robot is in contact p_c and the probability of the robot's south directed sensor p_{south} within a certain range of the predicted position of the robot.

The events are daisy chained, representing an open-loop policy at the event to event layer. However, the control policy associated with each event, which occurs at the belief to belief layer, is either open-loop or closed-loop. Each closed-loop policy is a greedy approach that chooses the next action that maximizes the probability that the next event triggers. Whether open-looped or closed-loop, the control policy relating to each event is executed until the next event is triggered. The events consist of three types: timed, accumulation and timed, and instantaneous and timed. For timed events, the policy is executed for a duration not exceeding some predetermined number of stages. Accumulation and timed events accumulate the probability of some event happening. When the probability exceeds some threshold or the given number of stages elapse, the event triggers. The probability of an event occurring by the current stage is computed as

$$p_{iter}(k) = p_{iter}(k-1) + [1 - p_{iter}(k-1)]p_{event},$$

where $p_{iter}(k)$ is the probability of the event occurring *by* stage k and p_{event} is the probability of the event of interest occurring *at* stage k. The instantaneous and timed events trigger when a certain probability threshold is exceeded at a particular stage or

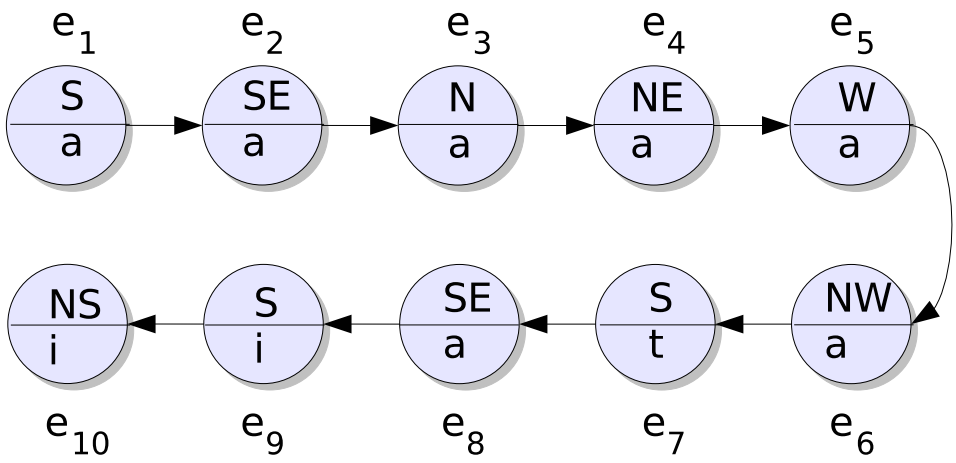
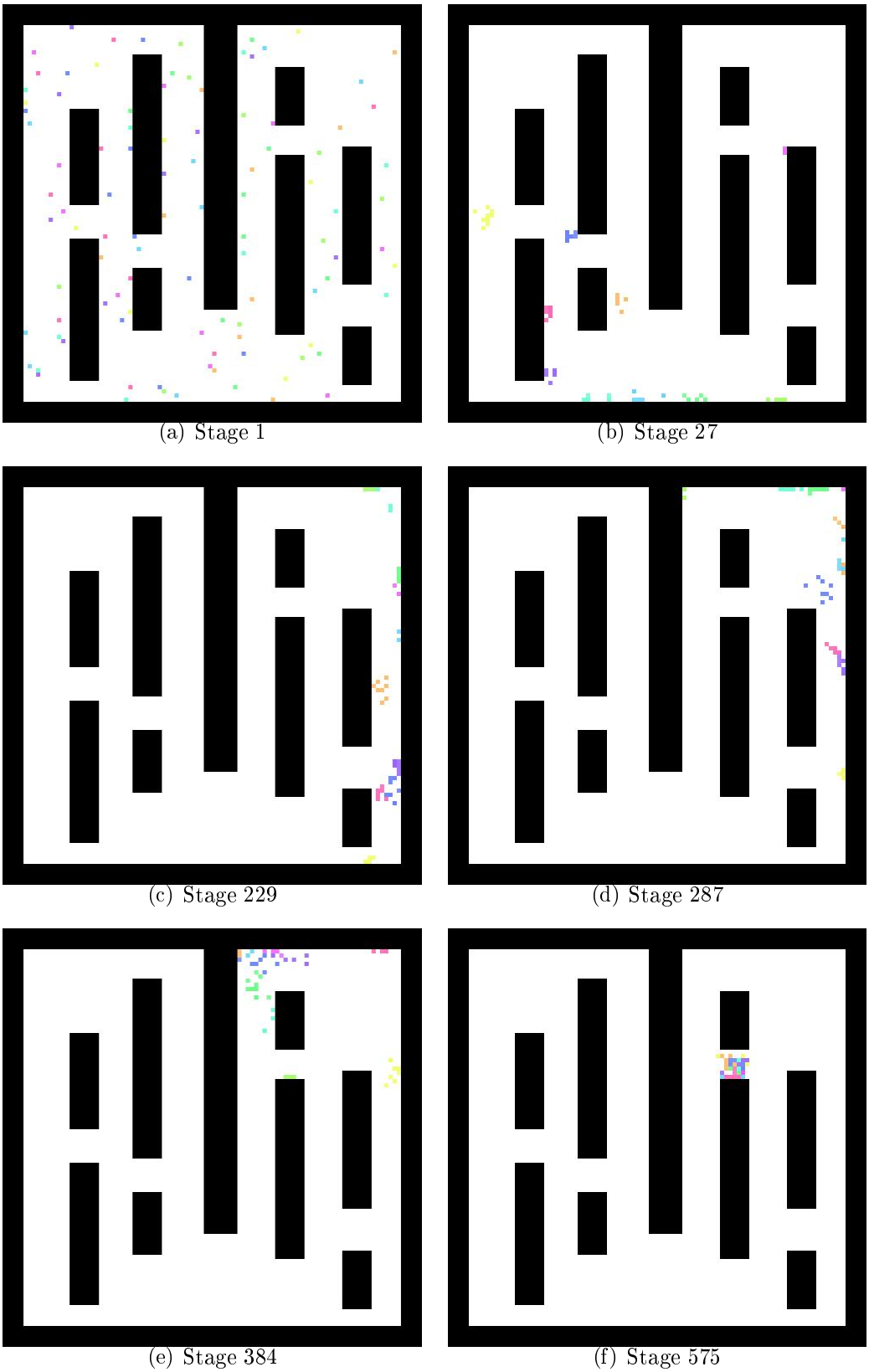


Figure 5.2: Event graph

when the predetermined maximum number of stages has elapsed. The goal of the policy is to direct the robot to the goal position $x_g = [33, 69]$. The event graph is illustrated in Figure 5.2. For each event in the figure the node indicates the sensing triggers and the type of event. The event triggers are any combination of North ("N"), South ("S"), East ("E") and West ("W") and the event type may be instantaneous and timed ("i"), accumulation and timed ("a"), or a timed ("t"). Essentially, the policy attempts to direct the robot southwards until it detects that it is in contact with a southern boundary. Upon triggering the event, the control policy directs the robot east until it detects it is at a eastern boundary. After detecting contact with the eastern boundary, the robot is directed to the northern boundary, the robot then moves toward to a western boundary. The robot then heads south for a fixed duration. Next, the robot is directed south-east with an open loop until it detects an eastern boundary. The robot then moves south until it perceives an open gap to the east. Finally, the robot attempts to center itself between a northern and southern boundary.

The results for two different simulations, which were generated with MATLAB, for this example are depicted in Figures 5.3 and 5.4. In the former, there are 10 hyper-particles each with 10 particles, whereas the later comprises 50 hyper-particles with 100

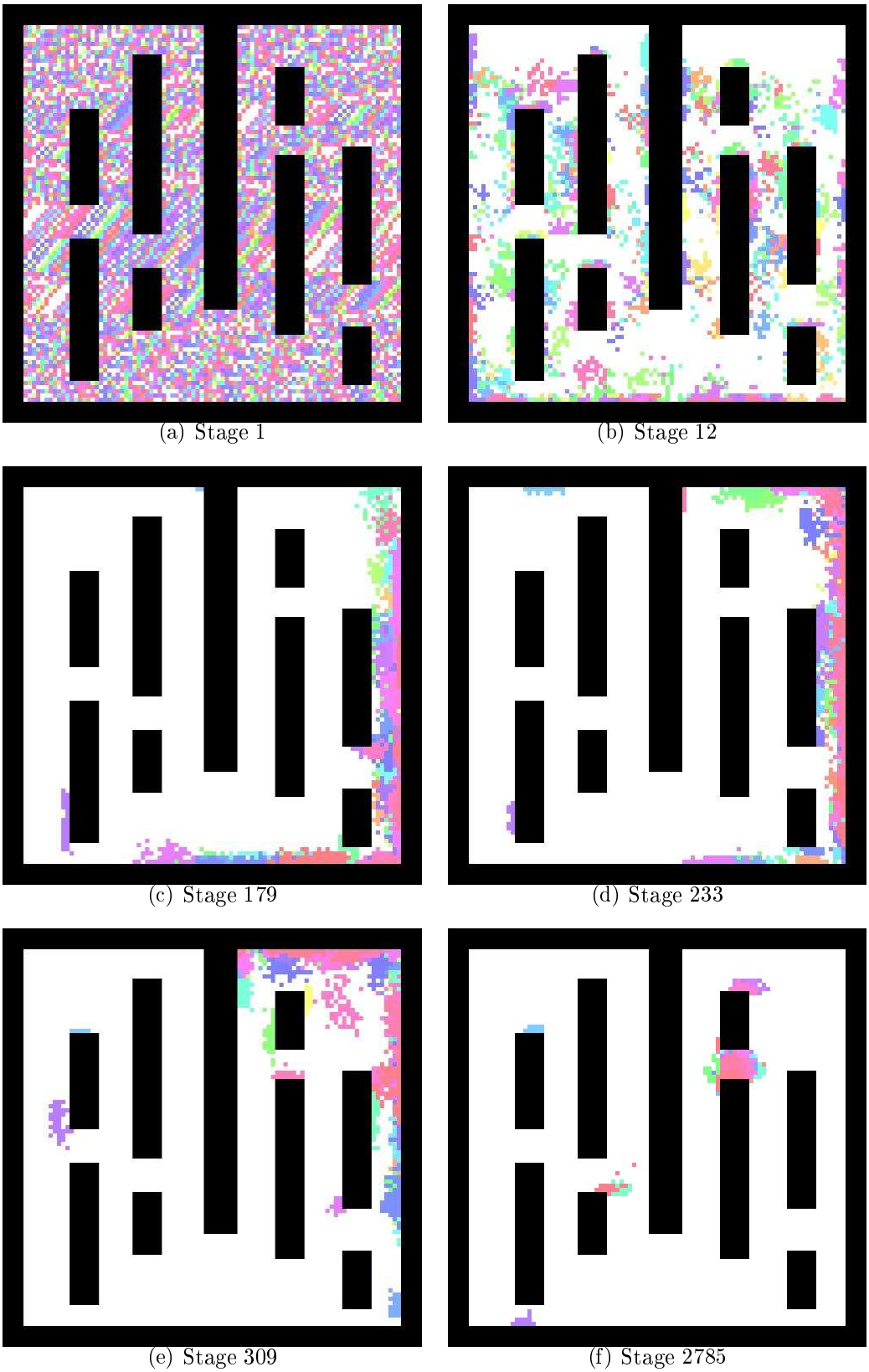


(a) Stage 1 (b) Stage 27

(c) Stage 229 (d) Stage 287

(e) Stage 384 (f) Stage 575

Figure 5.3: Results for the example: 10 hyper-particles



(a) Stage 1

(b) Stage 12

(c) Stage 179

(d) Stage 233

(e) Stage 309

(f) Stage 2785

Figure 5.4: Results for the example: 50 hyper-particles

particles a piece. For both simulations, each hyper-particle is represented over $\mathcal{W}_{free}$ as a set of particles that are colored to identify to which of the hyper-particles the particle set belongs. The evolution of the system through a progressive number of iterations is depicted by Figure 5.3(a)-(f) and Figure 5.4(a)-(f) for the example with 10 hyper-particles and 50 hyper-particles, respectively. Initially, the simulations begin with a uniform probability of being located anywhere in $\mathcal{W}_{free}$ as portrayed at Figure 5.3(a) and Figure 5.4(a). The process terminates as shown at Figure 5.3(f) and Figure 5.4(f) when each hyper-particle has reached the termination event, which may occur after a maximum number of iterations or when the hyper-particle triggers that the system is between a northern boundary and southern boundary.

In particular at Figure 5.3(a), the system is initialized and all the hyper-particle sets are in event e_1. Then at Figure 5.3(b) several of the hyper-particles transition to event e_2 and head toward the southeast corner of the state space. Having reached the northern corner at Figure 5.3(c) one of the hyper-particles transitions into e_4, while the all but one of the remaining particles are currently in event e_3. At Figure 5.3(d), several of the events have transitioned into event e_5 and are directed westward, while the remaining are in event e_3 and are still moving north. One of the hyper-particle sets is in event e_{10} and attempting to localize between the northern and southern boundaries at Figure 5.3(e). Also, in Figure 5.3(e) one hyper-particle set is in event e_8 moving to the southeast until it recognizes being at an eastern boundary. Still other of the hyper-particle sets are in event e_6, trying to localize the robot in the corner. Finally, all of the hyper-particles are in event e_{10} and terminate at Figure 5.3(f).

Because events may trigger when the accumulated probability reaches a threshold for the chosen policy, it is possible that even though the hyper-particle may not have reached the desired objective, the robot will be convinced it has with probability greater than the desired threshold. In such situations, the robot did not achieve the desired objective so that the event-based control policy directs the robot in an undesired manner

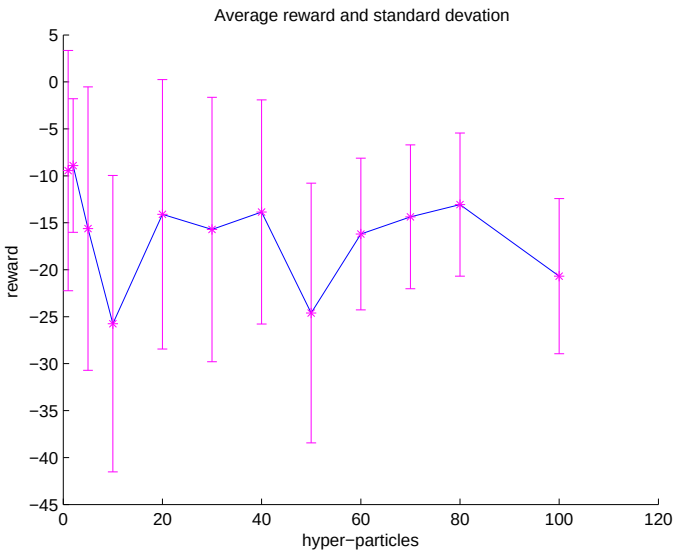


Figure 5.5: Results for a varying number of hyper-particles

so that the robot becomes lost and wanders through the space. This illustrates the limitations of the policy employed and not of the hyper-particle filtering method. What is important is that such unlikely events present themselves when a greater number of hyper-particles are employed to describe the system. Such situations, even though they happen with low probability, can have a dramatic impact on the performance of the system as each hyper-particle set that does not reach the goal is penalized with a lower reward. Thus, as the number of hyper-particles increase, the quality of the solution becomes more accurate even if the reward decreases.

To understand to what extent the accuracy of the predicted performance is affected, an examination of this example was performed in order to determine the effect on the performance as the number of particles and hyper-particles samples vary. The convex sum of the entropy (as described in [114]) and squared L2-norm from expected position to the goal x_g was chosen as the objective function because it incorporates both a measure of the uncertainty and a measure of the distance of the robot to the goal state.

The results illustrated in Figure 5.5 depict the average reward over the set of iterations as well as the standard deviation in the reward for a varying number of hyper-

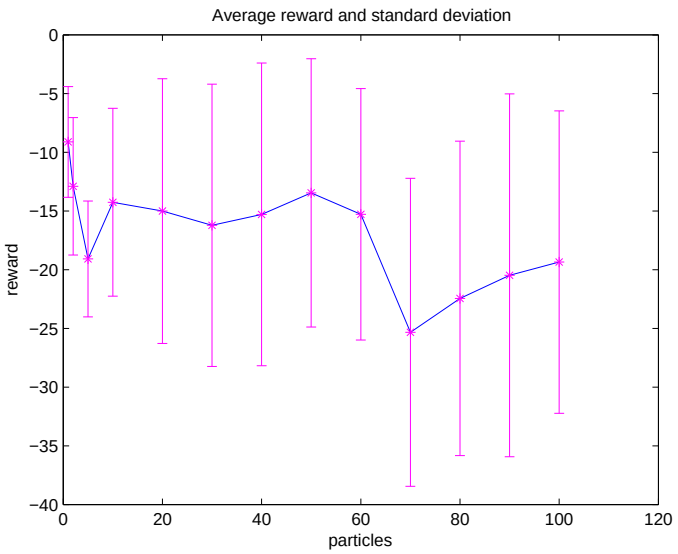


Figure 5.6: Results for a varying number of particles

particles, with each hyper-particle represented by a particle set with 100 particles. In Figure 5.6 the performance is evaluated for a fixed number of 50 hyper-particles and a variable number of particles represented each hyper-particle.

Interestingly, as can be seen in Figure 5.6, the cost increases as the number of particles increases. This result is logical considering the representation. When there are fewer particles, the probability function is represented by a small set of points, artificially making the probability function appear more condensed and, hence, more certain. However, as the number of samples increases, the representation of the probability function becomes more accurate and the uncertainty present in the system becomes better represented, which is reflected in the expected reward.

Because a significant amount of variation exists in the problem (as illustrated at Figure 5.4 by the probability functions at the terminal stage), the standard deviation is rather large and only appears to converge for more than 60 hyper-particles. However, as will be demonstrated below in Section 5.2, the hyper-particle filtering approach converges quickly as the number of hyper-particles increases and, more importantly,

the standard deviation of the hyperfiltering approach is significantly less than that of sample path approaches.

This example was chosen not only to demonstrate the effectiveness of the hyper-particle filtering method but to access its applicability in evaluating event-driven policies. While in this example the policy is given, future research will use hyperfiltering to aid in determining policies automatically. In future research event-driven policies, like the one used in this example, will be searched to find nearly optimal policies for POMDP systems.

Additionally, this example highlights the importance of information. In techniques where the future observation is not taken into account, the probability function over the space diffuses as time progresses. However, by taking into account the future unseen observations, that probability function over the space does not diffuse—at least as rapidly—in this example.

5.2 Comparison Examples

In addition to the example above, an exhaustive evaluation of the performance of the hyper-particle filtering method was performed for three representative problems. The problems vary dramatically in size:

- 4×4 grid: comprising a set of 16 possible states, a set of 2 possible observations, and a set of 4 possible actions
- *Hallway2*: comprising a set of 92 possible states, a set of 17 possible observations, and a set of 5 possible actions
- *Tag*: comprising a set of 870 possible states, a set of 30 possible observations, and a set of 5 possible actions

and were chosen to demonstrate how the hyper-particle filtering method performs as the size of the state space and observation space vary.

To obtain a meaningful analysis of the quality of the hyper-particle filtering technique, the performance of each example was evaluated relative to the optimal or nearly-optimal policy for the objective function associated with each problem. In each of these examples the objective function is the sum of a discounted expected total reward at each stage:

$$R(b_1) = \lim_{K \to \infty} E[\sum_{k=1}^{K} \gamma^k r(x_k, u_k) | y_1],$$

where $r(\cdot)$ is a state and control dependent reward function, γ is the discount factor, K is the time horizon (K is chosen to be large to approximate the infinite time horizon assumed for each of these problems). For the $4x4$ example, the optimal solution was found using the Incremental Pruning method described in [98] using pomdp-solve [115]. Unfortunately, the other examples (i.e., *Hallway2* and *Tag*) are too large to find exact solutions so an approximate solution was found using zmdp [116], which implements the HSVI algorithm of [71].

The performance is evaluated by finding the statistical average of the expected total reward and standard deviation of the expected total reward taken over series of simulations (30 iterations for 4×4 *and Hallway2* and 10 iterations for *Tag*). For the hyper-particle filtering method, the expected total reward is obtained by evaluating the expected value of the reward over the hyperbelief at each stage.

5.2.1 Comparison of the examples for a variety of hyper-particle and particle samples

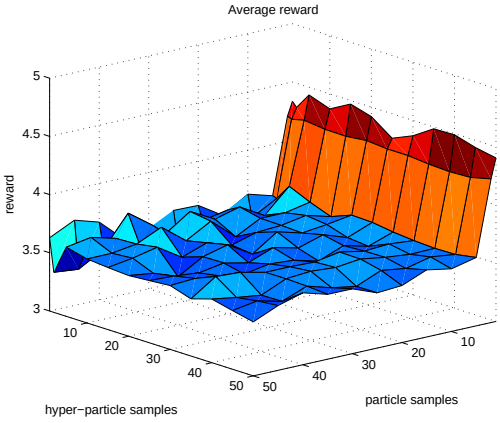
For each of the examples, a particle filtering approximation of the belief is employed and simulations are performed to compare the effect of particle filtering on the precision of result. This allows not only the effect of just the variation in the number of hyper-

particles to be fully evaluated but also the effect of the variation in the number of particles. As can be seen in Figure 5.7 the reward and variance quickly converge as the number of hyper-particles increase. The effect of the number of particles seems to be influential as well, but the impact/benefit drops off significantly as the number of particles increases. This may be a result of the small amount of noise the systems are subject to, unlike the example above in Section 5.1.

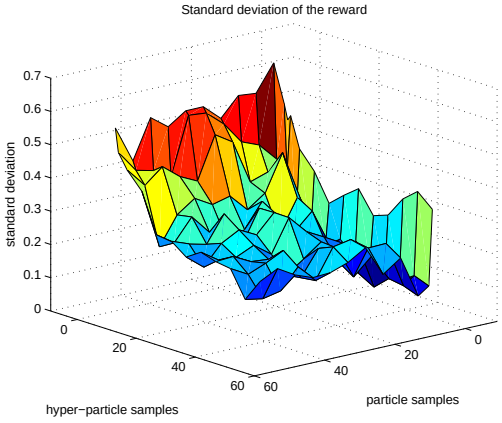
It is also interesting to note that the quality of the approximation stabilizes for just a few particles and hyper-particles. This may imply that, for problems with relatively small numbers of possible observations and states, only a small number of particles and hyper-particles are needed to obtain reasonable results. This contrasts dramatically with the example in Section 5.1, where there are a set of 10 000 possible states and a set of 328 420 possible observations for which the reward and variance did not display any substantial convergence until a relatively large number of hyper-particles and particles were used. This may be an indicator that the number of hyper-particles and particles needed are more of a function of the uncertainty than the size of the state and observation spaces.

5.2.2 Comparison of the hyper-particle filtering technique to sample path simulation

While hyper-particle filtering and sample path simulation have some aspects in common, i.e., randomly sampling beliefs, they are fundamentally different approaches. Moreover, hyperfiltering via hyper-particle filtering enables various other aspects of the prediction of the behavior or a system to be explored that sample path methods are incapable of performing. Namely, hyper-particle filtering enables a complete understanding of the behavior at each stage as well as how the behavior changes from one stage to the next. Regardless, a comparison can be made between the two when evaluating the performance of a system for a specific objective function at a specified time horizon.

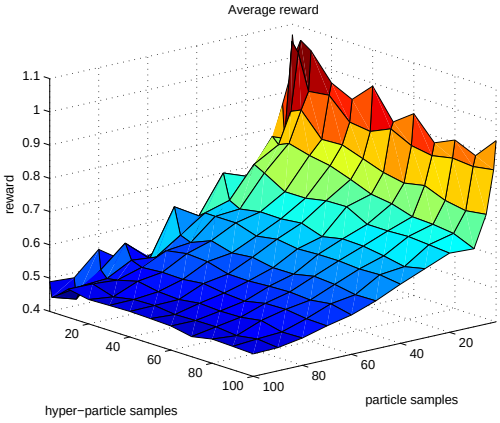


(a) 4×4 example

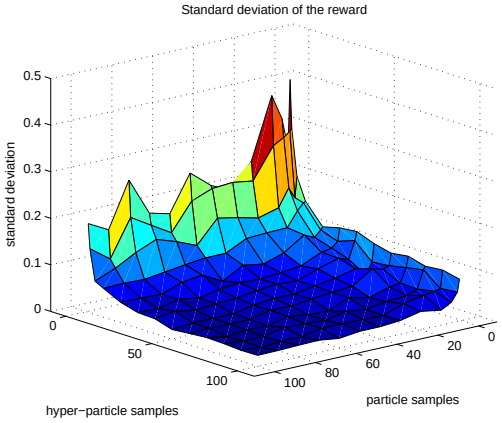


(b) 4×4 example

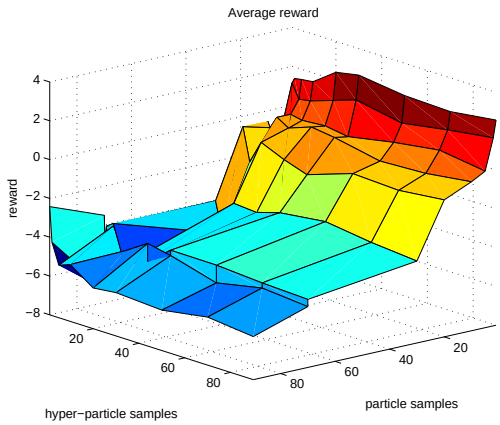
tb



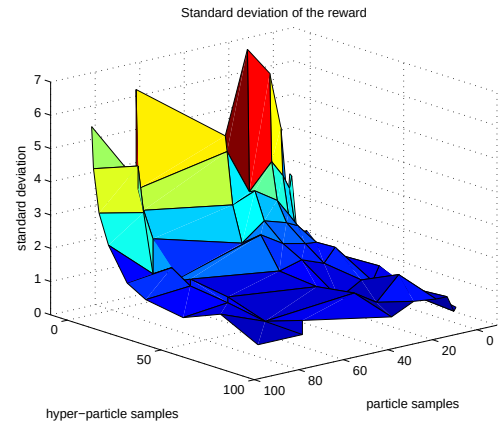
(c) Hallway2 example



(d) Hallway2 example

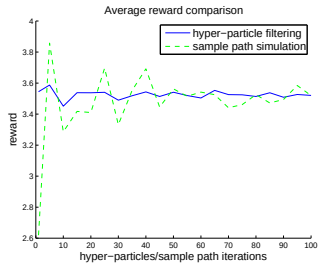


(e) Tag example

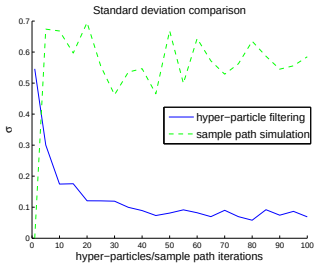


(f) Tag example

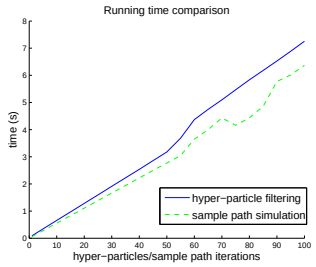
Figure 5.7: Average reward and standard deviation



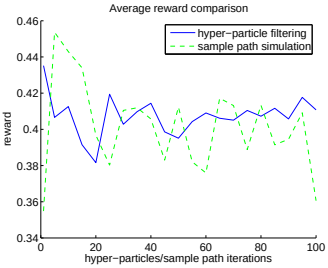
(a) Average reward for the 4×4 example



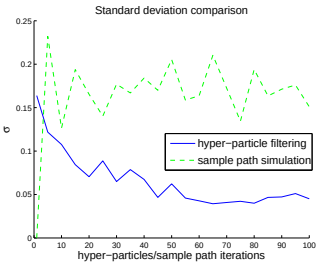
(b) Standard deviation of the reward for the 4×4 example



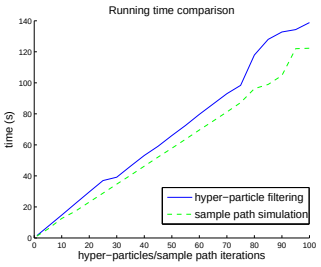
(c) Running time for the 4×4 example



(d) Average reward for the *Hallway2* example



(e) Standard deviation of the reward for the *Hallway2* example



(f) Running time for the *Hallway2* example

Figure 5.8: Comparison of sample path methods to hyper-particle filtering

Such analysis is used often in POMDP research to evaluate the performance of a policy. To reiterate, the hyperfilter is a tool developed for the sequential evaluation of the estimate of the system and its uncertainty from stage to stage, unlike sample path methods which are used to evaluate an objective function for a given time horizon. To make an equitable comparison, both approaches are evaluated based on results in conjunction with the amount of time to to evaluate the system for the specified time horizon. The results illustrated in Figure 5.8, where the full belief was used to represent each hyper-particle, clearly demonstrate the superiority of the hyper-particle filtering approach.

The reason for the impressive performance difference between a simple sample path evaluation and that of the hyper-particle filter is likely the product of two effects. First, the hyper-particle filter evaluates the reward over the approximated hyper-belief at each stage. The sample path method on the other hand just evaluates the average reward for each sample path. In this way, by considering a probability function over the belief

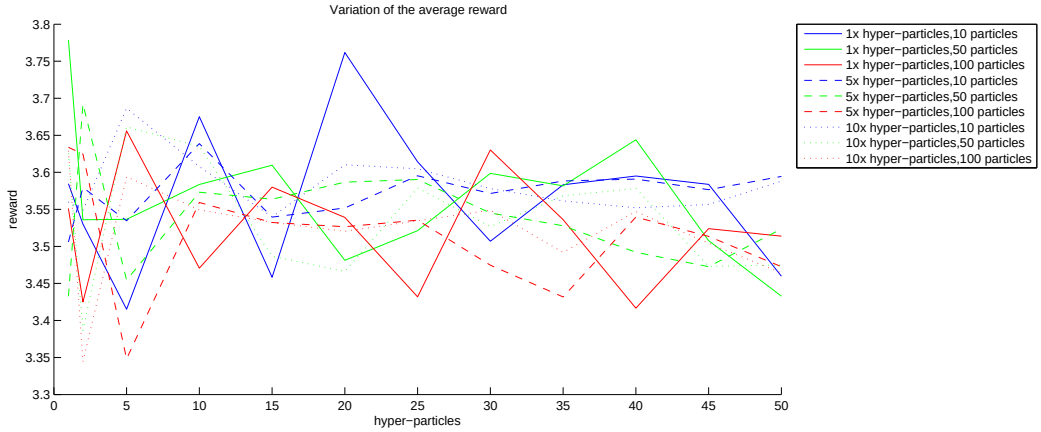
Table 5.1: Number of redundant hyper-particles

	5	10	20	30	40	50	60	70	80
4×4	3.17	4.97	5.17	5.37	5.57	5.77	5.97	6.17	6.37
Hallway2	1.23	3.12	7.03	12.37	17.27	22.88	29.17	35.19	43.17

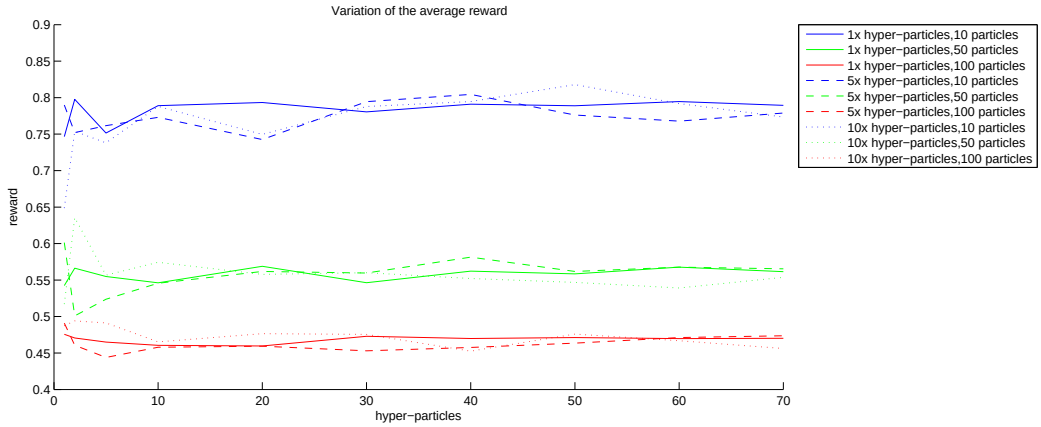
a more accurate representation can be found by taking the expected value over the beliefs instead of just taking the average of each sample in sample path approaches. Secondly, the hyper-particle filter evaluates the probability of each new hyper-particle by evaluating over the posterior given the approximated prior hyperbelief. By doing this, a more accurate representation of the posterior can be evaluated because the next stage hyper-particle samples are generated based on the posterior of the hyperbelief for which the redundancy in the hyper-particle has a significant impact. For instance, Table 5.1 shows the number of redundant hyper-particles. The number of redundant hyper-particles grows with the number of hyper-particles, implying that a more accurate representation of the posterior is generated as more likely beliefs are sampled more frequently, generating a higher weight, or probability, associated with the beliefs.

5.2.3 Comparison of the effect of the number of intermediate hyper-particle samples on the performance

For several of the examples, simulations were run for a variety of number of particles, hyper-particles, and intermediate hyper-particle samples. In each case the number of hyper-particles is fixed from stage to stage, so there is no growth in the number of hyper-particles at each iteration. However, there may be an increase in the number of intermediate hyper-particles between stages as a number of intermediate hyper-particles may be sampled for each hyper-particle in the current set to generate the future hyper-particle set (refer to Algorithm 4). Because a better representation of (4.18) should be attained with a greater number of intermediate hyper-particles, the influence of the number of intermediate hyper-particles was deemed worth investigating.

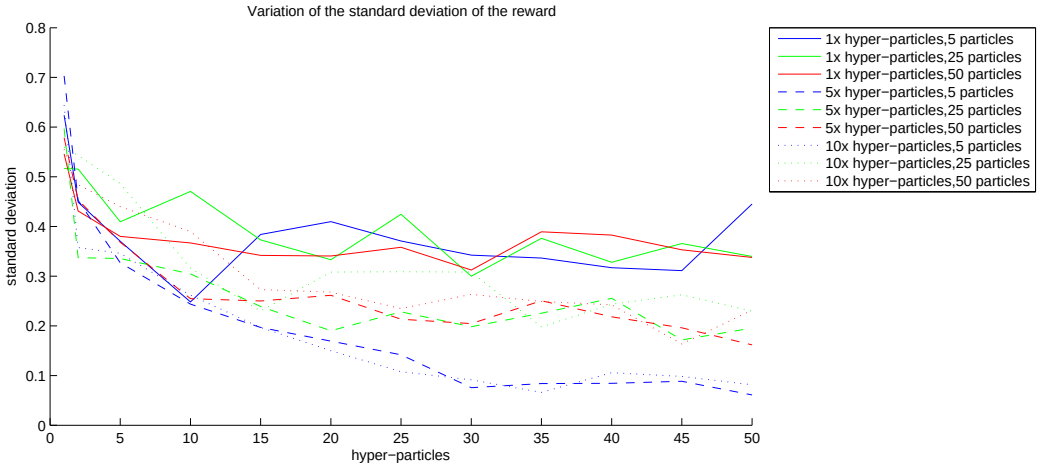


(a) Results for the 4×4 example

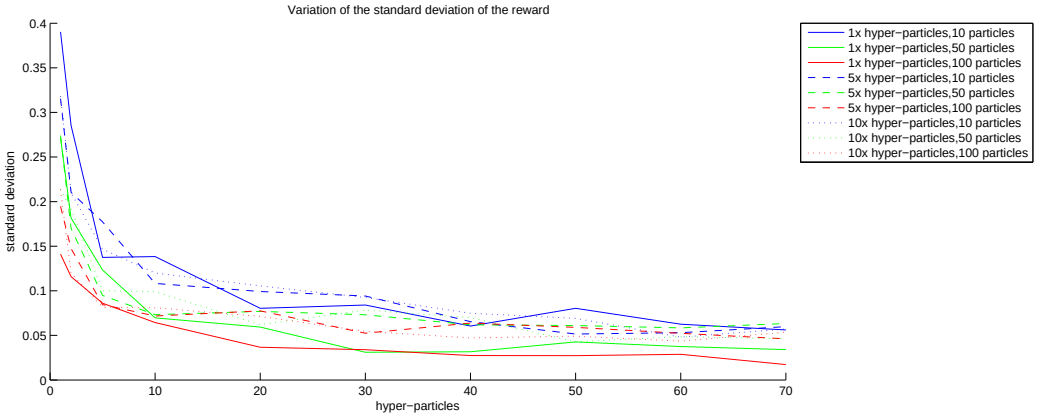


(b) Results for the *Hallway2* example

Figure 5.9: Average reward for a varying number of intermediate samples



(a) Results for the 4×4 example



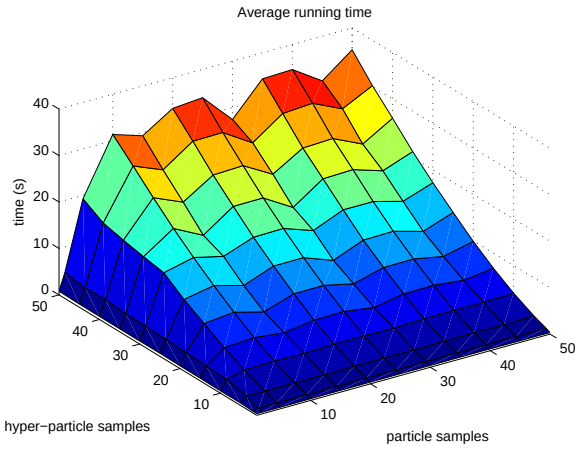
(b) Results for the *Hallway2* example

Figure 5.10: Standard deviation in the reward for a varying number of intermediate samples

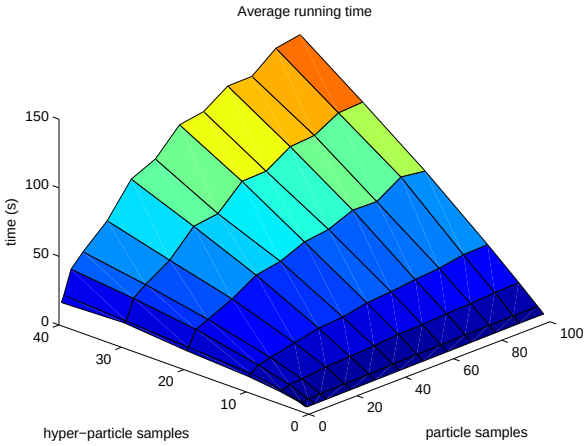
The results of the simulations of the 4×4 and *Hallway2* examples for various number of intermediate samples are depicted in Figures 5.9 and 5.10, where the former illustrates the average reward over the series of simulations, and the latter, the standard deviation of the reward over the series of simulations. In these examples, the beliefs are approximated by a particle set generated via the particle filtering method described in Section 3.3. As can be seen in the 4×4 example the standard deviation shows a modest improvement with the increased number of intermediate hyper-particles. However, in the *Hallway2* example, the number of intermediate hyper-particles appears to have little to no discernible impact. This is contrary to the initial hypothesis. Because the example problems are subject to only a small amount of noise, it is possible that the increase in the resolution of the probability function is insignificant and does greatly affect result of the resampling stage.

5.2.4 Experimental results on the running time of the hyper-particle filtering algorithm

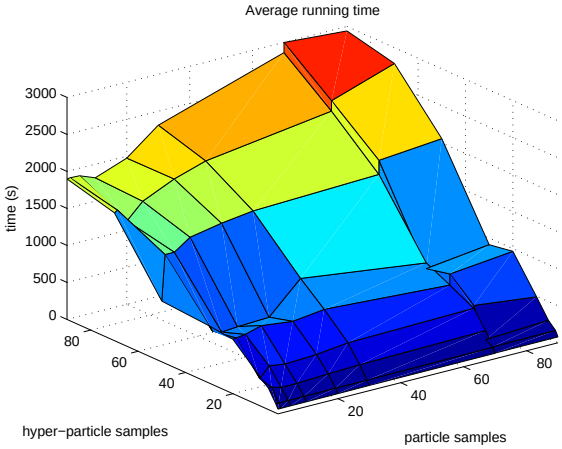
The average running time for each for the variety of number of hyper-particle samples, particle samples, and intermediate hyper-particle samples is illustrated in Figure 5.11. The running time grows linearly in both the number of hyper-particle samples as well as the particle samples confirming the running time established in Section 4.3.2.



(a) Results for the 4×4 example



(b) Results for the *Hallway2* example



(c) Results for the *tag* example

Figure 5.11: Average running time

6 CONCLUSION

6.1 Discussion

To my knowledge, this is the first attempt to provide a method to predict the effect of partially observed systems by a forward sequential method via the evolution of the probability function over the beliefs. As discussed in Section 4.1, there are numerous situations where such a formulation for the forward sequential synthesis of system are useful if not advantageous.

The various methods currently used in the literature either discard the observation process entirely (as done in [76, 78]) or perform sample path simulation, whereby a series of sample paths are generated and the ensemble average is taken to estimate the behavior of a system. Alternatively, the behavior is just predicted one state into the future using forward projection techniques (refer to [79]). Another method, which is popular in control theory for optimization is model predictive control, which predicts the evolution of the system without considering the effect of the uncertainty in the system (see [77]). To generate more robust plans, researchers employed minimax optimization by bounding the uncertainty and then optimizing the minimax reward around the expected result in [117–119]. These methods perform worst-case analysis through backward-based simulation, not forward-based simulation. Alternatively, for hybrid systems that are piecewise linear Gaussian systems, all the parameters relating to the evolution of a system up to a given stage can be explicitly represented (e.g., [120, 121]). As discussed previously, closed-form solutions for the representation from stage to stage are elusive for most problems. These techniques utilize multiparametric programming

to then find a control policy. Unfortunately, multiparametric programming is NP-Hard for general nonlinear problems.

Sample path simulation is a forward-based method, whereby a set of sample paths is generated from an initial stage to some specified future stage. However, this unappealing possibility generates geometric growth in the computational complexity for each additional stage evaluated, assuming there are only a finite number of sample paths. Evaluating every possible sample over again just to simulate the system an additional stage requires quadratic time complexity in the number of stages to be evaluated. Thus this iterative approach is redundant and wasteful: none of the results are retained from stage to stage.

Avoiding redundancy in the reevaluation of the system from stage to stage is paramount if a method is to be developed that proceeds in an efficient manner. A reasonable alternative to this iterative scheme is to pursue a forward-based sequential method. By maintaining a representation of the probability function defined over the belief at each stage, hyperfiltering is such a sequential forward-based method.

However, while reducing the redundant reevaluation from one stage to the next, the representation of the probability function over the belief may require a finite number of parameters, which can potentially grow as the product of the current number of parameters and the number of observations at each stage. While hyperfiltering eliminates one source of the exponential growth in the computational time complexity, the growth in the number of elements that represent the probability function between stages can become burdensome requiring a computational time complexity that is exponential for a given time horizon. Hyperfiltering has the potential to evaluate systems where there are an uncountable number of sample paths.

Sample path simulation methods alleviate this computational burden by simulating only a finite set of the possible sample paths to evaluate the ensemble average behavior of the system, yet, this method itself is not performed sequentially; the method still

is redundant, requiring the evaluation of all previous stages to simulate an additional stage and still has an exponential time complexity in the time horizon. Moreover, is a does it represent the probabilistic evolution from one stage to the next.

The hyper-particle filtering approximation method is introduced to reduce the computational burden that arises from the potential growth in the number of parameters representing the probability function over the belief. Hyper-particle filtering is based on particle filtering, however, hyper-particle filtering approximates the probability function over the belief space, whereas particle filtering approximates the evolution of a probability function over the state space.

Hyper-particle filtering is more than just a sequential implementation of sample path simulation. Hyper-particle filtering samples from the posterior over the beliefs generated from the approximated hyperbelief. Sample path simulation, on the other hand, samples a single observation from each prior belief. By sampling from the posterior over the belief, a more accurate representation of the next stages hyperbelief can be obtained. While in some respects, sample path simulation and hyper-particle filtering appear similar, they are fundamentally different approaches. Hyper-particle filtering is an approach to approximating the probability function over the belief at each stage, whereas sample path simulation generates a series of sample paths to a given stage. Still, there are some algorithmic similarities: both generate random samples and both are forward-based methods.

By treating the sample set as an approximation over the belief space at the current stage and sampling from the posterior at each stage, the hyper-particle filter reduces degeneracy and focuses the sampled set on the more likely future beliefs. Sample path simulation methods lack this feature entirely and any unlikely sample paths are never eliminated. Moreover, when analyzing the performance for a given objective function, the reward at each stage is taken as the expectation over the hyper-particle set, unlike sample path methods, where the reward for each simulation is the reward

of the path. Because of these differences, the hyper-particle filter outperforms the sample path approximation technique even when considering the performance relative to a given objective at a specific horizon as outlined above as shown in Section 5.2. Moreover, hyper-particle filtering achieves this with a computational time complexity less than or equal to sample path methods for simulating the evolution to a particular stage–depending on the sampling function employed.

Recently, researchers have sampled the belief space to obtain feasible/reachable beliefs to reduce the computational burden of finding nearly optimal policies (e.g., [69–72, 101]). Relating more specifically to the hyper-particle filtering approximation, these approaches sample a feasible set of the belief space by generating a random set of belief samples in the belief space. However, these methods neglect to retain probabilities associated with each sample and instead are used to discretize the belief space into a meaningful finite set, they also fail to address the fundamental sequential nature that arises when a probability over the belief is evaluated forward over multiple stages. Like these methods, however, hyperfiltering only requires the effect of the possible beliefs as represented by the hyperbelief at each iteration. This is an important point and is one of the motivations for this research.

6.2 Potential Applications

By applying hyperfiltering, control policies can be more accurately assessed and can be evaluated from on future state to the next. These aspects of hyperfiltering may prove useful for a variety of applications.

By evaluating the evolution of the hyperbelief instead of a specific measure, it is possible to understand the full effect a policy has on the evolution of a system. Moreover, the interesting possibility exists to use hyperfiltering to enhance the planning currently achieved by model predictive control techniques, such as [118–121]. By generating the

hyper-belief at each stage a more accurate measurement of the performance can be estimated for a given time horizon.

The importance of information in planning and estimation has been understood for quite some time (as discussed in [32,88]). However, only recently have researchers begun to address to what extent and to what degree information is useful (e.g., [17, 106, 122]). These researchers are attempting to understand the effect of sensing limitations on the ability of a robot to achieve certain tasks as well as what tasks can be achieved for given sensing capabilities. Hyperfiltering is an ideal tool for analyzing these issues for stochastic systems. By representing the probability function over the beliefs at each stage, hyperfiltering can be employed to analyze the effect of observations on evolution of the system. In addition, hyperfiltering can be used through algorithmic analysis to answer queries regarding how sensing capabilities affect the evolution or even to discern which sensing configurations are sufficient to obtain a desired objective.

Besides the potential application of determining the utility of information, the hyperfiltering technique may be useful in finding nearly optimal solutions for POMDP systems. There are various methods to approximate POMDPs to find nearly optimal solutions by reducing the set of possibilities explored including: considering only a subset of the points in belief space that need to be searched and by considering only a subset of the possible policies. In either or both scenarios, a method to propagate the estimate of the belief and its uncertainty from point to point may be useful as the hyperfiltering method can propagate the estimate of the belief and its uncertainty between belief points and/or evaluate the finite set of possible policies in a policy iteration framework.

6.3 Final Remarks

Hyperfiltering, a method for predicting the evolution of a stochastic partially observed Markov decision processes forward to future stages, was presented. Hyperfiltering is a sequential method that estimates the probability function over the belief forward one stage to the next for a given policy. Because of computational issues as well as the highly nonlinear belief transition probability function, the hyper-particle filtering method was introduced to approximate the exact hyperfiltering method.

Numerous simulations of the hyper-particle filtering approximation method were performed to verify its performance over a variety of parameters. The results indicated that the method converges quickly in both mean and standard deviation as the number samples representing the probability function over the belief space increases. In fact, it appeared that very few hyper-particles are needed, relative to the number of observations and iterations, to achieve an accurate analysis of the system. Additionally, the simulations verified the computational complexity as being $O(Knq)$, where K is the desired time-horizon, n is the number samples representing the probability function over the belief space, and q is the number of samples representing a belief.

The hyperfiltering method itself is not a method to generate policies; its function is only to evaluate the effect of a given policy. However, one of the eventual goals is to utilize hyperfiltering to generate a hybrid-control/event-based technique to generate nearly optimal policies for stochastic problems with high dimensional closed and bounded continuous nonsimple state spaces. Another avenue of research is to incorporate the hyperfiltering method with limited-look-ahead planning methods like model predictive control. The current state of the art for these methods fail to generate a method to represent the possibility over all unseen observations and instead employs overly simplified techniques to bound the error on the system.

The hyperfiltering method was also created to be used to understand the value of information when planning. Because of the ability to sequentially evaluate the evolution of the system for future unseen observations, at each stage the impact of an observation can be evaluated and, thus, the role the observations have in evolution of a system can be analyzed directly.

REFERENCES

[1] R. Paul, *Robot Manipulators: Mathematics, Programming and Control.* Cambridge, MA: MIT Press, 1981.

[2] C. Lee, R. C. Gonzales, and K. S. Fu, *Tutorial on Robotics.* Silver Spring, MD: IEEE Computer Society Press, 1983.

[3] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence.* New York, NY: McGraw-Hill Book Co., 1987.

[4] B. K. P. Horn, *Robot Vision.* Cambridge, MA: MIT Press, 1986.

[5] R. Bajcsy, "Active perception," *Proceedings of the IEEE*, vol. 78, pp. 996–1005, 1988.

[6] J. Crowley, "World modeling and position estimation for a mobile robot using ultrasound ranging," in *IEEE International Conference on Robotics and Automation*, 1989, pp. 674–680.

[7] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal of Robotics and Automation*, vol. RA-3, pp. 249–265, June 1987.

[8] K. Konolige, "Markov localization using correlation," in *Proceedings International Joint Conference on Artificial Intelligence*, 1999, pp. 1154–1159.

[9] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo localization for mobile robots," in *IEEE International Conference on Robotics and Automation (ICRA99)*, May 1999.

[10] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, no. 1-2, pp. 99–141, 2000.

[11] J. Spletzer and C. Taylor, "A bounded uncertainty approach to multi-robot localization," in *IEEE International Conference on Intelligent Robots and Systems*, 2003, pp. 1258–1265.

[12] K. Arras, N. Tomatis, B. Jensen, and R. Siegwart, "Multisensor on-the-fly localization: Precision and reliability for applications," *Robotics and Autonomous Systems*, vol. 34, no. 2-3, pp. 131–143, 2001.

[13] L. J. Guibas, R. Motwani, and P. Raghavan, "The robot localization problem," in *Algorithmic Foundations of Robotics*, 1995, pp. 269–282.

[14] W. Burgard, D. Fox, and S. Thrun, "Active mobile robot localization by entropy minimization," in *Proceedings of the EUROMICRO Second Workshop on Advanced Mobile Robots*, 1997, pp. 155–162.

[15] G. Dudek, K. Romanik, and S. Whitesides, "Global localization: Localizing a robot with minimal travel," *SIAM Journal on Computing*, vol. 27, no. 2, pp. 583–604, April 1998.

[16] J.-S. Gutmann, T. Weigel, and B. Nebel, "A fast, accurate, and robust method for self-localization in polygonal environments using laser-range-finders," *Advanced Robotics*, vol. 14, no. 8, pp. 651–668, 2001.

[17] J. O'Kane. and S. LaValle, "Almost-sensorless localization," in *IEEE International Conference on Robotics and Automation*, 2005, pp. 3764–3769.

[18] R. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–86, 1987.

[19] H. Durrant-Whyte, "Uncertain geometry in robotics," in *IEEE International Conference on Robotics and Automation*, vol. 4, 1987, pp. 851–856.

[20] N. Ayache and O. Faugeras, "Maintaining representations of the environment of a mobile robot," in *IEEE International Conference on Robotics and Automation*, 1989, pp. 804–819.

[21] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," *Autonomous Robot Vehicles*, vol. 4, pp. 167–193, 1990.

[22] J. Castellanos, J. Tardos, and G. Schmidt, "Building a global map of the environment of a mobile robot: The importance of correlations," in *IEEE International Conference on Robotics and Automation*, 1997, pp. 1053–1059.

[23] S. Thrun, D. Fox, and W. Burgard, "Probabilistic mapping of an environment by a mobile robot," in *IEEE International Conference on Robotics and Automation*, 1998, pp. 1546–1551.

[24] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, June 2001.

[25] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and N. A.Y., "Simultaneous mapping and localization with sparse extended information filters," in *International Workshop on the Algorithmic Foundations of Robotics*, 2003, pp. 363–380.

[26] Y. Liu and S. Thrun, "Results for outdoor-SLAM using sparse extended information filters," in *IEEE International Conference on Robotics and Automation*, 2003, pp. 1227–1233.

[27] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2002.

[28] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping," in *Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[29] F. Masson, J. Guivant, and E. Nebot, "Hybrid architecture for simultaneous localization and map building in large outdoor areas," in *IEEE International Conference on Intelligent Robots and Systems*, 2002, pp. 570–575.

[30] H. Choset and K. Nagatani, "Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 2, pp. 125–137, April 2001.

[31] A. Vale and M. I. Ribeiro, "Environment mapping as a topological representation," in *IEEE International Conference on Advanced Robotics*, 2003, pp. 29–34.

[32] D. P. Bertsekas, *Dynamic Programming and Optimal Control.* Cambridge, MA: Athena Scientific, 2007.

[33] S. Loo, "Statistical design of optimum multivariable systems whose inputs are corrupted by multiplicative noise," *IEEE Transactions on Automatic Control*, vol. 13, no. 3, pp. 300–301, June 1968.

[34] D. Kleinman, "Optimal stationary control of linear systems with control-dependent noise," *IEEE Transactions on Automatic Control*, vol. 14, no. 6, pp. 673–677, Dec. 1969.

[35] P. McLane, "Optimal stochastic control of linear systems with state- and control-dependent disturbances," *IEEE Transactions on Automatic Control*, vol. 16, no. 6, pp. 793–798, Dec. 1971.

[36] R. Rana and A. Mahalanabis, "On optimal stationary control of systems with state dependent noise," *IEEE Transactions on Automatic Control*, vol. 20, no. 5, pp. 718–719, Oct. 1975.

[37] A. Bagchi and T. Schilperoort, "Optimal linear stochastic control for systems with multiplicative noise," *IEEE Transactions on Automatic Control*, vol. 25, no. 5, pp. 1005–1007, Oct. 1980.

[38] R. Mohler and W. Kolodziej, "Optimal control of a class of nonlinear stochastic systems," *IEEE Transactions on Automatic Control*, vol. 26, no. 5, pp. 1048–1054, Oct. 1981.

[39] Y. Phillis, "Controller design of systems with multiplicative noise," *IEEE Transactions on Automatic Control*, vol. 30, no. 10, pp. 1017–1019, Oct. 1985.

[40] Y. Phillis, "Estimation and control of systems with unknown covariance and multiplicative noise," *IEEE Transactions on Automatic Control*, vol. 34, no. 10, pp. 1075–1078, Oct. 1989.

[41] J. Riordon, "Optimal feedback characteristics from stochastic automaton models," *IEEE Transactions on Automatic Control*, vol. 14, no. 1, pp. 89–92, Feb. 1969.

[42] M. Rami, X. Chen, and X. Zhou, "Discrete-time indefinite LQ control with state and control dependent noises," in *IEEE Conference on Decision and Control*, 2001, pp. 1249–1250.

[43] E. Todorov, "Stochastic optimal control and estimation methods adapted to the noise characteristics of the sensorimotor system," *Neural Computation*, vol. 17, no. 5, pp. 1084–1108, 2005.

[44] R. E. Shostak, "On the SUP-INF method for proving Presburger formulas," *Journal of the Association for Computing Machinery*, vol. 24, no. 4, pp. 529–543, 1977.

[45] A. Ambler and R. Popplestone, "Inferring the positions of bodies from specified spatial relationships," *Artificial Intelligence*, vol. 6, pp. 157–174, 1975.

[46] R. A. Brooks, "Symbolic error analysis and robot planning," *International Journal of Robotics Research*, vol. 1, no. 4, pp. 29–78, Winter 1982.

[47] J. Pertin-Troccaz and P. Puget, "Dealing with uncertainties in robot planning using program proving techniques," in *Fourth International Symposium on Robotic Research*, Aug. 1987.

[48] R. E. Kalman, "A new approach to filtering and prediction problems," *Journal of Basic Engineering*, vol. 82D, pp. 35–45, 1960.

[49] R. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986.

[50] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," *Autonomous Robot Vehicles*, vol. 4, pp. 167–193, 1990.

[51] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor, "Automatic synthesis of fine-motion strategies for robots," *International Journal of Robotics Research*, vol. 3, no. 1, pp. 3–24, Spring 1984.

[52] M. Erdmann, "On motion planning with uncertainty," Master's thesis, Massachusetts Institute of Technology, 1986.

[53] B. R. Donald, "A geometric approach to error detection and recovery for robot motion planning with uncertainty," *Artificial Intelligence*, vol. 37, no. 1-3, pp. 223–271, Dec. 1988.

[54] B. R. Donald, "Planning multi-step error detection and recovery strategies," *International Journal of Robotics Research*, vol. 9, no. 1, pp. 3–60, Feb. 1990.

[55] J. Latombe, A. Lazanas, and S. Shekhar, "Robot motion planning with uncertainty in control and sensing," *Artificial Intelligence*, vol. 52, pp. 1–47, 1991.

[56] A. Doucet, N. de Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*. New York, NY: Springer Verlag, 2001.

[57] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 50, no. 2, pp. 174–188, February 2002.

[58] N. Gordon, D. Salmond, and A. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *Radar and Signal Processing, IEE Proceedings F*, vol. 140, no. 2, pp. 107–113, 1993.

[59] C. Kwok, D. Fox, and M. Meila, "Real-time particle filters," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 469–484, March 2004.

[60] S. Thrun, "Particle filters in robotics," in *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence*, 2002.

[61] D. Fox, "Adapting the sample size in particle filters through kld-sampling," *International Journal of Robotics Research (IJRR)*, vol. 22, pp. 985–1003, 2003.

[62] J. H. Kotecha and P. M. Djuric, "Gaussian sum particle filtering," *IEEE Transactions on Signal Processing*, vol. 51, pp. 2602–2612, Oct. 2003.

[63] M. Isard and A. Blake, "Condensation – conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, no. 1, pp. 5–28, 1998. [Online]. Available: citeseer.ist.psu.edu/isard98condensation.html

[64] A. Doucet, "On sequential simulation-based methods for Bayesian filtering," Department of Engineering, University of Cambridge, Tech. Rep. CUED/F-INFENG, TR. 310, 1998.

[65] D. Crisan, P. D. Moral, and T. Lyons, "Discrete filtering using branching and interacting particle systems," *Markov Processes and Related Fields*, vol. 5, no. 3, pp. 293–318, 1999.

[66] K. Kanazawa, D. Koller, and S. Russell, "Stochastic simulation algorithms for dynamic probabilistic networks," in *Proceedings of the 11th Annual Conference on Uncertainty in Artifical Intelligence*, 1995, pp. 346–351.

[67] D. Crisan and A. Doucet, "Convergence of sequential Monte Carlo methods," Cambridge University, Tech. Rep. CUED/FINFENG, TR381, 2000.

[68] Z. Feng and E. Hansen, "Approximate planning for factored POMDPs," in *Proceedings of the Sixth European Conference on Planning (ECP-01)*, 2001, pp. 409–416.

[69] D. Bernstein, E. Hansen, and S. Zilberstein, "Bounded policy iteration for decentralized POMDPs," in *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2005, pp. 52–57.

[70] S. Thrun, "Monte Carlo POMDPs," in *Advances in Neural Information Processing Systems*, 2000, pp. 1064–1070.

[71] T. Smith and R. Simmons, "Heuristic search value iteration for POMDPs," in *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 2004, pp. 520–527.

[72] T. Smith and R. Simmons, "Point-based POMDP algorithms: Improved analysis and implementation," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2005.

[73] N. Roy and G. Gordon, "Exponential family PCA for belief compression in POMDPs," in *Advances in Neural Information Processing Systems*, 2002, pp. 1–8.

[74] M. T. Spaan and N. Vlassis, "A point-based POMDP algorithm for robot planning," in *IEEE International Conference on Robotics and Automation*, 2004, pp. 2399–2404.

[75] M. T. Spaan and N. Vlassis, "PERSEUS: Randomized point-based value iteration for POMDPs," in *Journal of Artificial Intelligence Research*, vol. 24, 2005, pp. 195–220.

[76] M. Erdmann and M. Mason, "An exploration of sensorless manipulation," *IEEE Journal of Robotics and Automation*, vol. 4, no. 4, pp. 369–379, 1988.

[77] M. Morari and J. Lee, "Model predictive control: Past, present, and future," *Computers and Chemical Engineering*, vol. 23, pp. 667–682, 1997.

[78] G. Shani, R. I. Brafman, and S. E. Shimony, "Forward search value iteration for POMDPs," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2007.

[79] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: The MIT Press, 2005.

[80] H. W. Sorenson, *Kalman Filtering: Theory and Applications*. New York, NY: IEEE Press, 1985.

[81] D. L. Alspach and H. W. Sorenson, "Nonlinear Bayesian estimation using Gaussian sum approximation," *IEEE Transactions on Automatic Control*, vol. 17, no. 4, pp. 439–448, 1972. [Online]. Available: http://www.cs.rpi.edu/ isler/new/pub/pubs/tr-04-13.pdf

[82] S. Julier and J. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, March 2004.

[83] U. Hanebeck, "Recursive nonlinear set-theoretic estimation based on pseudo ellipsoids," in *International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2001, pp. 159–164.

[84] E. Stump, B. Grocholsky, and V. Kumar, "Extensive representations and algorithms for nonlinear filtering and estimation," in *International Workshop on the Algorithmic Foundations of Robotics*, 2006.

[85] U. D. Hanebeck, K. Briechle, and A. Rauh, "Progressive Bayes: A new framework for nonlinear state estimation," in *Proceedings of SPIE*, 2003, pp. 256–267.

[86] V. M. Klumpp, D. Brunn, and U. D. Hanebeck, "Approximate nonlinear Bayesian estimation based on lower and upper densities," in *The 9th International Conference on Information Fusion*, 2006, pp. 1–8.

[87] X. Boyen and D. Koller, "Tractable inference for complex stochastic processes," in *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 1998, pp. 33–42.

[88] K. J. Astrom, "Optimal control of Markov decision processes with incomplete state estimation," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 174–205, 1965.

[89] M. L. Littman, "Algorithms for sequential decision making," Ph.D. dissertation, Brown University, 1996.

[90] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2003.

[91] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, Jan. 1998.

[92] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable Markov processes over a finite horizon," *Operations Research*, vol. 21, no. 5, pp. 1071–1088, Sep. 1973.

[93] E. Sondik, "The optimal control of partially observable Markov processes," Ph.D. dissertation, Stanford University, 1971.

[94] E. J. Sondik, "The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs," *Operations Research*, vol. 26, no. 2, pp. 282–304, March 1978.

[95] G. E. Monahan, "A survey of partially observable Markov decision processes: Theory, models, and algorithms," *Management Science*, vol. 28, no. 1, pp. 1–16, Jan. 1982.

[96] H. T. Cheng, "Algorithms for partially observable Markov decision processes," Ph.D. dissertation, University of British Columbia, Vancouver, BC, Canada, 1988.

[97] N. L. Zhang and W. Liu, "Planning in stochastic domains: Problem characteristics and approximation," Department of Computer Science, Hong Kong University of Science and Technology, Tech. Rep. HKUST-CS96-31, 1996.

[98] A. Cassandra, M. L. Littman, and N. L. Zhang, "Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes," in *Proceedings of Uncertainty in Artificial Intelligence*, 1997, pp. 54–61.

[99] D. A. McAllester and S. Singh, "Approximate planning for factored POMDPs using belief state simplification," in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 1999, pp. 409–416.

[100] P. Poupart and C. Boutilier, "Value directed compression of POMDPs," in *Advances in Neural Information Processing Systems*, 2003.

[101] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).*, 2003, pp. 1025 – 1032.

[102] E. A. Hansen, "An improved policy iteration algorithm for partially observable MDPs," in *Advances in Neural Information Processing Systems*, vol. 10, 1998, pp. 1015–1021.

[103] E. Hansen, "Solving POMDPs by searching in policy space," in *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, 1998, pp. 211–21.

[104] N. Meuleau, L. Peshkin, K.-E. Kim, and L. P. Kaelbling, "Learning finite-state controllers for partially observable environments," in *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, 1999, pp. 427–436.

[105] P. Poupart and C. Boutilier, "Bounded finite state controllers," in *In Advances in Neural Information Processing Systems (NIPS)*, 2003.

[106] M. L. Littman, "Memoryless policies: Theoretical limitations and practical results," in *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, 1994, pp. 238–247.

[107] S. M. LaValle, *Planning Algorithms*. Cambridge, MA: Cambridge University Press, 2006.

[108] X. Boyen and D. Koller, "Exploiting the architecture of dynamic systems," in *Proceedings of the 16th National Conference on Artificial Intelligence*, 1999, pp. 313–320.

[109] M. Huber, D. Brunn, and U. D. Hanebeck, "Closed-form prediction of nonlinear dynamic systems by means of Gaussian mixture approximation of the transition density," in *International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2006, pp. 98–103.

[110] J. Li and A. Barron, "Mixture density estimation," in *Advances in Neural Information Processing Systems*, 2000, pp. 279–285.

[111] W. Lovejoy, "Computationally feasible bounds for partially observed Markov decision processes," in *Operations Research*, vol. 39, 1991, pp. 162–175.

[112] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[113] R. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME, Journal of Basic Engineering*, vol. 82, pp. 35–45, March 1960.

[114] R. B. Ash, *Information Theory*. New York, NY: Dover Publications, 1990.

[115] T. Cassandra, "POMDP solver software: pomdp-solve v. 5.3," 2007. [Online]. Available: http://pomdp.org/pomdp/code/index.shtml

[116] T. Smith, "ZMDP software for POMDP and MDP planning: ZMDPv. 1.1.3," 2007. [Online]. Available: http://www.cs.cmu.edu/ trey/zmdp/

[117] E. Kerrigan and D. Mayne, "Optimal control of constrained, piecewise affine systems with bounded disturbances," in *IEEE Conference on Decision and Control*, 2002, pp. 1552–1557.

[118] W. Langson, I. Chryssochoos, S. V. Rakovic, and D. Q. Mayne, "Robust model predictive control using tubes," *Automatica*, vol. 40, no. 9, pp. 125–133, Jan. 2004.

[119] P. J. Goulart, E. C. Kerrigan, and J. M. Maciejowski, "Optimization over state feedback policies for robust control with constraints," *Automatica*, vol. 42, pp. 523–533, April 2006.

[120] A. Bemporad, F. Borrelli, and M. Morari, "Model predictive control based on linear programming the explicit solution," *IEEE Transactions on Automatic Control*, vol. 47, no. 12, pp. 1974–1985, Dec. 2002.

[121] A. Bemporad, F. Borrelli, and M. Morari, "Min-max control of constrained uncertain discrete-time linear systems," *IEEE Transactions on Automatic Control*, vol. 48, no. 9, pp. 1600–1606, Sept. 2003.

[122] J. M. O'Kane and S. M. LaValle, "On comparing the power of mobile robots," in *Proceedings of Robotics: Science and Systems*, 2006.